IN2107 - Image Segmentation and Shape Analysis (Seminar)

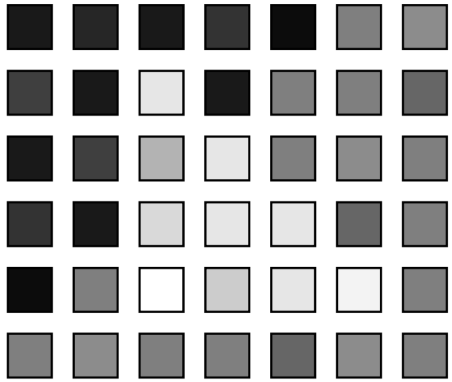
Prof. Dr. Daniel Cremers
Dr. Frank R. Schmidt
Dr. Csaba Domokos
Matthias Vestner
Zorah Lähner

Winter Semester 2016/2017

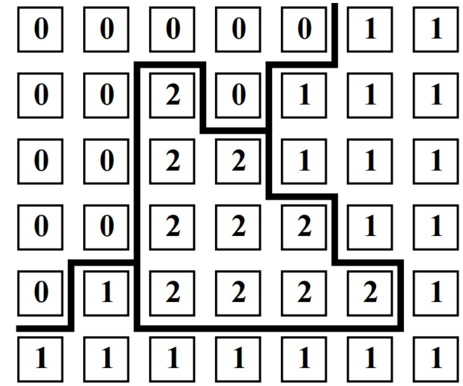
An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision

Energy minimization

The labeling problem

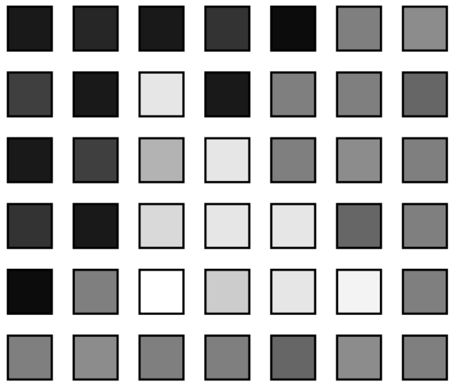


An input image

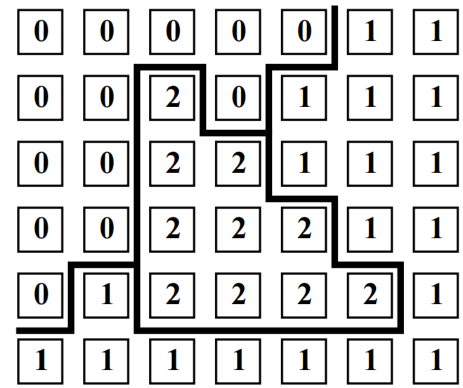


A labeling

The labeling problem



An input image

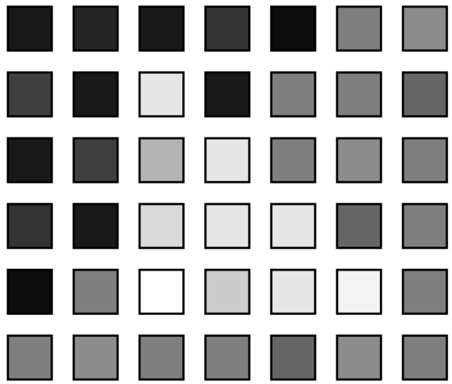


A labeling

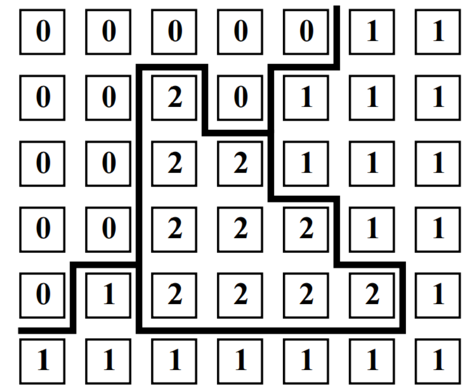
We will use the following notations

- $\mathcal{V}$ denotes a *set of output variables* (e.g., for pixels) and the corresponding random variables are denoted by Y_i for all $i \in \mathcal{V}$

