Chapter 6

Graph Algorithms

Algorithm Theory
WS 2019/20

Fabian Kuhn

Circulations with Demands

Given: Directed network $G = (V, E)$ with

- Edge capacities $c_e > 0$ for all $e \in E$
- Node demands $d_v \in \mathbb{R}$ for all $v \in V$
  - $d_v > 0$: node needs flow and therefore is a sink
  - $d_v < 0$: node has a supply of $-d_v$ and is therefore a source
  - $d_v = 0$: node is neither a source nor a sink

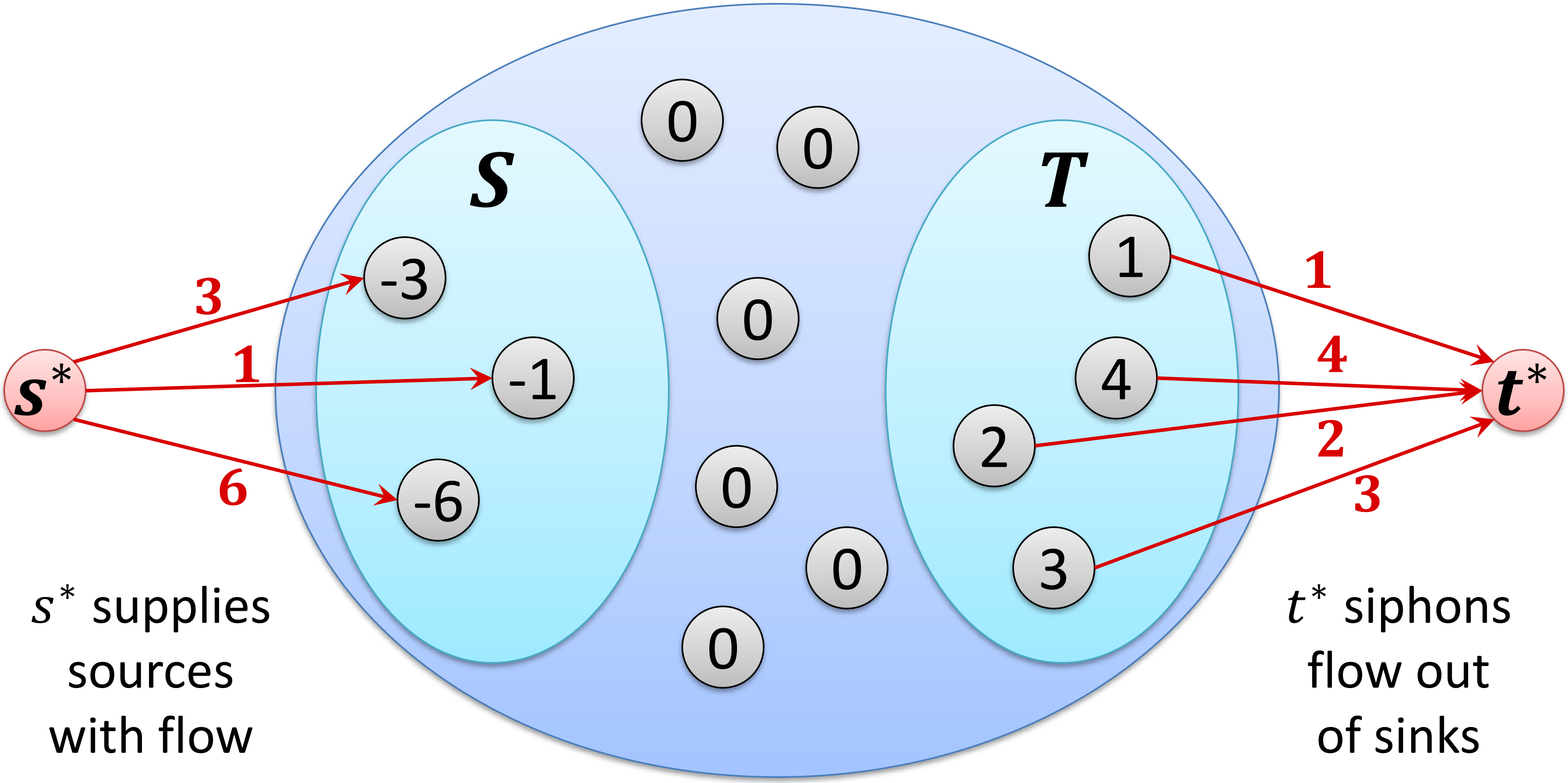
Flow: Function $f: E \to \mathbb{R}_{\geq 0}$ satisfying

- *Capacity Conditions*: $\forall e \in E$: $0 \leq f(e) \leq c_e$
- *Demand Conditions*: $\forall v \in V$: $f^{\text{in}}(v) - f^{\text{out}}(v) = d_v$

Objective: Does a flow f satisfying all conditions exist?
If yes, find such a flow f.

Reduction to Maximum Flow

- Add "super-source" s^* and "super-sink" t^* to network



s^* supplies sources with flow

t^* siphons flow out of sinks

Circulation: Demands and Lower Bounds

Given: Directed network $G = (V, E)$ with

- Edge capacities $c_e > 0$ and **lower bounds $\mathbf{0} \leq \boldsymbol{\ell_e} \leq \boldsymbol{c_e}$ for $\boldsymbol{e} \in \boldsymbol{E}$**
- Node demands $d_v \in \mathbb{R}$ for all $v \in V$
  - $d_v > 0$: node needs flow and therefore is a sink
  - $d_v < 0$: node has a supply of $-d_v$ and is therefore a source
  - $d_v = 0$: node is neither a source nor a sink

Flow: Function $f: E \to \mathbb{R}_{\geq 0}$ satisfying

- *Capacity Conditions*: $\forall e \in E$: $\boldsymbol{\ell_e \leq f(e) \leq c_e}$
- *Demand Conditions*: $\forall v \in V$: $f^{\text{in}}(v) - f^{\text{out}}(v) = d_v$

Objective: Does a flow f satisfying all conditions exist?
If yes, find such a flow f.

Solution Idea

- Define initial circulation $f_0(e) = \ell_e$
  Satisfies capacity constraints: $\forall e \in E: \ell_e \leq f_0(e) \leq c_e$

- Define

$$L_v \coloneqq f_0^{\text{in}}(v) - f_0^{\text{out}}(v) = \sum_{e \text{ into } v} \ell_e - \sum_{e \text{ out of } v} \ell_e$$

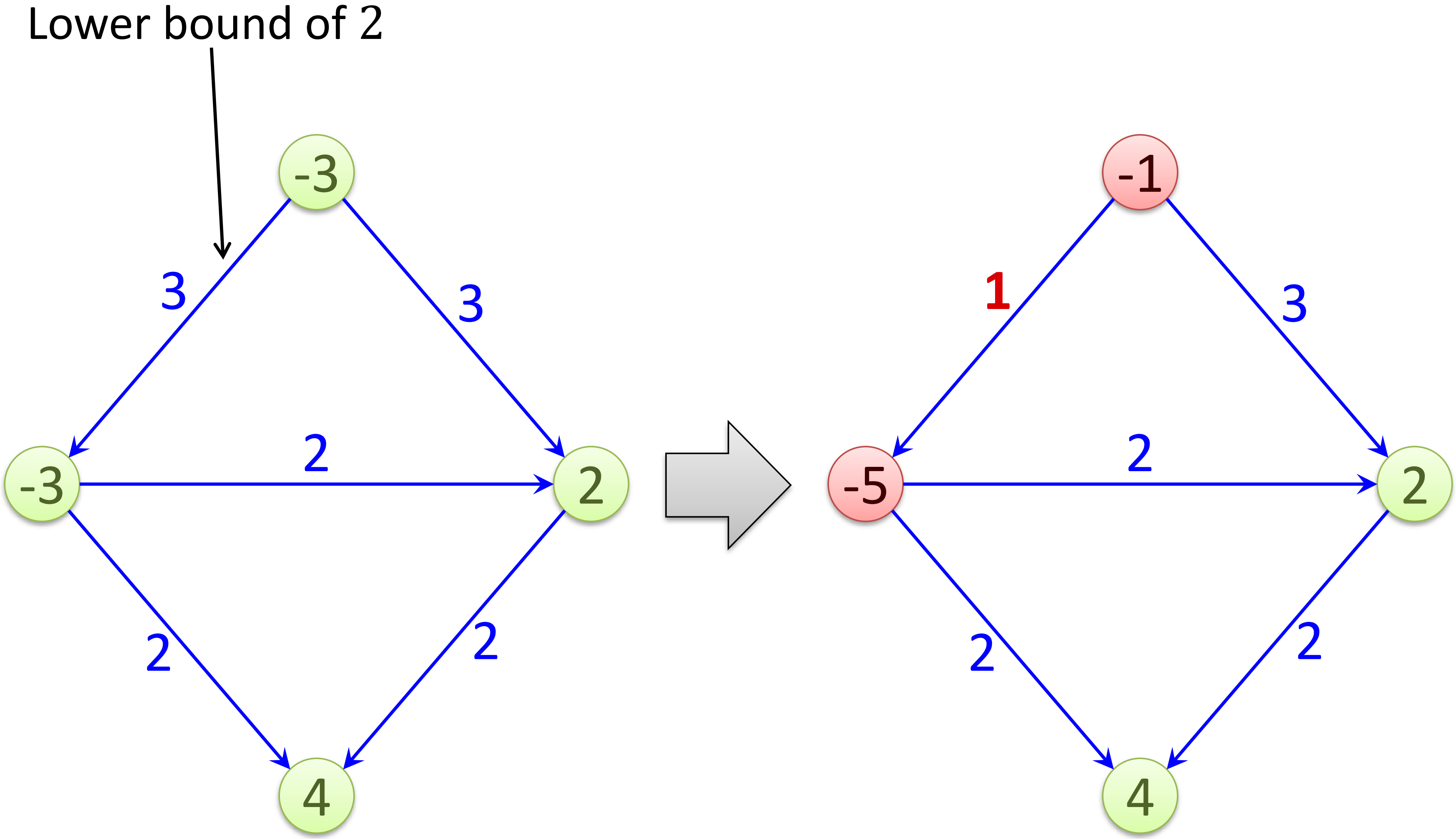
- If $L_v = d_v$, demand condition is satisfied at v by f_0, otherwise, we need to superimpose another circulation f_1 such that

$$d'_v \coloneqq f_1^{\text{in}}(v) - f_1^{\text{out}}(v) = d_v - L_v$$

- Remaining capacity of edge e: $c'_e \coloneqq c_e - \ell_e$

- We get a circulation problem with new demands d'_v, new capacities c'_e, and no lower bounds

