

# 16 Weighted Bipartite Matching

## Weighted Bipartite Matching/Assignment

- ▶ Input: undirected, bipartite graph  $G = L \cup R, E$ .
- ▶ an edge  $e = (\ell, r)$  has weight  $w_e \geq 0$
- ▶ find a matching of maximum weight, where the weight of a matching is the sum of the weights of its edges

## Simplifying Assumptions (wlog [why?]):

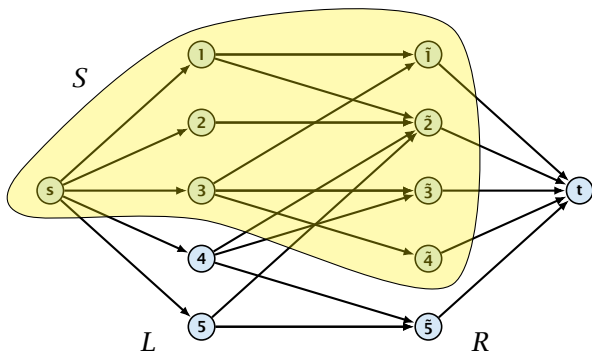
- ▶ assume that  $|L| = |R| = n$
- ▶ assume that there is an edge between every pair of nodes  $(\ell, r) \in V \times V$
- ▶ can assume goal is to construct maximum weight **perfect** matching

# Weighted Bipartite Matching

## Theorem 86 (Halls Theorem)

*A bipartite graph  $G = (L \cup R, E)$  has a perfect matching if and only if for all sets  $S \subseteq L$ ,  $|\Gamma(S)| \geq |S|$ , where  $\Gamma(S)$  denotes the set of nodes in  $R$  that have a neighbour in  $S$ .*

## 16 Weighted Bipartite Matching



# Halls Theorem

## Proof:

- ← Of course, the condition is necessary as otherwise not all nodes in  $S$  could be matched to different neighbours.

# Halls Theorem

## Proof:

- ⇐ Of course, the condition is necessary as otherwise not all nodes in  $S$  could be matched to different neighbours.
- ⇒ For the other direction we need to argue that the minimum cut in the graph  $G'$  is at least  $|L|$ .

# Halls Theorem

## Proof:

- ⇐ Of course, the condition is necessary as otherwise not all nodes in  $S$  could be matched to different neighbours.
- ⇒ For the other direction we need to argue that the minimum cut in the graph  $G'$  is at least  $|L|$ .
  - ▶ Let  $S$  denote a minimum cut and let  $L_S \stackrel{\text{def}}{=} L \cap S$  and  $R_S \stackrel{\text{def}}{=} R \cap S$  denote the portion of  $S$  inside  $L$  and  $R$ , respectively.

# Halls Theorem

## Proof:

- ⇐ Of course, the condition is necessary as otherwise not all nodes in  $S$  could be matched to different neighbours.
- ⇒ For the other direction we need to argue that the minimum cut in the graph  $G'$  is at least  $|L|$ .
  - ▶ Let  $S$  denote a minimum cut and let  $L_S \stackrel{\text{def}}{=} L \cap S$  and  $R_S \stackrel{\text{def}}{=} R \cap S$  denote the portion of  $S$  inside  $L$  and  $R$ , respectively.
  - ▶ Clearly, all neighbours of nodes in  $L_S$  have to be in  $S$ , as otherwise we would cut an edge of infinite capacity.

# Halls Theorem

## Proof:

- ⇐ Of course, the condition is necessary as otherwise not all nodes in  $S$  could be matched to different neighbours.
- ⇒ For the other direction we need to argue that the minimum cut in the graph  $G'$  is at least  $|L|$ .
  - ▶ Let  $S$  denote a minimum cut and let  $L_S \stackrel{\text{def}}{=} L \cap S$  and  $R_S \stackrel{\text{def}}{=} R \cap S$  denote the portion of  $S$  inside  $L$  and  $R$ , respectively.
  - ▶ Clearly, all neighbours of nodes in  $L_S$  have to be in  $S$ , as otherwise we would cut an edge of infinite capacity.
  - ▶ This gives  $R_S \geq |\Gamma(L_S)|$ .