The labeling problem



An input image

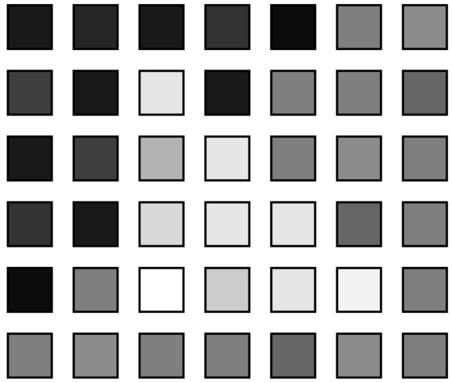


A labeling

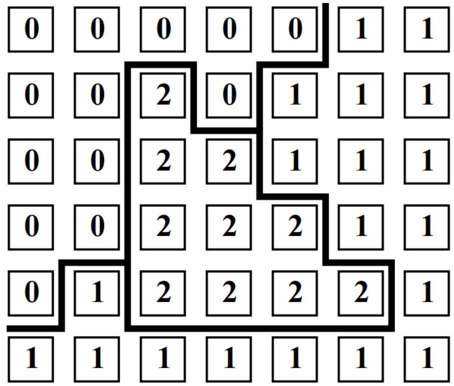
We will use the following notations

- $\mathcal{V}$ denotes a *set of output variables* (e.g., for pixels) and the corresponding random variables are denoted by Y_i for all $i \in \mathcal{V}$
- The *output domain* $\mathcal{Y}$ is given by the product of individual variable domains $\mathcal{Y}_i$ (e.g., a single label set $\mathcal{L}$), that is $\mathcal{Y} = \times_{i \in \mathcal{V}} \mathcal{Y}_i = \mathcal{L}^{\mathcal{V}}$

The labeling problem



An input image

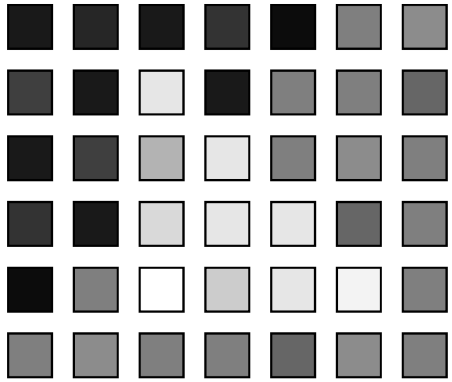


A labeling

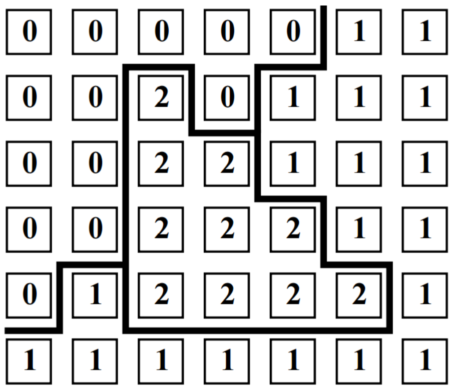
We will use the following notations

- $\mathcal{V}$ denotes a *set of output variables* (e.g., for pixels) and the corresponding random variables are denoted by Y_i for all $i \in \mathcal{V}$
- The *output domain* $\mathcal{Y}$ is given by the product of individual variable domains $\mathcal{Y}_i$ (e.g., a single label set $\mathcal{L}$), that is $\mathcal{Y} = \times_{i \in \mathcal{V}} \mathcal{Y}_i = \mathcal{L}^{\mathcal{V}}$
- The *realization* $\mathbf{Y} = \mathbf{y}$ means that $Y_i = y_i$ for all $i \in \mathcal{V}$

The labeling problem



An input image



A labeling

We will use the following notations

- $\mathcal{V}$ denotes a *set of output variables* (e.g., for pixels) and the corresponding random variables are denoted by Y_i for all $i \in \mathcal{V}$
- The *output domain* $\mathcal{Y}$ is given by the product of individual variable domains $\mathcal{Y}_i$ (e.g., a single label set $\mathcal{L}$), that is $\mathcal{Y} = \times_{i \in \mathcal{V}} \mathcal{Y}_i = \mathcal{L}^{\mathcal{V}}$
- The *realization* $\mathbf{Y} = \mathbf{y}$ means that $Y_i = y_i$ for all $i \in \mathcal{V}$

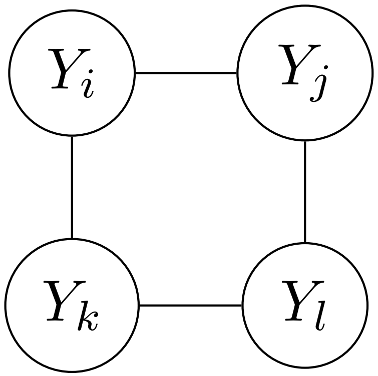
We aim to model the joint probability distribution $p(\mathbf{y})$, and find the *best* labeling $\mathbf{y}^*$