Eliminating a Lower Bound: Example

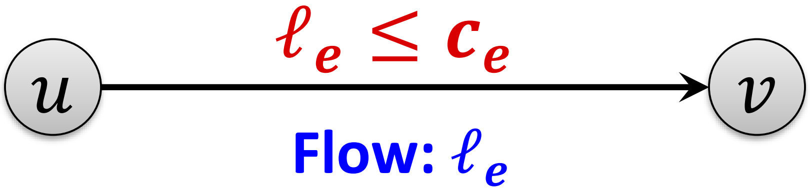


Reduce to Problem Without Lower Bounds

Graph $\boldsymbol{G = (V, E)}$:

- Capacity: For each edge $e \in E$: $\ell_e \le f(e) \le c_e$
- Demand: For each node $v \in V$: $f^{\text{in}}(v) - f^{\text{out}}(v) = d_v$

Model lower bounds with supplies & demands:



Create Network $\boldsymbol{G'}$ (without lower bounds):

- For each edge $e \in E$: $c'_e = c_e - \ell_e$
- For each node $v \in V$: $d'_v = d_v - L_v$

Circulation: Demands and Lower Bounds

Theorem: There is a feasible circulation in G (with lower bounds) if and only if there is feasible circulation in G' (without lower bounds).

- Given circulation f' in G', $f(e) = f'(e) + \ell_e$ is circulation in G
  - The capacity constraints are satisfied because $f'(e) \leq c_e - \ell_e$
  - Demand conditions:

$$f^{\text{in}}(v) - f^{\text{out}}(v) = \sum_{e \text{ into } v} \big(\ell_e + f'(e)\big) - \sum_{e \text{ out of } v} \big(\ell_e + f'(e)\big)$$

$$= L_v + (d_v - L_v) = d_v$$

- Given circulation f in G, $f'(e) = f(e) - \ell_e$ is circulation in G'
  - The capacity constraints are satisfied because $\ell_e \leq f(e) \leq c_e$
  - Demand conditions:

$$f'^{\text{in}}(v) - f'^{\text{out}}(v) = \sum_{e \text{ into } v} \big(f(e) - \ell_e\big) - \sum_{e \text{ out of } v} \big(f(e) - \ell_e\big)$$

$$= d_v - L_v$$

Integrality

Theorem: Consider a circulation problem with integral capacities, flow lower bounds, and node demands. If the problem is feasible, then it also has an integral solution.

Proof:

- Graph G' has only integral capacities and demands
- Thus, the flow network used in the reduction to solve circulation with demands and no lower bounds has only integral capacities
- The theorem now follows because a max flow problem with integral capacities also has an optimal integral solution

- It also follows that with the max flow algorithms we studied, we get an integral feasible circulation solution.

Matrix Rounding

- **Given:** $p \times q$ matrix $D = \{d_{i,j}\}$ of real numbers
- **row *i* sum:** $a_i = \sum_j d_{i,j}$, **column *j* sum:** $b_j = \sum_i d_{i,j}$
- **Goal:** Round each $d_{i,j}$, as well as a_i and b_j up or down to the next integer so that the sum of rounded elements in each row (column) equals the rounded row (column) sum
- **Original application:** publishing census data

Example:

3.14	6.80	7.30	17.24
9.60	2.40	0.70	12.70
3.60	1.20	6.50	11.30
16.34	10.40	14.50	

original data

3	7	7	17
10	2	1	13
3	1	7	11
16	10	15	

possible rounding

Matrix Rounding

Theorem: For any matrix, there exists a feasible rounding.

Remark: Just rounding to the nearest integer doesn't work

0.35	0.35	0.35	1.05
0.55	0.55	0.55	1.65
0.90	0.90	0.90	

original data

0	0	0	0
1	1	1	3
1	1	1	

rounding to nearest integer

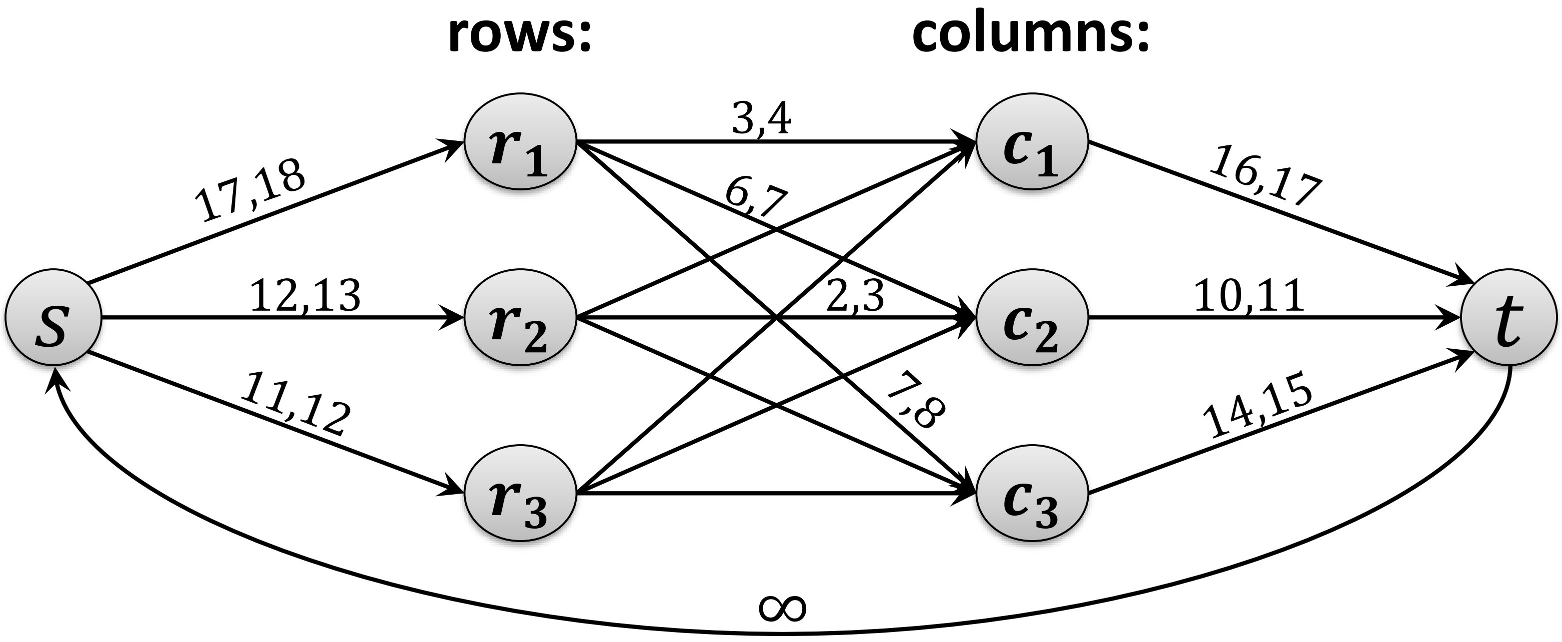
0	0	1	1
1	1	0	2
1	1	1	

feasible rounding

Reduction to Circulation

3.14	6.80	7.30	17.24
9.60	2.40	0.70	12.70
3.60	1.20	6.50	11.30
16.34	10.40	14.50	

Matrix elements and row/column sums give a feasible circulation that satisfies all lower bound, capacity, and demand constraints



all demands $d_v = 0$

Matrix Rounding

Theorem: For any matrix, there exists a feasible rounding.