# Halls Theorem

## Proof:

- ⇐ Of course, the condition is necessary as otherwise not all nodes in  $S$  could be matched to different neighbours.
- ⇒ For the other direction we need to argue that the minimum cut in the graph  $G'$  is at least  $|L|$ .
  - ▶ Let  $S$  denote a minimum cut and let  $L_S \stackrel{\text{def}}{=} L \cap S$  and  $R_S \stackrel{\text{def}}{=} R \cap S$  denote the portion of  $S$  inside  $L$  and  $R$ , respectively.
  - ▶ Clearly, all neighbours of nodes in  $L_S$  have to be in  $S$ , as otherwise we would cut an edge of infinite capacity.
  - ▶ This gives  $R_S \geq |\Gamma(L_S)|$ .
  - ▶ The size of the cut is  $|L| - |L_S| + |R_S|$ .

# Halls Theorem

## Proof:

- ⇐ Of course, the condition is necessary as otherwise not all nodes in  $S$  could be matched to different neighbours.
- ⇒ For the other direction we need to argue that the minimum cut in the graph  $G'$  is at least  $|L|$ .
  - ▶ Let  $S$  denote a minimum cut and let  $L_S \stackrel{\text{def}}{=} L \cap S$  and  $R_S \stackrel{\text{def}}{=} R \cap S$  denote the portion of  $S$  inside  $L$  and  $R$ , respectively.
  - ▶ Clearly, all neighbours of nodes in  $L_S$  have to be in  $S$ , as otherwise we would cut an edge of infinite capacity.
  - ▶ This gives  $R_S \geq |\Gamma(L_S)|$ .
  - ▶ The size of the cut is  $|L| - |L_S| + |R_S|$ .
  - ▶ Using the fact that  $|\Gamma(L_S)| \geq |L_S|$  gives that this is at least  $|L|$ .

# Algorithm Outline

## Idea:

We introduce a node weighting  $\vec{x}$ . Let for a node  $v \in V$ ,  $x_v \in \mathbb{R}$  denote the weight of node  $v$ .

# Algorithm Outline

## Idea:

We introduce a node weighting  $\vec{x}$ . Let for a node  $v \in V$ ,  $x_v \in \mathbb{R}$  denote the weight of node  $v$ .

- ▶ Suppose that the node weights dominate the edge-weights in the following sense:

$$x_u + x_v \geq w_e \text{ for every edge } e = (u, v).$$

# Algorithm Outline

## Idea:

We introduce a node weighting  $\vec{x}$ . Let for a node  $v \in V$ ,  $x_v \in \mathbb{R}$  denote the weight of node  $v$ .

- ▶ Suppose that the node weights dominate the edge-weights in the following sense:

$$x_u + x_v \geq w_e \text{ for every edge } e = (u, v).$$

- ▶ Let  $H(\vec{x})$  denote the subgraph of  $G$  that only contains edges that are **tight** w.r.t. the node weighting  $\vec{x}$ , i.e. edges  $e = (u, v)$  for which  $w_e = x_u + x_v$ .

# Algorithm Outline

## Idea:

We introduce a node weighting  $\vec{x}$ . Let for a node  $v \in V$ ,  $x_v \in \mathbb{R}$  denote the weight of node  $v$ .

- ▶ Suppose that the node weights dominate the edge-weights in the following sense:

$$x_u + x_v \geq w_e \text{ for every edge } e = (u, v).$$

- ▶ Let  $H(\vec{x})$  denote the subgraph of  $G$  that only contains edges that are **tight** w.r.t. the node weighting  $\vec{x}$ , i.e. edges  $e = (u, v)$  for which  $w_e = x_u + x_v$ .
- ▶ Try to compute a perfect matching in the subgraph  $H(\vec{x})$ . If you are successful you found an optimal matching.

# Algorithm Outline

## Reason:

- ▶ The weight of your matching  $M^*$  is

$$\sum_{(u,v) \in M^*} w(u,v) = \sum_{(u,v) \in M^*} (x_u + x_v) = \sum_v x_v .$$

- ▶ Any other perfect matching  $M$  (in  $G$ , not necessarily in  $H(\vec{x})$ ) has

$$\sum_{(u,v) \in M} w(u,v) \leq \sum_{(u,v) \in M} (x_u + x_v) = \sum_v x_v .$$

# Algorithm Outline

## What if you don't find a perfect matching?

Then, Hall's theorem guarantees you that there is a set  $S \subseteq L$ , with  $|\Gamma(S)| < |S|$ , where  $\Gamma$  denotes the neighbourhood w.r.t. the subgraph  $H(\vec{x})$ .