Markov random field

Consider an undirected graph $G = (\mathcal{V}, \mathcal{E})$ with the following assumption:

Two nodes (i.e. random variables) are *conditionally independent* whenever they are not connected, that is for any node i in the graph

$$p(Y_i \mid Y_{\mathcal{V}\setminus\{i\}}) = p(Y_i \mid Y_{N(i)}) \ ,$$



where $N(i)$ is denotes the neighbors of node i in the graph

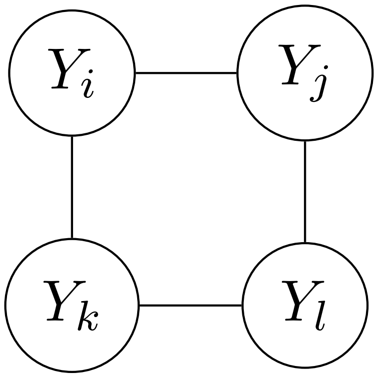
Markov random field

Consider an undirected graph $G = (\mathcal{V}, \mathcal{E})$ with the following assumption:

Two nodes (i.e. random variables) are *conditionally independent* whenever they are not connected, that is for any node i in the graph

$$p(Y_i \mid Y_{\mathcal{V}\setminus\{i\}}) = p(Y_i \mid Y_{N(i)}) \ ,$$

where $N(i)$ is denotes the neighbors of node i in the graph



A *probability distribution* $p(\mathbf{y})$ is called *Gibbs distribution* if it can be factorized into potential functions $\psi_c(\mathbf{y}_c) > 0$ defined on cliques:

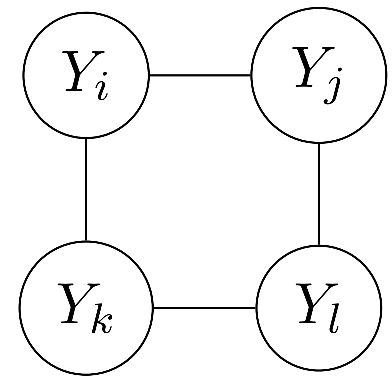
$$p(\mathbf{y}) = \frac{1}{Z} \prod_{c \in \mathcal{C}_G} \psi_c(\mathbf{y}_c) \ , \text{ where } Z = \sum_{\mathbf{y} \in \mathcal{Y}} \prod_{c \in \mathcal{C}_G} \psi_c(\mathbf{y}_c) \ ,$$

and $\mathcal{C}_G$ denotes the set of all (maximal) cliques in G

Markov random field

Consider an undirected graph $G = (\mathcal{V}, \mathcal{E})$ with the following assumption:

Two nodes (i.e. random variables) are *conditionally independent* whenever they are not connected, that is for any node i in the graph



$$p(Y_i \mid Y_{\mathcal{V}\setminus\{i\}}) = p(Y_i \mid Y_{N(i)}) \ ,$$

where $N(i)$ is denotes the neighbors of node i in the graph

A *probability distribution* $p(\mathbf{y})$ is called *Gibbs distribution* if it can be factorized into potential functions $\psi_c(\mathbf{y}_c) > 0$ defined on cliques:

$$p(\mathbf{y}) = \frac{1}{Z} \prod_{c \in \mathcal{C}_G} \psi_c(\mathbf{y}_c) \ , \ \text{where } Z = \sum_{\mathbf{y} \in \mathcal{Y}} \prod_{c \in \mathcal{C}_G} \psi_c(\mathbf{y}_c) \ ,$$

and $\mathcal{C}_G$ denotes the set of all (maximal) cliques in G

The *Hammersley-Clifford theorem* tells us that the above two definitions are equivalent.

Potentials and energy functions

Assuming $\psi_c : \mathcal{Y}_c \rightarrow \mathbb{R}^+$, where $\mathcal{Y}_c = \times_{i \in N(c)} \mathcal{Y}_i$ is the product domain of the clique c, instead of *potentials*, we can also work with *energies*

Potentials and energy functions

Assuming $\psi_c : \mathcal{Y}_c \to \mathbb{R}^+$, where $\mathcal{Y}_c = \times_{i \in N(c)} \mathcal{Y}_i$ is the product domain of the clique c, instead of *potentials*, we can also work with *energies*