Proof:

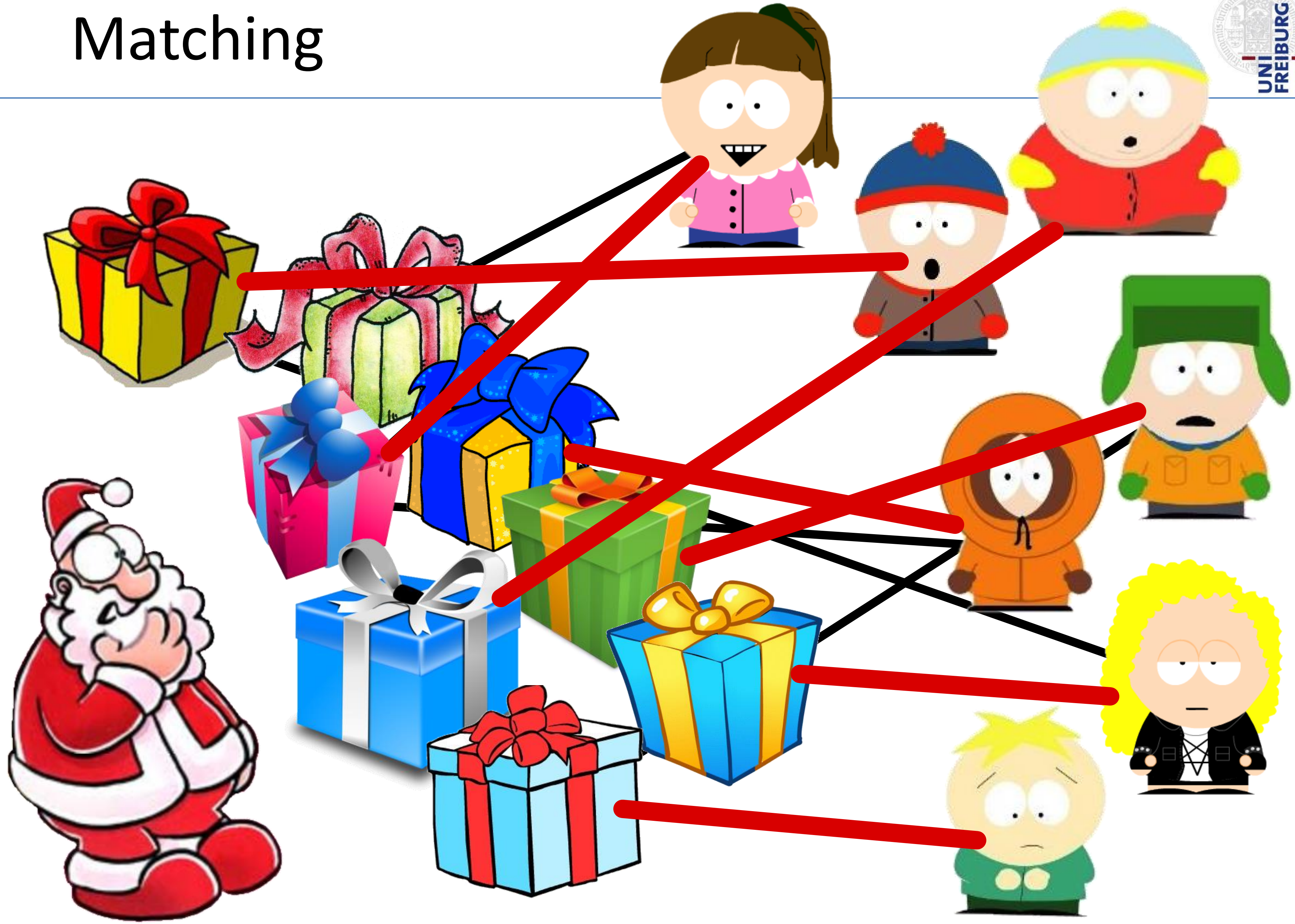
- The matrix entries $d_{i,j}$ and the row and column sums a_i and b_j give a feasible circulation for the constructed network

- Every feasible circulation gives matrix entries with corresponding row and column sums (follows from demand constraints)

- Because all demands, capacities, and flow lower bounds are integral, there is an integral solution to the circulation problem