# Algorithm Outline

## What if you don't find a perfect matching?

Then, Halls theorem guarantees you that there is a set  $S \subseteq L$ , with  $|\Gamma(S)| < |S|$ , where  $\Gamma$  denotes the neighbourhood w.r.t. the subgraph  $H(\vec{x})$ .

**Idea:** reweight such that:

- ▶ the total weight assigned to nodes decreases
- ▶ the weight function still dominates the edge-weights

# Algorithm Outline

## What if you don't find a perfect matching?

Then, Hall's theorem guarantees you that there is a set  $S \subseteq L$ , with  $|\Gamma(S)| < |S|$ , where  $\Gamma$  denotes the neighbourhood w.r.t. the subgraph  $H(\vec{x})$ .

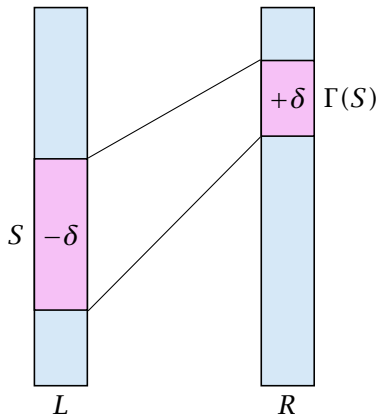
**Idea:** reweight such that:

- ▶ the total weight assigned to nodes decreases
- ▶ the weight function still dominates the edge-weights

If we can do this we have an algorithm that terminates with an optimal solution (we analyze the running time later).

## Changing Node Weights

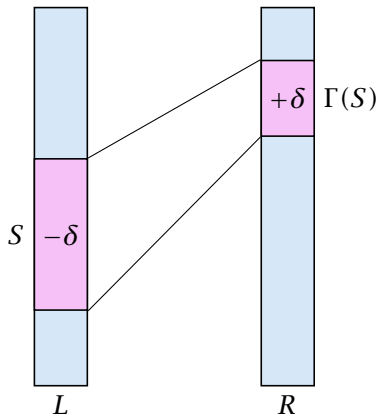
Increase node-weights in  $\Gamma(S)$  by  $+\delta$ , and decrease the node-weights in  $S$  by  $-\delta$ .



## Changing Node Weights

Increase node-weights in  $\Gamma(S)$  by  $+\delta$ , and decrease the node-weights in  $S$  by  $-\delta$ .

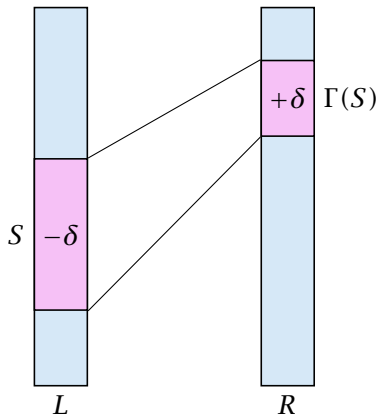
- ▶ Total node-weight decreases.



# Changing Node Weights

Increase node-weights in  $\Gamma(S)$  by  $+\delta$ , and decrease the node-weights in  $S$  by  $-\delta$ .

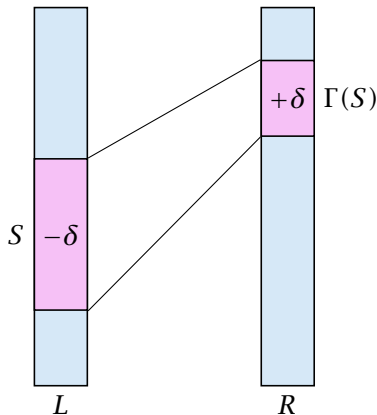
- ▶ Total node-weight decreases.
- ▶ Only edges from  $S$  to  $R - \Gamma(S)$  decrease in their weight.



## Changing Node Weights

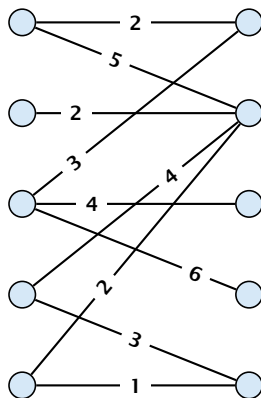
Increase node-weights in  $\Gamma(S)$  by  $+\delta$ , and decrease the node-weights in  $S$  by  $-\delta$ .

- ▶ Total node-weight decreases.
- ▶ Only edges from  $S$  to  $R - \Gamma(S)$  decrease in their weight.
- ▶ Since, none of these edges is tight (otw. the edge would be contained in  $H(\vec{x})$ , and hence would go between  $S$  and  $\Gamma(S)$ ) we can do this decrement for small enough  $\delta > 0$  until a new edge gets tight.