We define an *energy function* $E_c : \mathcal{Y}_c \to \mathbb{R}$ for each clique $c \in \mathcal{C}_G$:

$$E_c(\mathbf{y}_c) = -\log(\psi_c(\mathbf{y}_c)) \quad \Leftrightarrow \quad \psi_c(\mathbf{y}_c) = \exp(-E_c(\mathbf{y}_c)) \; .$$

Potentials and energy functions

Assuming $\psi_c : \mathcal{Y}_c \to \mathbb{R}^+$, where $\mathcal{Y}_c = \times_{i \in N(c)} \mathcal{Y}_i$ is the product domain of the clique c, instead of *potentials*, we can also work with *energies*

We define an *energy function* $E_c : \mathcal{Y}_c \to \mathbb{R}$ for each clique $c \in \mathcal{C}_G$:

$$E_c(\mathbf{y}_c) = -\log(\psi_c(\mathbf{y}_c)) \quad \Leftrightarrow \quad \psi_c(\mathbf{y}_c) = \exp(-E_c(\mathbf{y}_c)) \ .$$

$$p(\mathbf{y}) = \frac{1}{Z} \prod_{c \in \mathcal{C}_G} \psi_c(\mathbf{y}_c)$$

Potentials and energy functions

Assuming $\psi_c : \mathcal{Y}_c \to \mathbb{R}^+$, where $\mathcal{Y}_c = \times_{i \in N(c)} \mathcal{Y}_i$ is the product domain of the clique c, instead of *potentials*, we can also work with *energies*

We define an *energy function* $E_c : \mathcal{Y}_c \to \mathbb{R}$ for each clique $c \in \mathcal{C}_G$:

$$E_c(\mathbf{y}_c) = -\log(\psi_c(\mathbf{y}_c)) \quad \Leftrightarrow \quad \psi_c(\mathbf{y}_c) = \exp(-E_c(\mathbf{y}_c)) \; .$$

$$p(\mathbf{y}) = \frac{1}{Z} \prod_{c \in \mathcal{C}_G} \psi_c(\mathbf{y}_c)$$

Potentials and energy functions

Assuming $\psi_c : \mathcal{Y}_c \rightarrow \mathbb{R}^+$, where $\mathcal{Y}_c = \times_{i \in N(c)} \mathcal{Y}_i$ is the product domain of the clique c, instead of *potentials*, we can also work with *energies*

We define an *energy function* $E_c : \mathcal{Y}_c \rightarrow \mathbb{R}$ for each clique $c \in \mathcal{C}_G$:

$$E_c(\mathbf{y}_c) = -\log(\psi_c(\mathbf{y}_c)) \quad \Leftrightarrow \quad \psi_c(\mathbf{y}_c) = \exp(-E_c(\mathbf{y}_c)) \; .$$

$$p(\mathbf{y}) = \frac{1}{Z} \prod_{c \in \mathcal{C}_G} \psi_c(\mathbf{y}_c) = \frac{1}{Z} \exp(-\sum_{c \in \mathcal{C}_G} E_c(\mathbf{y}_c))$$

Potentials and energy functions

Assuming $\psi_c : \mathcal{Y}_c \to \mathbb{R}^+$, where $\mathcal{Y}_c = \times_{i \in N(c)} \mathcal{Y}_i$ is the product domain of the clique c, instead of *potentials*, we can also work with *energies*

We define an *energy function* $E_c : \mathcal{Y}_c \to \mathbb{R}$ for each clique $c \in \mathcal{C}_G$:

$$E_c(\mathbf{y}_c) = -\log(\psi_c(\mathbf{y}_c)) \quad \Leftrightarrow \quad \psi_c(\mathbf{y}_c) = \exp(-E_c(\mathbf{y}_c)) \ .$$

$$\begin{aligned} p(\mathbf{y}) &= \frac{1}{Z} \prod_{c \in \mathcal{C}_G} \psi_c(\mathbf{y}_c) = \frac{1}{Z} \exp(-\sum_{c \in \mathcal{C}_G} E_c(\mathbf{y}_c)) \\ &= \frac{1}{Z} \exp(-E(\mathbf{y})) \end{aligned}$$

Potentials and energy functions

Assuming $\psi_c : \mathcal{Y}_c \to \mathbb{R}^+$, where $\mathcal{Y}_c = \times_{i \in N(c)} \mathcal{Y}_i$ is the product domain of the clique c, instead of *potentials*, we can also work with *energies*