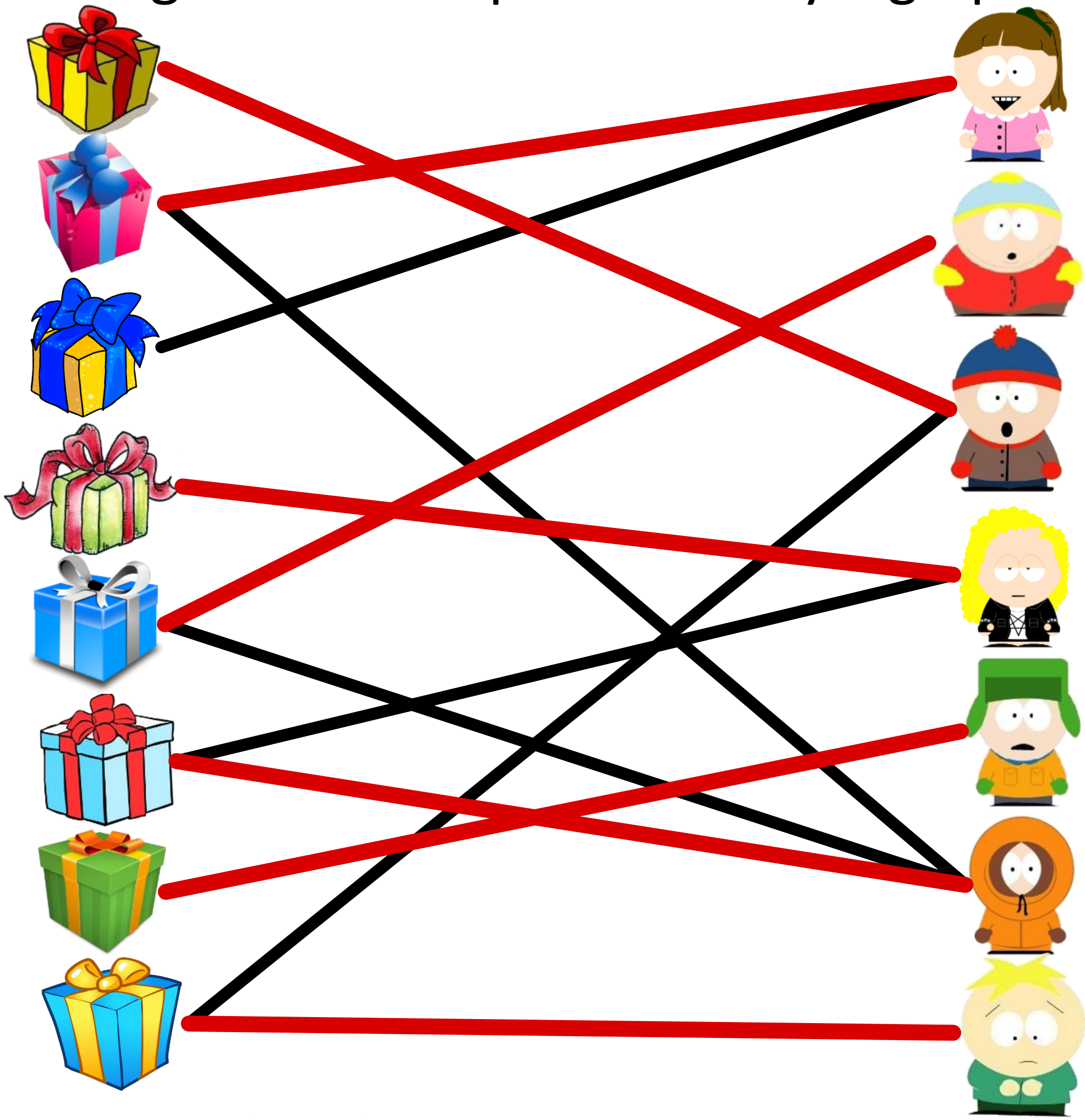
→ **gives a feasible rounding!**

Matching



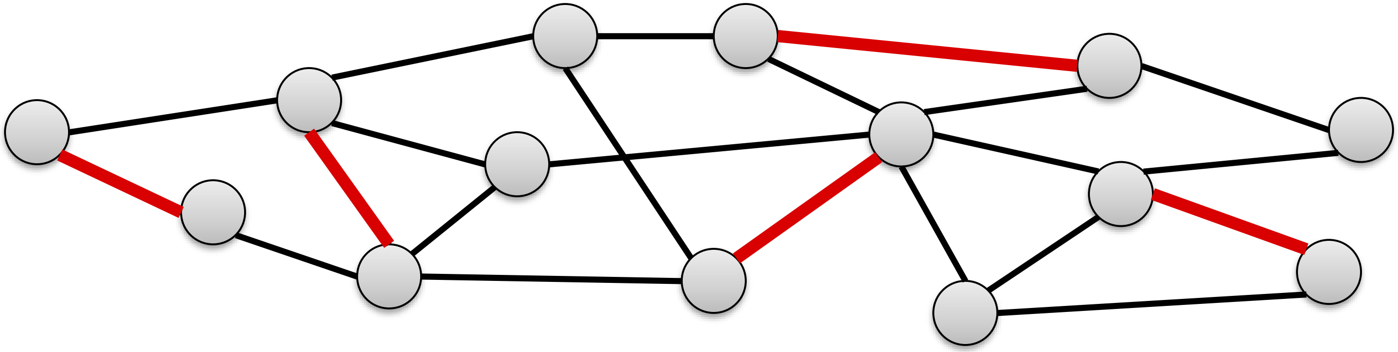
Gifts-Children Graph

- Which child likes which gift can be represented by a graph



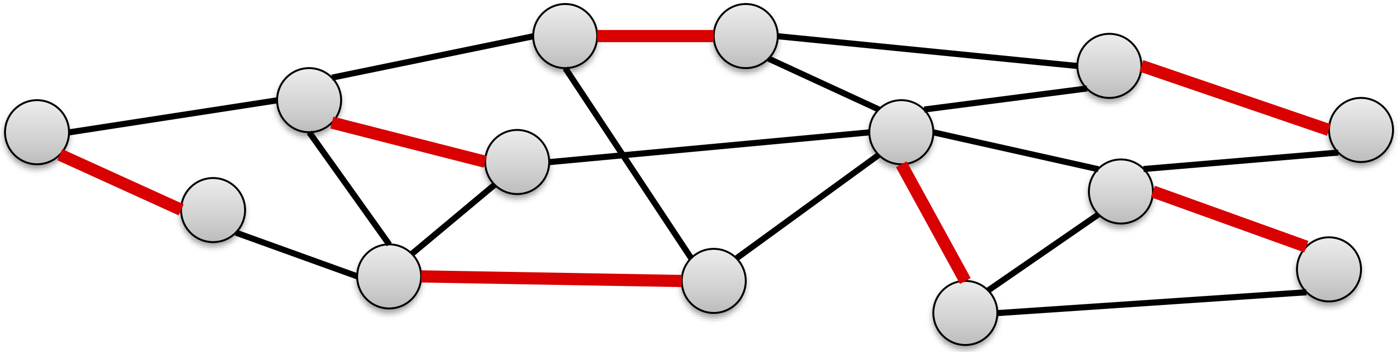
Matching

Matching: Set of pairwise non-incident edges



Maximal Matching: A matching s.t. no more edges can be added

Maximum Matching: A matching of maximum possible size



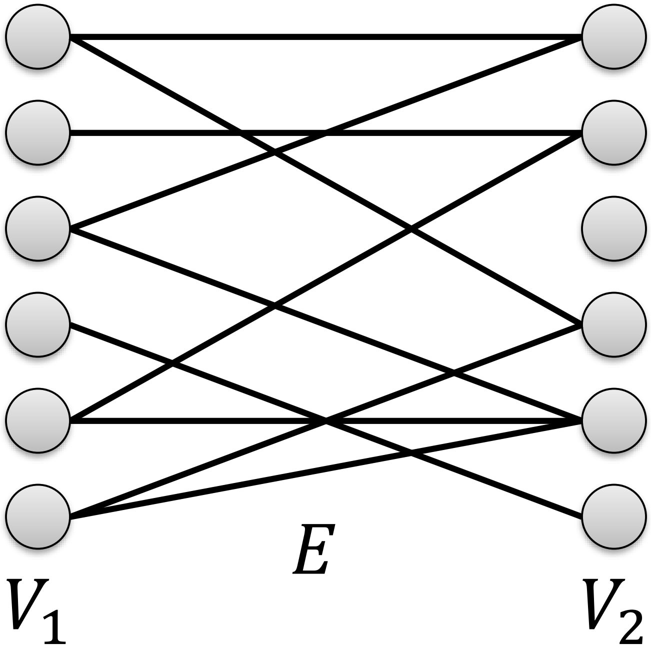
Perfect Matching: Matching of size ${}^{n}/_{2}$ (every node is matched)

Bipartite Graph

Definition: A graph $G = (V, E)$ is called bipartite iff its node set can be partitioned into two parts $V = V_1 \dot{\cup} V_2$ such that for each edge $\{\mathrm{u}, \mathrm{v}\} \in E$,

$$|\{u, v\} \cap V_1| = 1.$$

- Thus, edges are only between the two parts



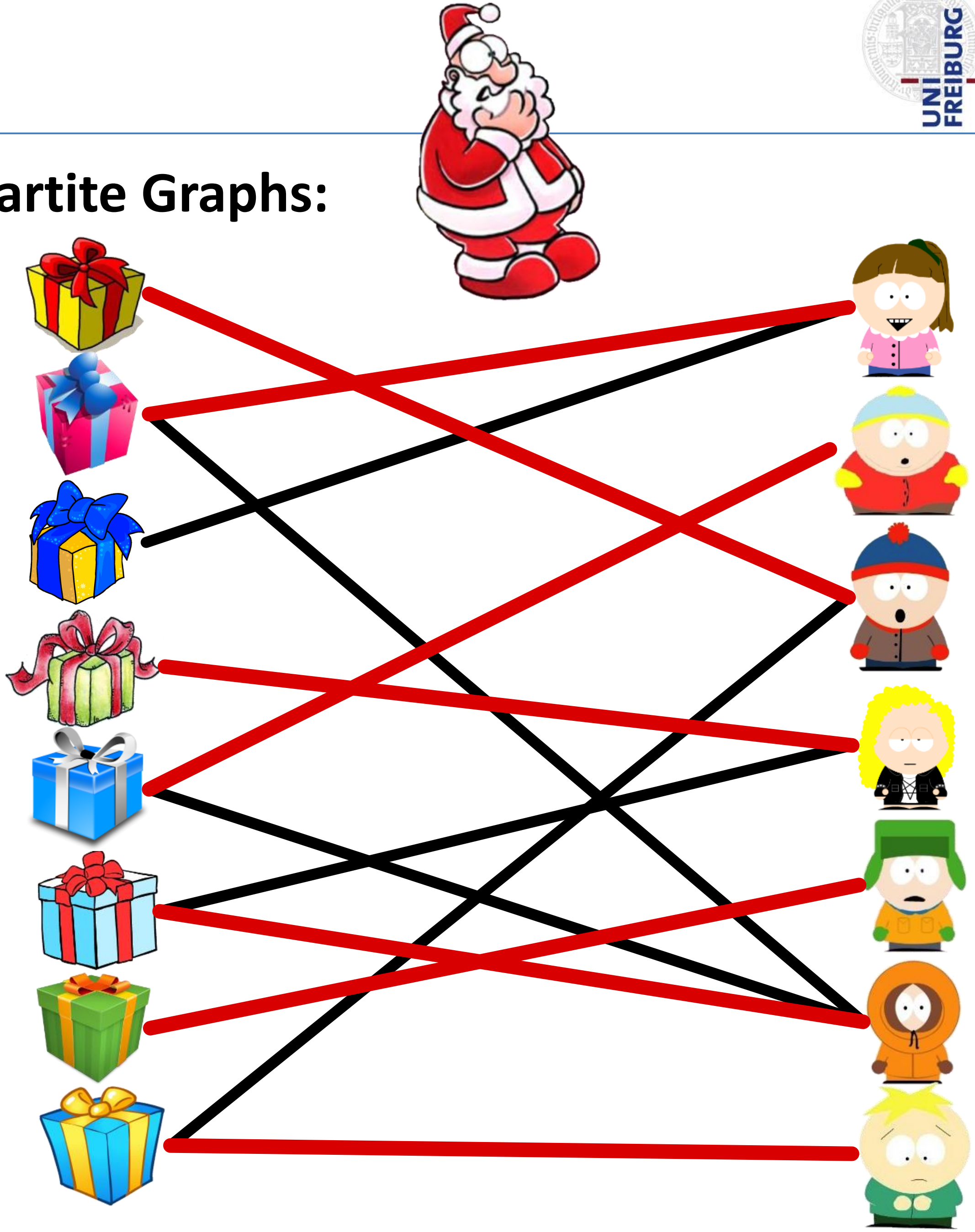
Santa's Problem

Maximum Matching in Bipartite Graphs:

Every child can get a gift iff there is a matching of size #children

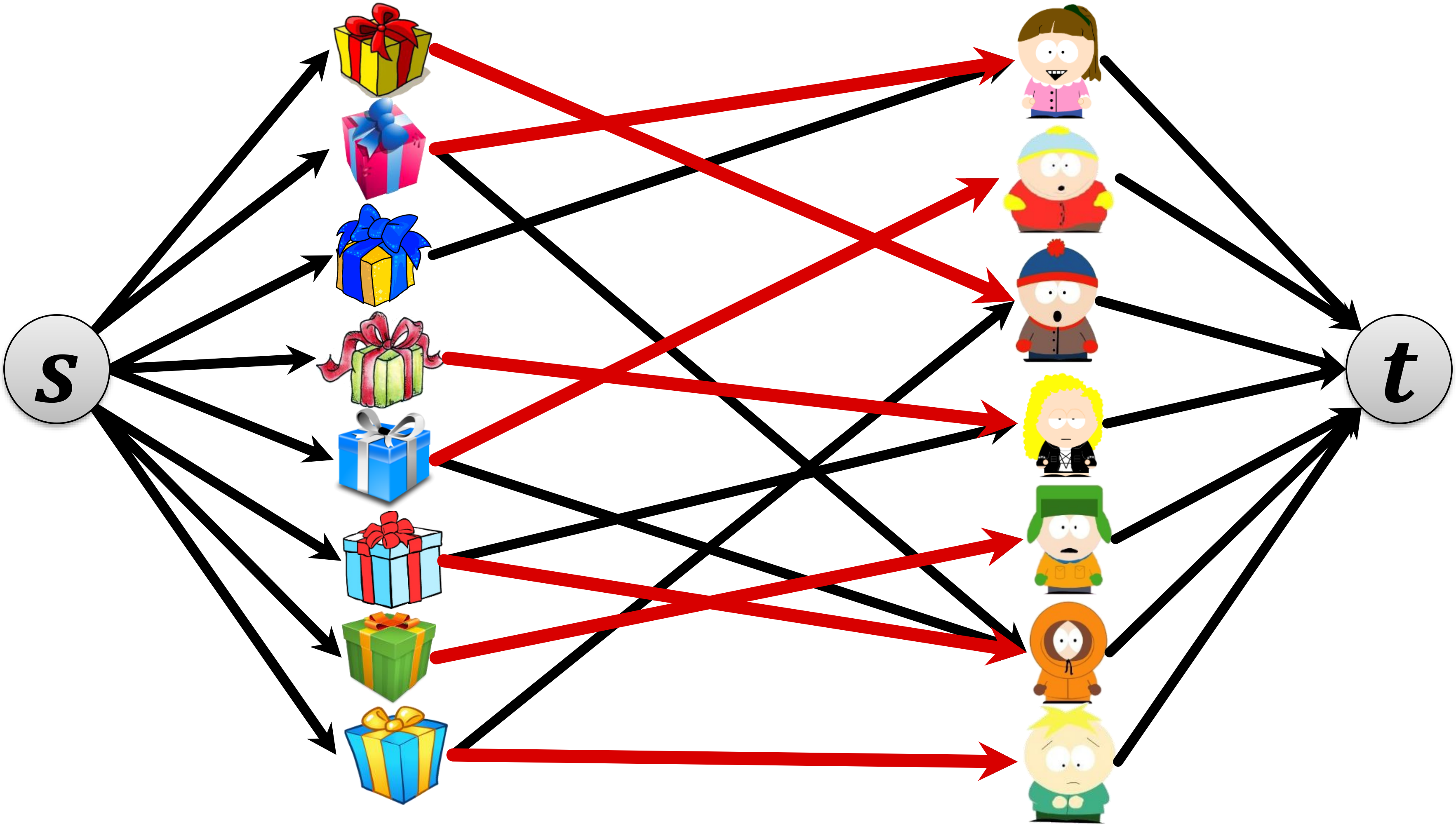
Clearly, every matching is at most as big

If #children = #gifts, there is a solution iff there is a perfect matching



Reducing to Maximum Flow

- Like edge-disjoint paths…



all capacities are 1

Reducing to Maximum Flow

Theorem: Every integer solution to the max flow problem on the constructed graph induces a maximum bipartite matching of G.