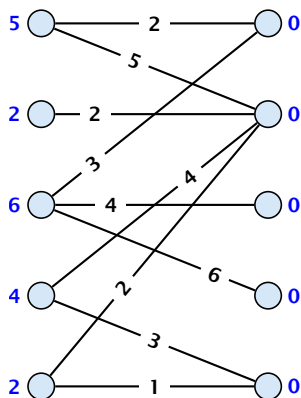
# Weighted Bipartite Matching

Edges not drawn have weight 0.



# Weighted Bipartite Matching

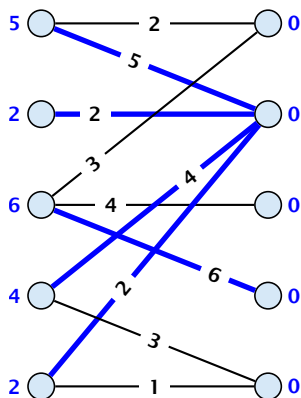
Edges not drawn have weight 0.





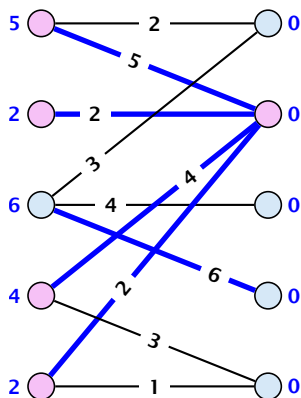
# Weighted Bipartite Matching

Edges not drawn have weight 0.



# Weighted Bipartite Matching

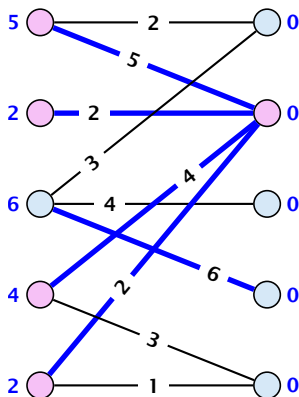
Edges not drawn have weight 0.



# Weighted Bipartite Matching

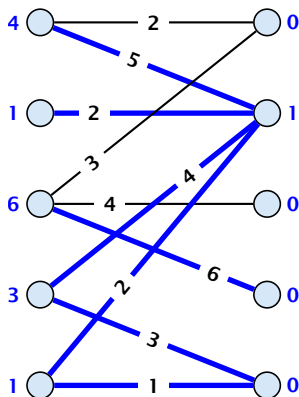
Edges not drawn have weight 0.

$$\delta = 1$$



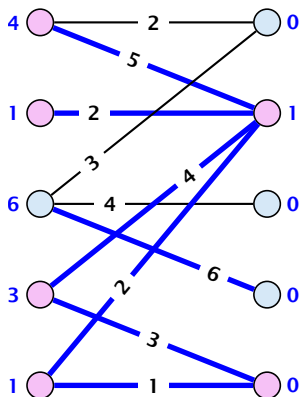
# Weighted Bipartite Matching

Edges not drawn have weight 0.



# Weighted Bipartite Matching

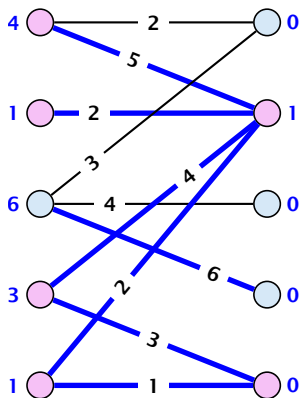
Edges not drawn have weight 0.



# Weighted Bipartite Matching

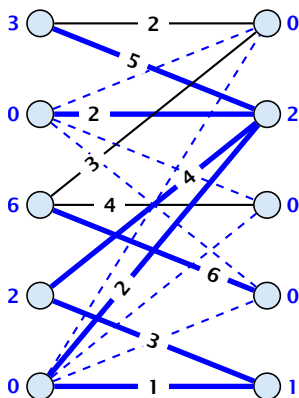
Edges not drawn have weight 0.

$$\delta = 1$$



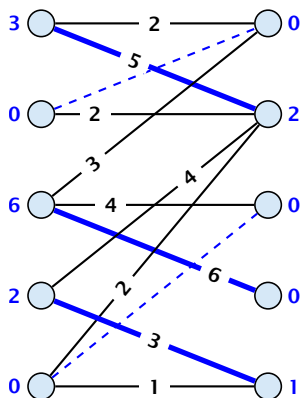
# Weighted Bipartite Matching

Edges not drawn have weight 0.