We define an *energy function* $E_c : \mathcal{Y}_c \to \mathbb{R}$ for each clique $c \in \mathcal{C}_G$:

$$E_c(\mathbf{y}_c) = -\log(\psi_c(\mathbf{y}_c)) \quad \Leftrightarrow \quad \psi_c(\mathbf{y}_c) = \exp(-E_c(\mathbf{y}_c)) \; .$$

$$\begin{aligned} p(\mathbf{y}) =& \frac{1}{Z} \prod_{c \in \mathcal{C}_G} \psi_c(\mathbf{y}_c) = \frac{1}{Z} \exp(-\sum_{c \in \mathcal{C}_G} E_c(\mathbf{y}_c)) \\ =& \frac{1}{Z} \exp(-E(\mathbf{y})) \end{aligned}$$

Hence, $p(\mathbf{y})$ is completely determined by $E(\mathbf{y})$

Energy minimization

Our goal is to solve $\mathbf{y}^* \in \operatorname{argmax}_{\mathbf{y}\in\mathcal{Y}} p(\mathbf{y})$

Energy minimization

Our goal is to solve $\mathbf{y}^* \in \operatorname{argmax}_{\mathbf{y}\in\mathcal{Y}} p(\mathbf{y})$

$$\operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} p(\mathbf{y}) = \operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} \frac{1}{Z}\exp(-E(\mathbf{y}))$$

Energy minimization

Our goal is to solve $\mathbf{y}^* \in \operatorname{argmax}_{\mathbf{y}\in\mathcal{Y}} p(\mathbf{y})$

$$\begin{aligned}\operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} p(\mathbf{y}) &= \operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} \frac{1}{Z}\exp(-E(\mathbf{y})) \\ &= \operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} \exp(-E(\mathbf{y}))\end{aligned}$$

Energy minimization

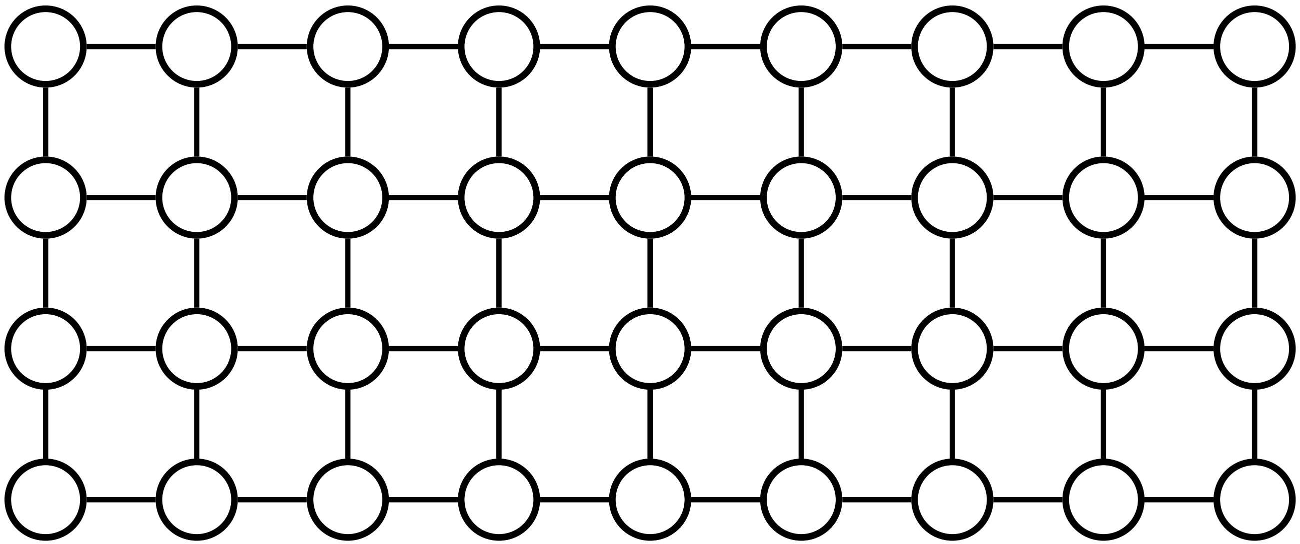
Our goal is to solve $\mathbf{y}^* \in \operatorname{argmax}_{\mathbf{y}\in\mathcal{Y}} p(\mathbf{y})$

$$\begin{aligned}\operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} p(\mathbf{y}) &= \operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} \frac{1}{Z}\exp(-E(\mathbf{y})) \\ &= \operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} \exp(-E(\mathbf{y})) \\ &= \operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} -E(\mathbf{y})\end{aligned}$$