Proof:

1. An integer flow f of value $|f|$ induces a matching of size $|f|$
   - Left nodes (gifts) have incoming capacity 1
   - Right nodes (children) have outgoing capacity 1
   - Left and right nodes are incident to ≤ 1 edge e of G with $f(e) = 1$

2. A matching of size k implies a flow f of value $|f| = k$
   - For each edge $\{u, v\}$ of the matching:

$$f\big((s,u)\big) = f\big((u,v)\big) = f\big((v,t)\big) = 1$$

   - All other flow values are 0

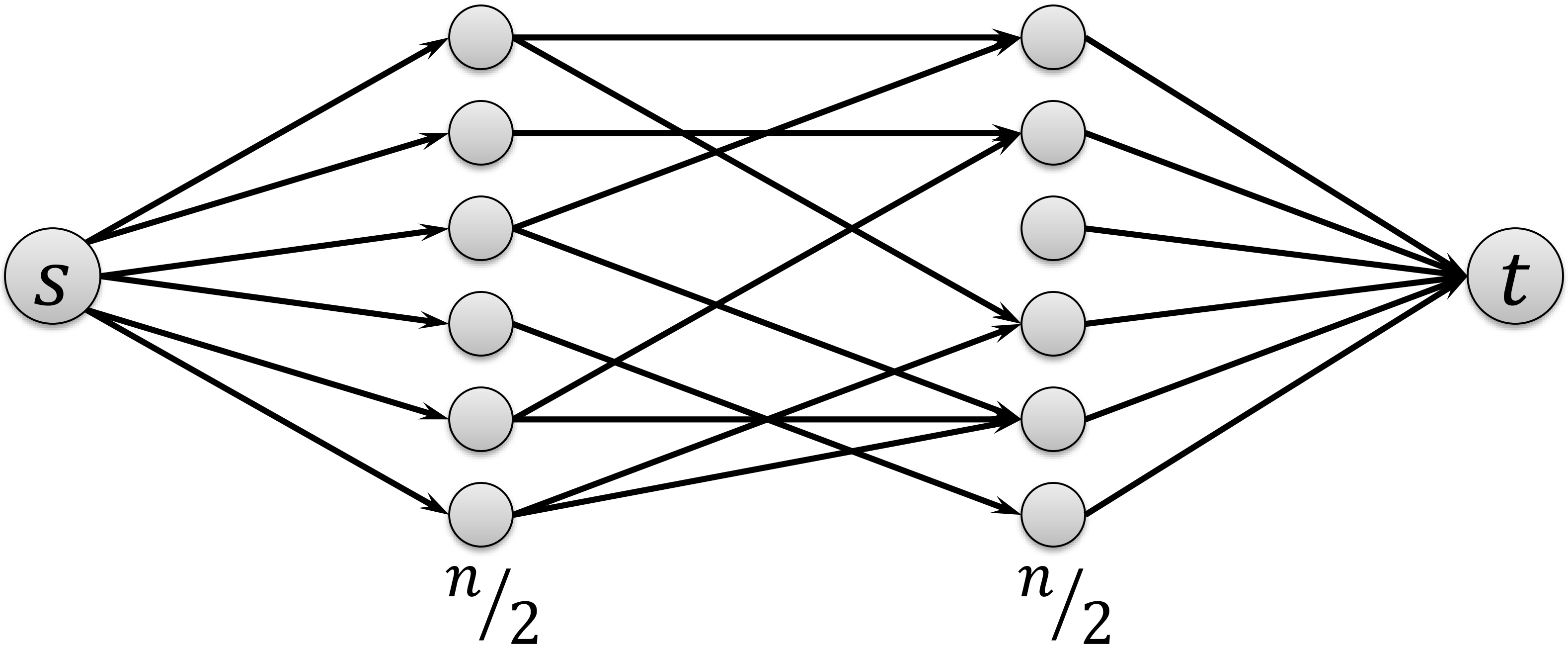
Running Time of Max. Bipartite Matching

Theorem: A maximum matching of a bipartite graph can be computed in time $O(m \cdot n)$.

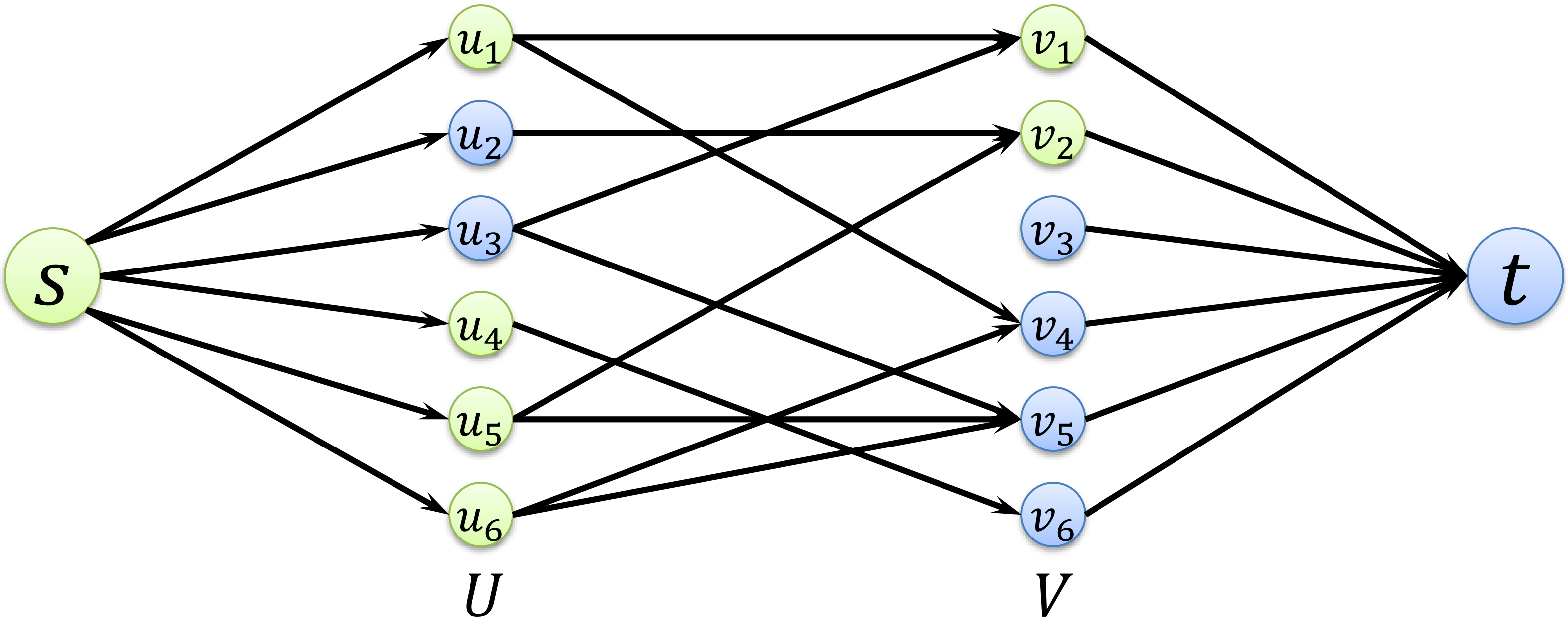
Perfect Matching?

- There can only be a perfect matching if both sides of the partition have size ${}^{n}/_{2}$.