# Weighted Bipartite Matching

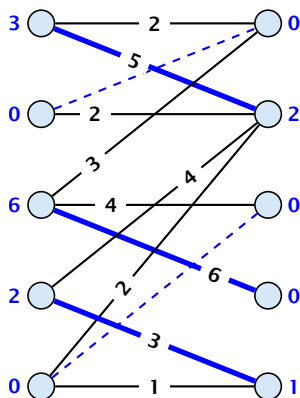
Edges not drawn have weight 0.





# Weighted Bipartite Matching

Edges not drawn have weight 0.



# Analysis

## How many iterations do we need?

- ▶ One reweighting step increases the number of edges out of  $S$  by at least one.

## How many iterations do we need?

- ▶ One reweighting step increases the number of edges out of  $S$  by at least one.
- ▶ Assume that we have a maximum matching that saturates the set  $\Gamma(S)$ , in the sense that every node in  $\Gamma(S)$  is matched to a node in  $S$  (we will show that we can always find  $S$  and a matching such that this holds).

## How many iterations do we need?

- ▶ One reweighting step increases the number of edges out of  $S$  by at least one.
- ▶ Assume that we have a maximum matching that saturates the set  $\Gamma(S)$ , in the sense that every node in  $\Gamma(S)$  is matched to a node in  $S$  (we will show that we can always find  $S$  and a matching such that this holds).
- ▶ This matching is still contained in the new graph, because all its edges either go between  $\Gamma(S)$  and  $S$  or between  $L - S$  and  $R - \Gamma(S)$ .

## How many iterations do we need?

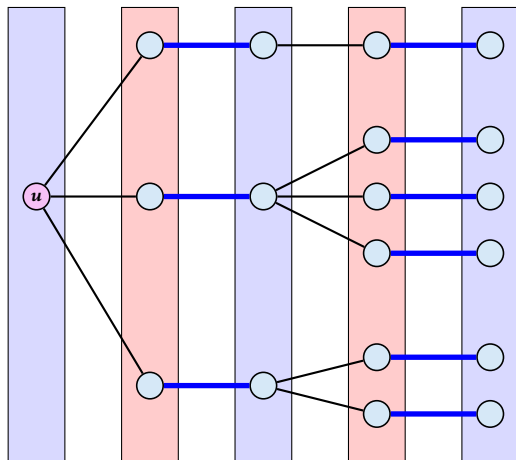
- ▶ One reweighting step increases the number of edges out of  $S$  by at least one.
- ▶ Assume that we have a maximum matching that saturates the set  $\Gamma(S)$ , in the sense that every node in  $\Gamma(S)$  is matched to a node in  $S$  (we will show that we can always find  $S$  and a matching such that this holds).
- ▶ This matching is still contained in the new graph, because all its edges either go between  $\Gamma(S)$  and  $S$  or between  $L - S$  and  $R - \Gamma(S)$ .
- ▶ Hence, reweighting does not decrease the size of a maximum matching in the tight sub-graph.

# Analysis

- ▶ We will show that after at most  $n$  reweighting steps the size of the maximum matching can be increased by finding an augmenting path.
- ▶ This gives a polynomial running time.

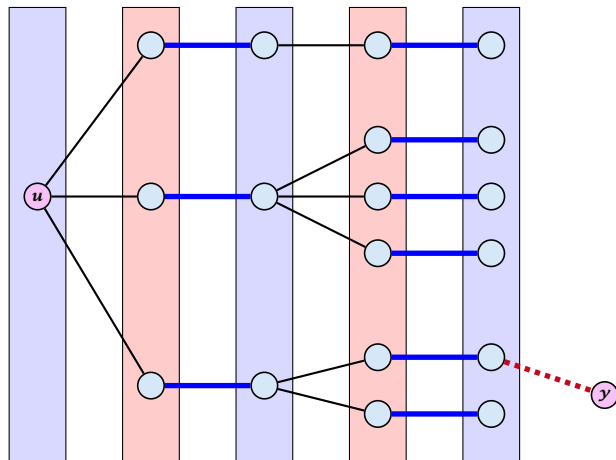
# How to find an augmenting path?

Construct an alternating tree.



# How to find an augmenting path?

Construct an alternating tree.





# Analysis

## How do we find $S$ ?

- ▶ Start on the left and compute an alternating tree, starting at any free node  $u$ .