Our goal is to solve $\mathbf{y}^* \in \operatorname{argmax}_{\mathbf{y}\in\mathcal{Y}} p(\mathbf{y})$

$$\begin{aligned}\operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} p(\mathbf{y}) &= \operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} \frac{1}{Z}\exp(-E(\mathbf{y}))\\ &= \operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} \exp(-E(\mathbf{y}))\\ &= \operatorname*{argmax}_{\mathbf{y}\in\mathcal{Y}} -E(\mathbf{y})\\ &= \operatorname*{argmin}_{\mathbf{y}\in\mathcal{Y}} E(\mathbf{y})\ .\end{aligned}$$

Energy



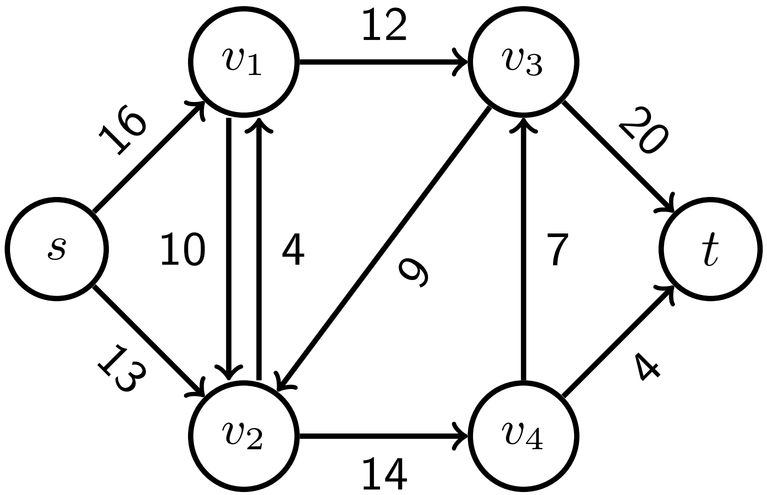
In practice, one typically models the energy function directly

$$E(\mathbf{y}) = \sum_{i \in \mathcal{V}} E_i(y_i) + \sum_{(i,j \in \mathcal{E})} E_{ij}(y_i, y_j)$$

minCut/maxFlow

Graph cut

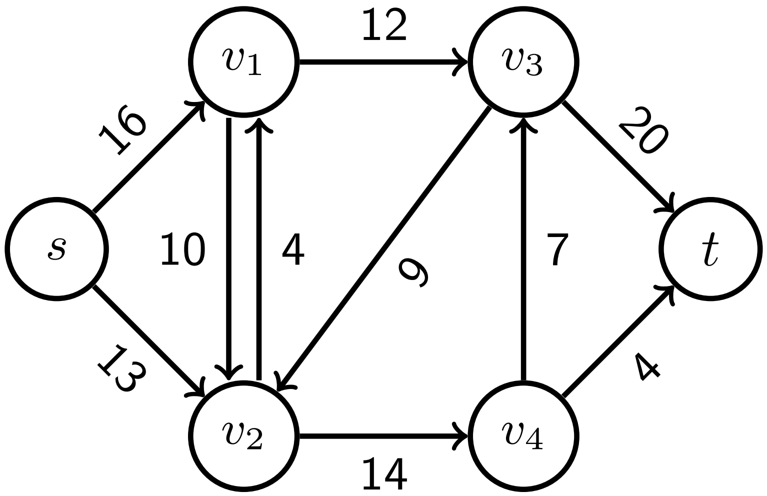
Assume a *weighted directed graph* $G = (\mathcal{V}, \mathcal{E}, c)$



Graph cut

Assume a *weighted directed graph* $G = (\mathcal{V}, \mathcal{E}, c)$

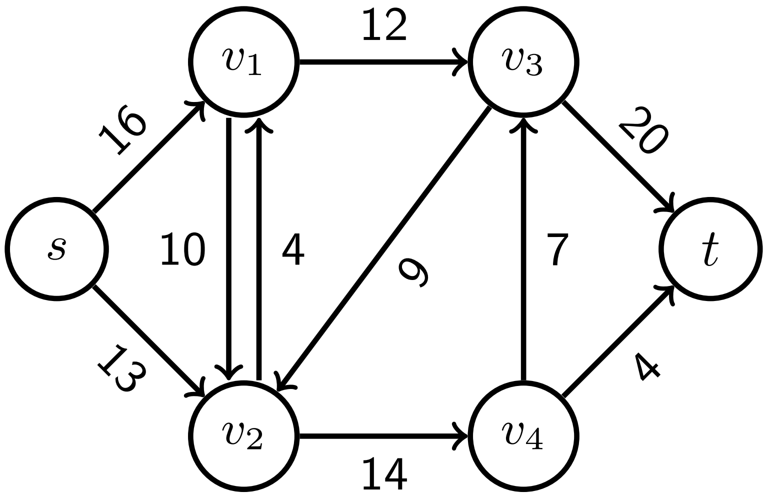
- $\mathcal{V} = \{1, \ldots, n\}$ is a finite set of nodes,