- There is no perfect matching, iff there is an s-t cut of size $< {}^{n}/_{2}$ in the flow network.



s-t Cuts



Partition (A, B) of node set such that $s \in A$ and $t \in B$

- If $v_i \in A$: edge (v_i, t) is in cut (A, B)
- If $u_i \in B$: edge (s, u_i) is in cut (A, B)
- Otherwise (if $u_i \in A$, $v_i \in B$), all edges from u_i to some $v_j \in B$ are in cut (A, B)

Hall's Marriage Theorem

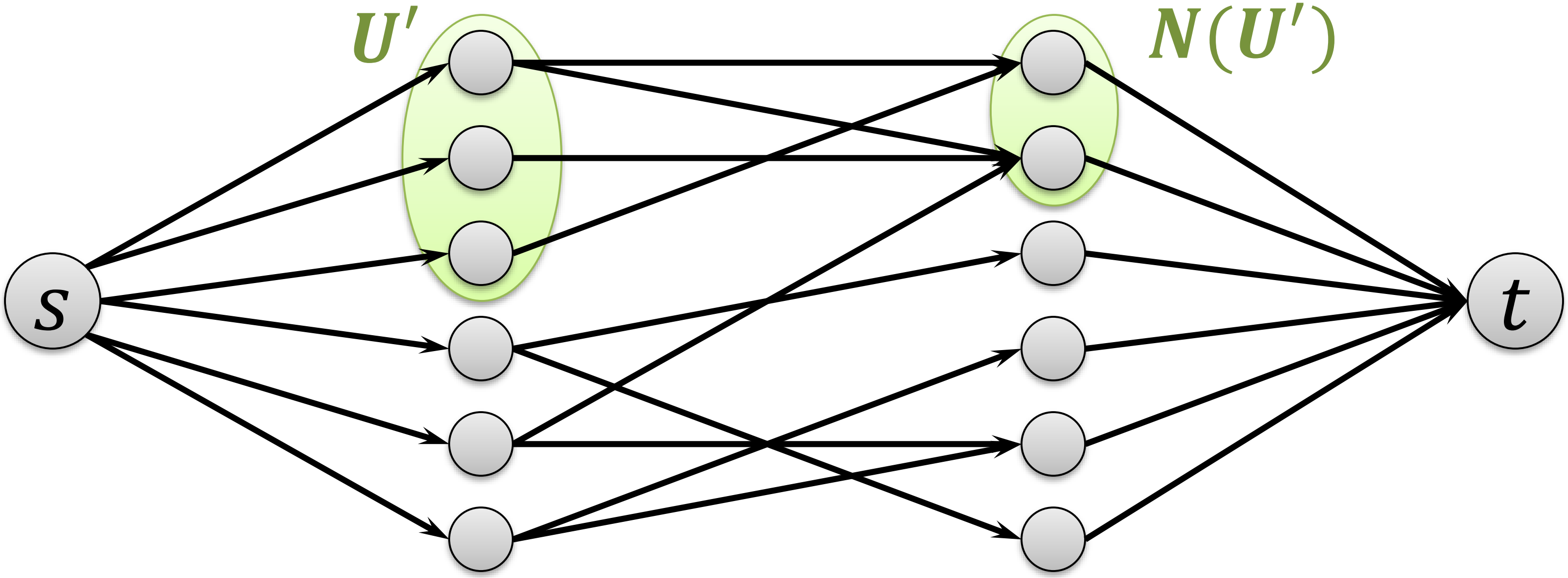
Theorem: A bipartite graph $G = (U \cup V, E)$ for which $|U| = |V|$ has a perfect matching if and only if

$$\forall \boldsymbol{U'} \subseteq \boldsymbol{U}: |\boldsymbol{N}(\boldsymbol{U'})| \geq |\boldsymbol{U'}|,$$

where $N(U') \subseteq V$ is the set of neighbors of nodes in U'.

Proof: No perfect matching $\Leftrightarrow$ some s-t cut has capacity $< n/2$

1. Assume there is U' for which $|N(U')| < |\mathrm{U}'|$:



Hall's Marriage Theorem

Theorem: A bipartite graph $G = (U \cup V, E)$ for which $|U| = |V|$ has a perfect matching if and only if

$$\forall U' \subseteq U: |N(U')| \geq |U'|,$$

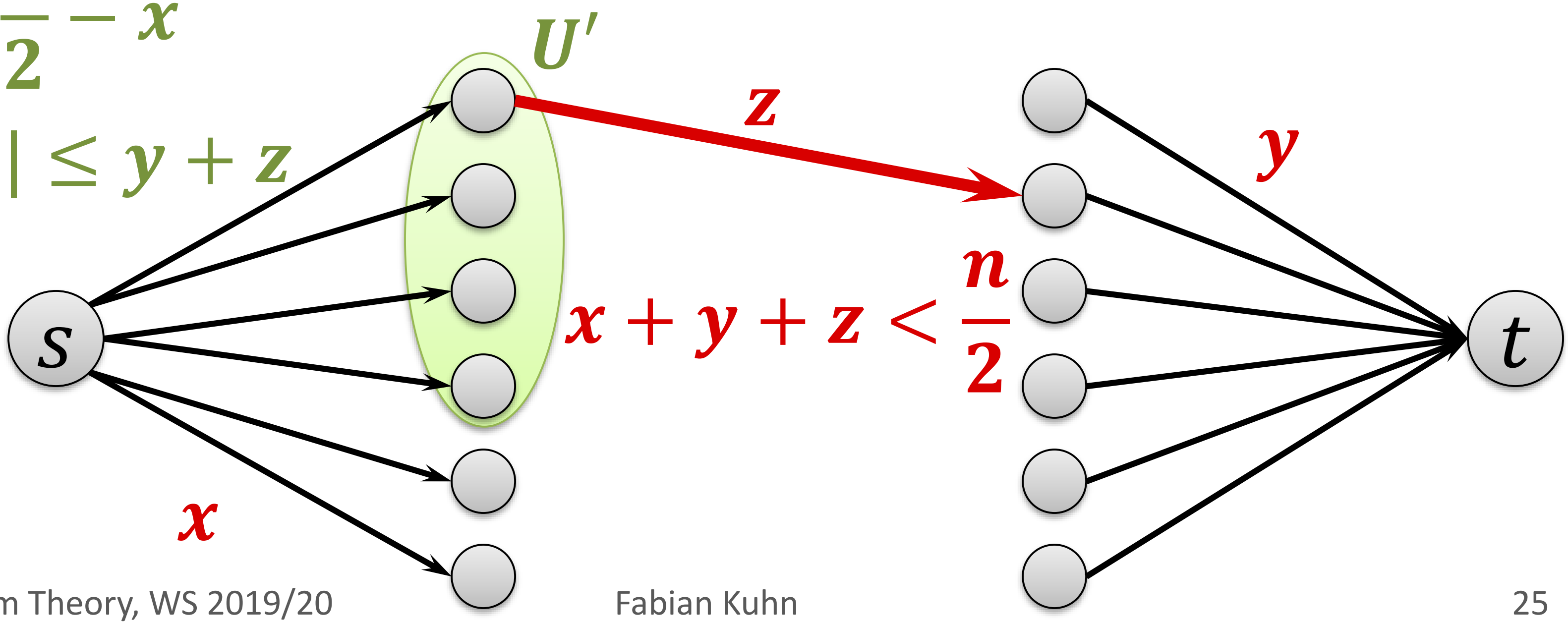
where $N(U') \subseteq V$ is the set of neighbors of nodes in U'.

Proof: No perfect matching $\Leftrightarrow$ some s-t cut has capacity $< n/2$

2. Assume that there is a cut (A, B) of capacity $< n/2$

$$|U'| = \frac{n}{2} - x$$

$$|N(U')| \leq y + z$$



Hall's Marriage Theorem

Theorem: A bipartite graph $G = (U \cup V, E)$ for which $|U| = |V|$ has a perfect matching if and only if

$$\forall U' \subseteq U : |N(U')| \geq |U'|,$$

where $N(U') \subseteq V$ is the set of neighbors of nodes in U'.

Proof: No perfect matching $\Longleftrightarrow$ some s-t cut has capacity $< n$

2. Assume that there is a cut (A, B) of capacity $< n$

$$|U'| = \frac{n}{2} - x$$

$$|N(U')| \leq y + z$$

$$x + y + z < \frac{n}{2}$$

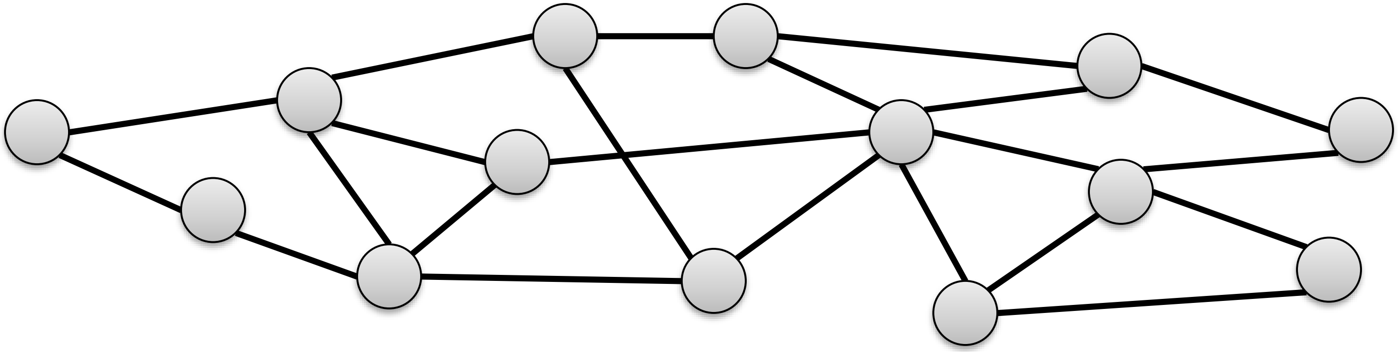
What About General Graphs

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

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

- **Vertex Cover:** set $S \subseteq V$ of nodes such that

$$\forall \{u,v\} \in E, \qquad \{u,v\} \cap S \neq \emptyset.$$



- A vertex cover covers all edges by incident nodes

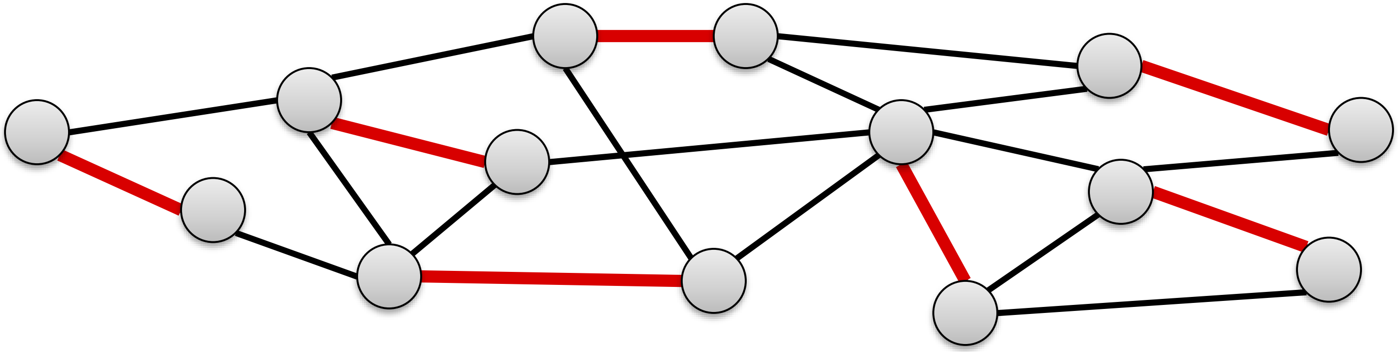
Vertex Cover vs Matching

Consider a matching M and a vertex cover S

Claim: $|M| \leq |S|$

Proof:

- At least one node of every edge $\{u, v\} \in M$ is in S
- Needs to be a different node for different edges from M



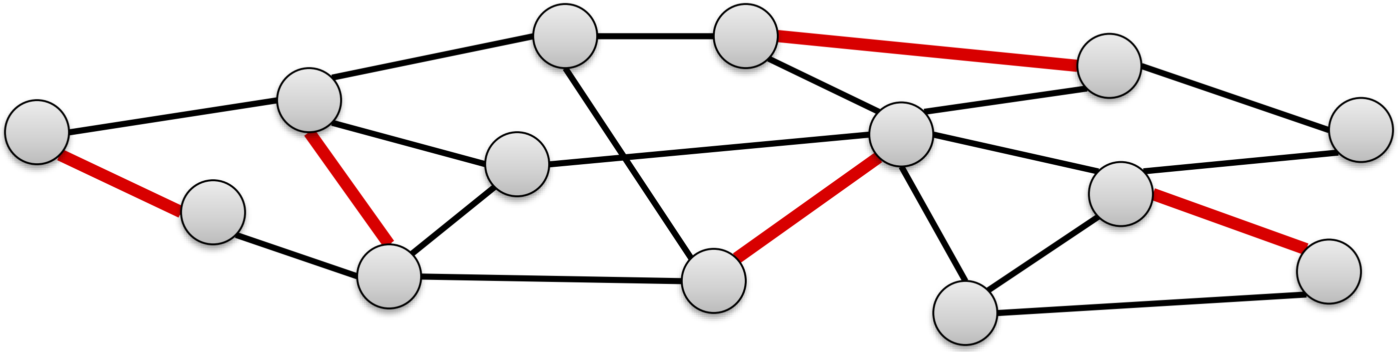
Vertex Cover vs Matching

Consider a matching M and a vertex cover S

Claim: If M is maximal and S is minimum, $|S| \leq 2|M|$

Proof:

- M is maximal: for every edge $\{u, v\} \in E$, either u or v (or both) are matched



- Every edge $e \in E$ is "covered" by at least one matching edge
- Thus, the set of the nodes of all matching edges gives a vertex cover S of size $|S| = 2|M|$.

Maximal Matching Approximation

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

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

Proof:

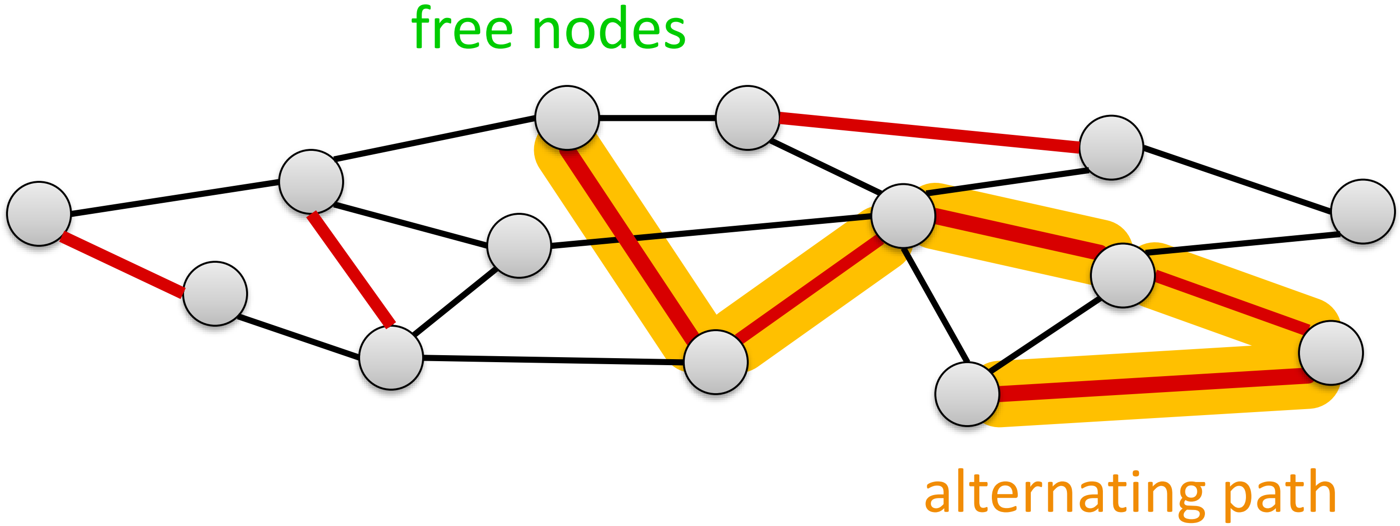
Theorem: The set of all matched nodes of a maximal matching M is a vertex cover of size at most twice the size of a min. vertex cover.

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

Augmenting Paths

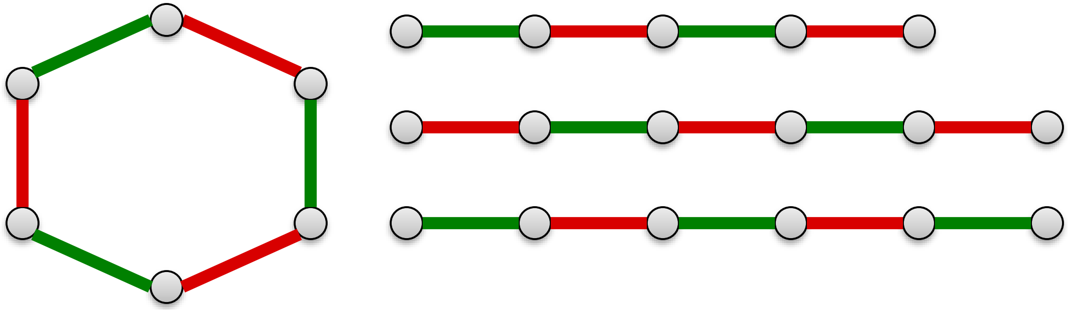
Theorem: A matching M of $G = (V, E)$ is maximum if and only if there is no augmenting path.

Proof:

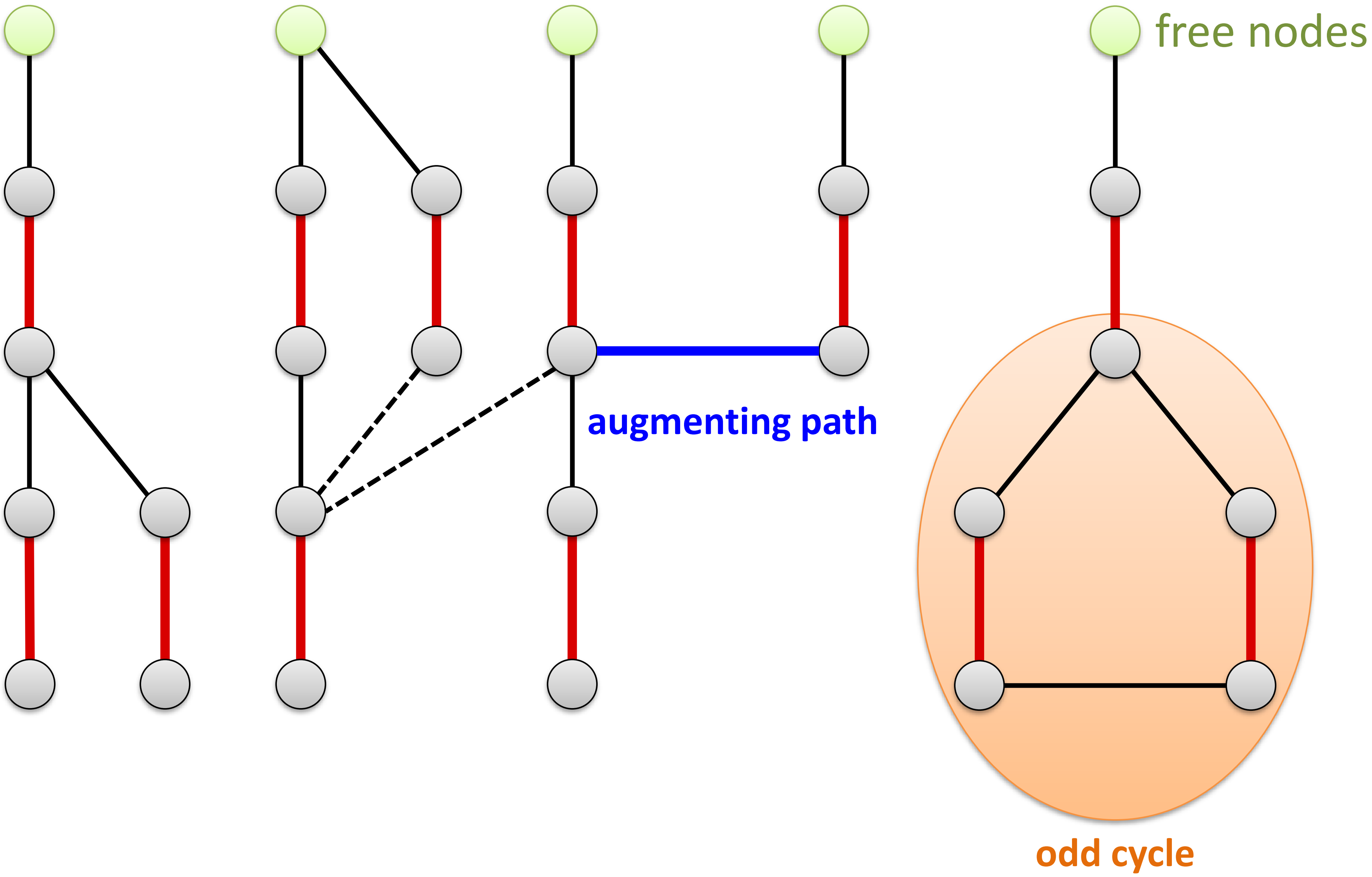
- Consider non-max. matching M and max. matching M^* and define