# Analysis

## How do we find $S$ ?

- ▶ Start on the left and compute an alternating tree, starting at any free node  $u$ .
- ▶ If this construction stops, there is no perfect matching in the tight subgraph (because for a perfect matching we need to find an augmenting path starting at  $u$ ).

# Analysis

## How do we find $S$ ?

- ▶ Start on the left and compute an alternating tree, starting at any free node  $u$ .
- ▶ If this construction stops, there is no perfect matching in the tight subgraph (because for a perfect matching we need to find an augmenting path starting at  $u$ ).
- ▶ The set of even vertices is on the left and the set of odd vertices is on the right **and** contains all neighbours of even nodes.

# Analysis

## How do we find $S$ ?

- ▶ Start on the left and compute an alternating tree, starting at any free node  $u$ .
- ▶ If this construction stops, there is no perfect matching in the tight subgraph (because for a perfect matching we need to find an augmenting path starting at  $u$ ).
- ▶ The set of even vertices is on the left and the set of odd vertices is on the right **and** contains all neighbours of even nodes.
- ▶ All odd vertices are matched to even vertices. Furthermore, the even vertices additionally contain the free vertex  $u$ . Hence,  $|V_{\text{odd}}| = |\Gamma(V_{\text{even}})| < |V_{\text{even}}|$ , and all odd vertices are saturated in the current matching.

# Analysis

- ▶ The current matching does not have any edges from  $V_{\text{odd}}$  to  $L \setminus V_{\text{even}}$  (edges that may possibly be deleted by changing weights).

# Analysis

- ▶ The current matching does not have any edges from  $V_{\text{odd}}$  to  $L \setminus V_{\text{even}}$  (edges that may possibly be deleted by changing weights).
- ▶ After changing weights, there is at least one more edge connecting  $V_{\text{even}}$  to a node outside of  $V_{\text{odd}}$ . After at most  $n$  reweights we can do an augmentation.

# Analysis

- ▶ The current matching does not have any edges from  $V_{\text{odd}}$  to  $L \setminus V_{\text{even}}$  (edges that may possibly be deleted by changing weights).
- ▶ After changing weights, there is at least one more edge connecting  $V_{\text{even}}$  to a node outside of  $V_{\text{odd}}$ . After at most  $n$  reweightings we can do an augmentation.
- ▶ A reweighting can be trivially performed in time  $\mathcal{O}(n^2)$  (keeping track of the tight edges).

# Analysis

- ▶ The current matching does not have any edges from  $V_{\text{odd}}$  to  $L \setminus V_{\text{even}}$  (edges that may possibly be deleted by changing weights).
- ▶ After changing weights, there is at least one more edge connecting  $V_{\text{even}}$  to a node outside of  $V_{\text{odd}}$ . After at most  $n$  reweights we can do an augmentation.
- ▶ A reweighting can be trivially performed in time  $\mathcal{O}(n^2)$  (keeping track of the tight edges).
- ▶ An augmentation takes at most  $\mathcal{O}(n)$  time.



# Analysis

- ▶ The current matching does not have any edges from  $V_{\text{odd}}$  to  $L \setminus V_{\text{even}}$  (edges that may possibly be deleted by changing weights).
- ▶ After changing weights, there is at least one more edge connecting  $V_{\text{even}}$  to a node outside of  $V_{\text{odd}}$ . After at most  $n$  reweightings we can do an augmentation.
- ▶ A reweighting can be trivially performed in time  $\mathcal{O}(n^2)$  (keeping track of the tight edges).
- ▶ An augmentation takes at most  $\mathcal{O}(n)$  time.
- ▶ In total we obtain a running time of  $\mathcal{O}(n^4)$ .

# Analysis

- ▶ The current matching does not have any edges from  $V_{\text{odd}}$  to  $L \setminus V_{\text{even}}$  (edges that may possibly be deleted by changing weights).
- ▶ After changing weights, there is at least one more edge connecting  $V_{\text{even}}$  to a node outside of  $V_{\text{odd}}$ . After at most  $n$  reweights we can do an augmentation.
- ▶ A reweighting can be trivially performed in time  $\mathcal{O}(n^2)$  (keeping track of the tight edges).
- ▶ An augmentation takes at most  $\mathcal{O}(n)$  time.
- ▶ In total we obtain a running time of  $\mathcal{O}(n^4)$ .
- ▶ A more careful implementation of the algorithm obtains a running time of  $\mathcal{O}(n^3)$ .