Graph cut

Assume a *weighted directed graph* $G = (\mathcal{V}, \mathcal{E}, c)$

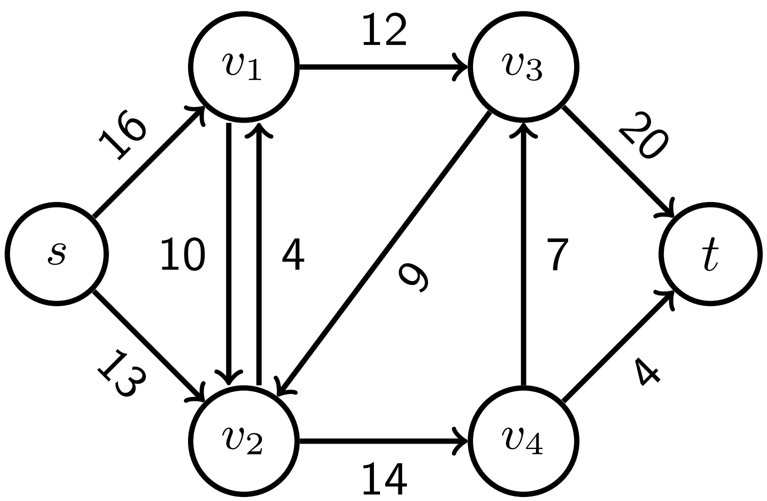
- $\mathcal{V} = \{1, \ldots, n\}$ is a finite set of nodes,
- $\mathcal{E} \subseteq \{(i, j) \in \mathcal{V} \times \mathcal{V} \mid i \neq j\}$ is the set of edges,



Graph cut

Assume a *weighted directed graph* $G = (\mathcal{V}, \mathcal{E}, c)$

- $\mathcal{V} = \{1, \ldots, n\}$ is a finite set of nodes,
- $\mathcal{E} \subseteq \{(i, j) \in \mathcal{V} \times \mathcal{V} \mid i \neq j\}$ is the set of edges,
- $c : \mathcal{V} \times \mathcal{V} \rightarrow \mathbb{R}$ is a weight function. (For any $(i, j) \notin \mathcal{E}$, $c(i, j) = 0$.)

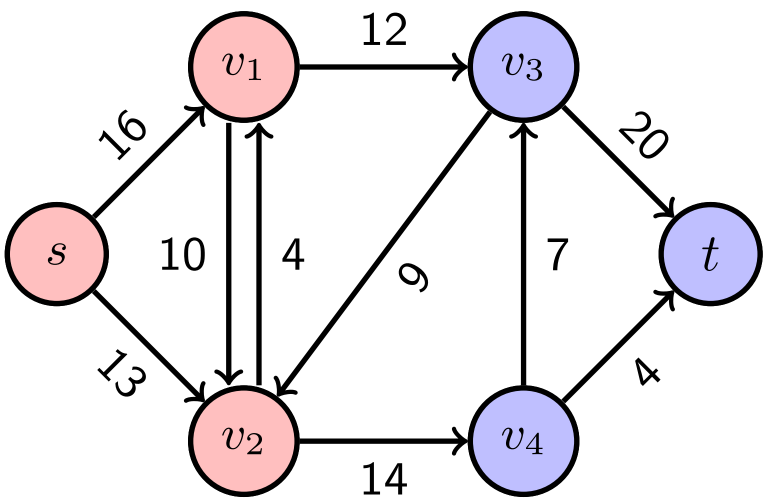


Graph cut

Assume a *weighted directed graph* $G = (\mathcal{V}, \mathcal{E}, c)$

- $\mathcal{V} = \{1, \ldots, n\}$ is a finite set of nodes,
- $\mathcal{E} \subseteq \{(i, j) \in \mathcal{V} \times \mathcal{V} \mid i \neq j\}$ is the set of edges,
- $c : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ is a weight function. (For any $(i, j) \notin \mathcal{E}$, $c(i, j) = 0$.)

A *cut* $(\mathcal{S}, \mathcal{T})$ of G is a *disjoint* partition of $\mathcal{V}$ into $\mathcal{S}$ and $\mathcal{T} = \mathcal{V} \backslash \mathcal{S}$.