$$F \coloneqq M \setminus M^*, \qquad F^* \coloneqq M^* \setminus M$$

- 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

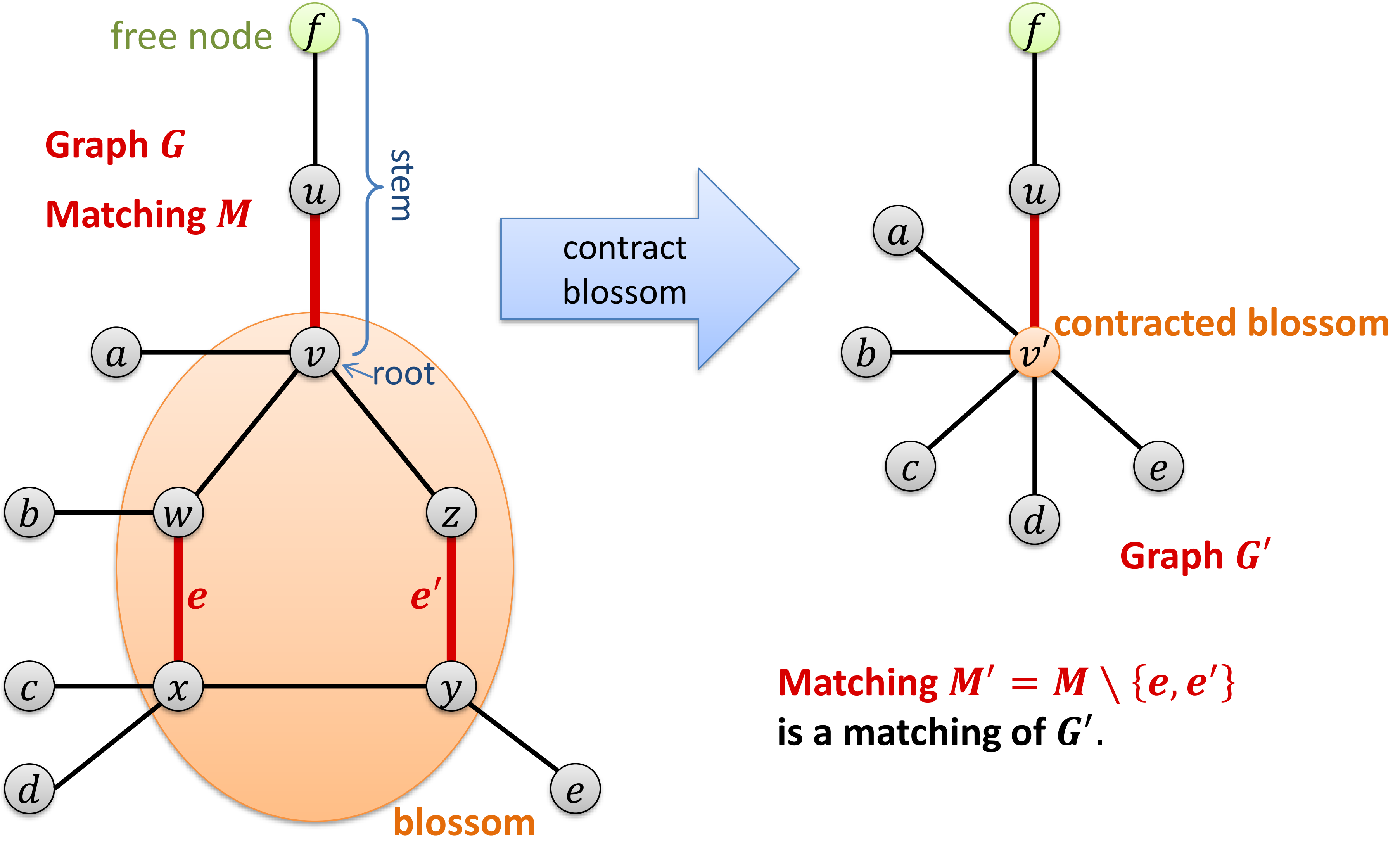


Finding Augmenting Paths



Blossoms

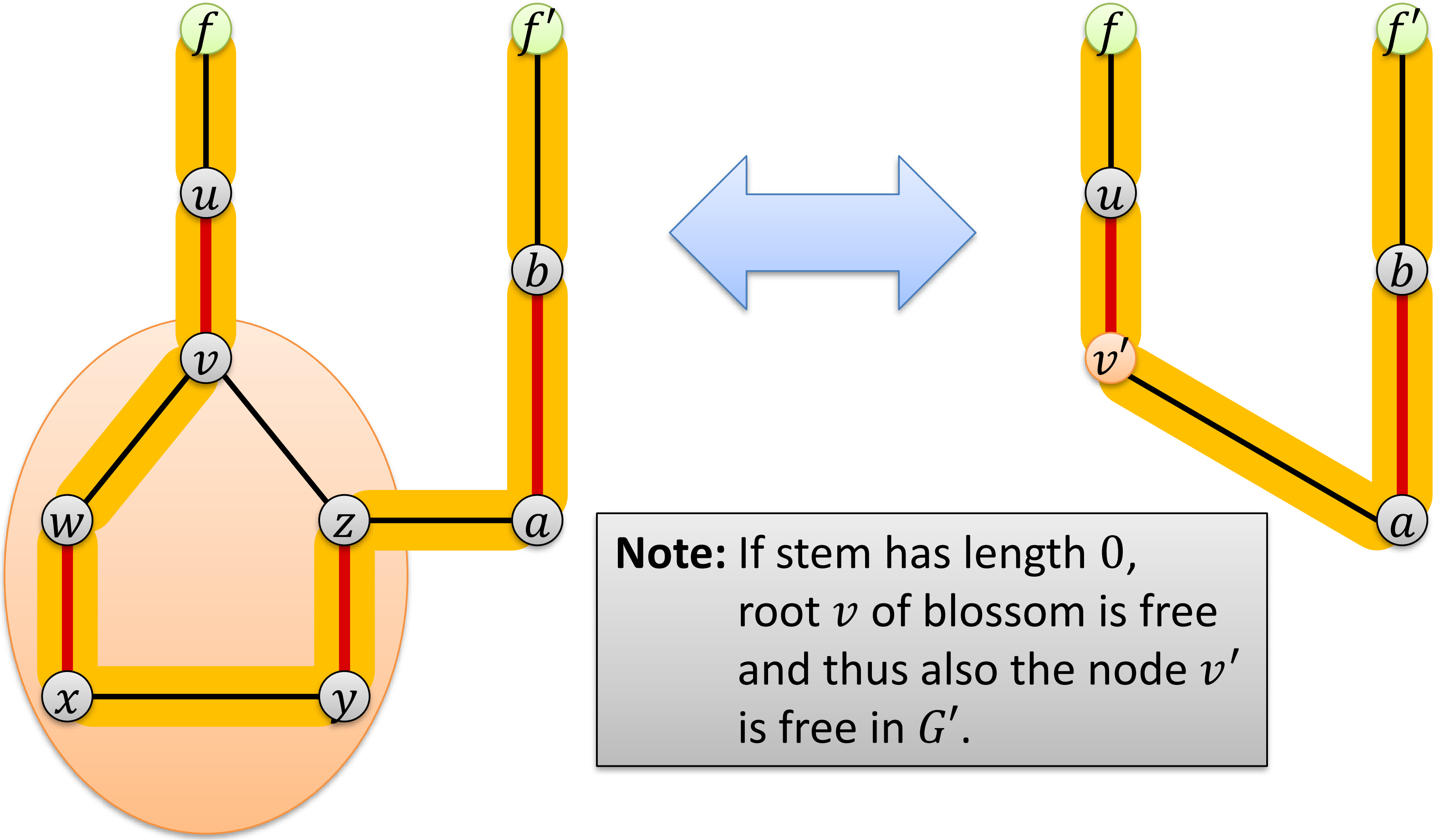
- If we find an odd cycle…



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'



Note: If stem has length 0, root v of blossom is free and thus also the node v' is free in G'.

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

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: Repeat on smaller graph

Finding an Augmenting Path: Improve matching

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