Graph cut

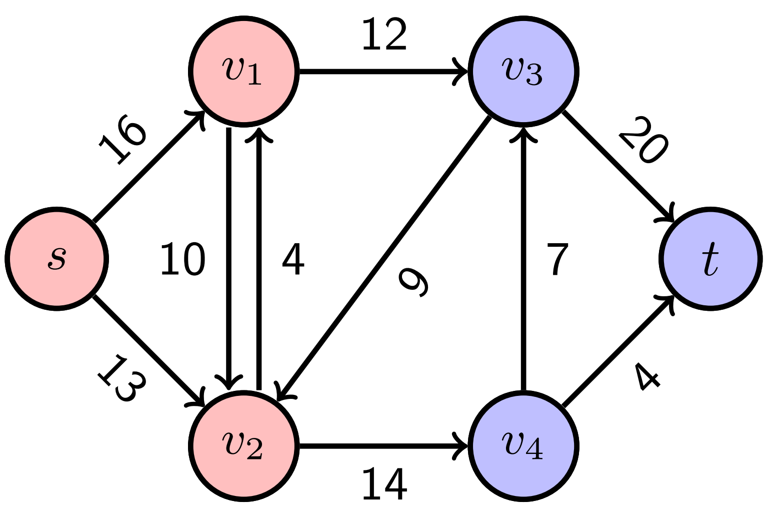
Assume a *weighted directed graph* $G = (\mathcal{V}, \mathcal{E}, c)$

- $\mathcal{V} = \{1, \ldots, n\}$ is a finite set of nodes,
- $\mathcal{E} \subseteq \{(i,j) \in \mathcal{V} \times \mathcal{V} \mid i \neq j\}$ is the set of edges,
- $c : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ is a weight function. (For any $(i,j) \notin \mathcal{E}$, $c(i,j) = 0$.)

A *cut* $(\mathcal{S}, \mathcal{T})$ of G is a *disjoint* partition of $\mathcal{V}$ into $\mathcal{S}$ and $\mathcal{T} = \mathcal{V} \backslash \mathcal{S}$.

The *capacity* of the cut $(\mathcal{S}, \mathcal{T})$ is defined as

$$\text{cut}(\mathcal{S}, \mathcal{T}) = \sum_{(i,j) \in \mathcal{S} \times \mathcal{T}} c(i,j) \; .$$



Graph cut

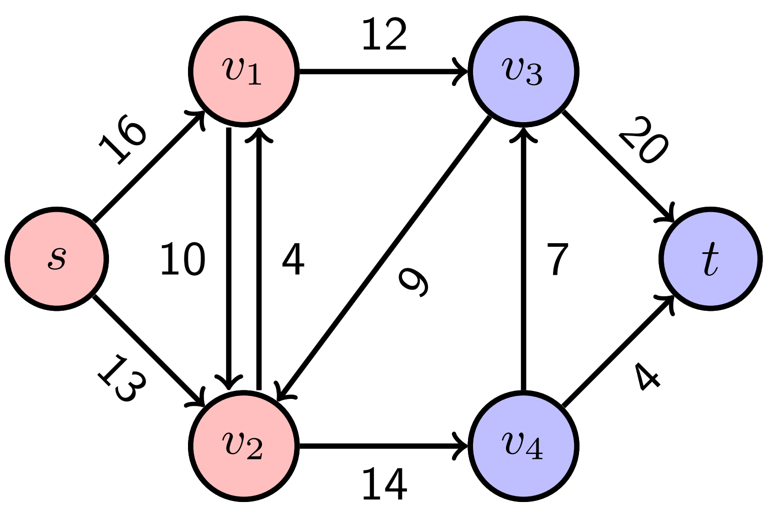
Assume a *weighted directed graph* $G = (\mathcal{V}, \mathcal{E}, c)$

- $\mathcal{V} = \{1, \ldots, n\}$ is a finite set of nodes,
- $\mathcal{E} \subseteq \{(i,j) \in \mathcal{V} \times \mathcal{V} \mid i \neq j\}$ is the set of edges,
- $c : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ is a weight function. (For any $(i,j) \notin \mathcal{E}$, $c(i,j) = 0$.)

A *cut* $(\mathcal{S}, \mathcal{T})$ of G is a *disjoint* partition of $\mathcal{V}$ into $\mathcal{S}$ and $\mathcal{T} = \mathcal{V} \backslash \mathcal{S}$.

The *capacity* of the cut $(\mathcal{S}, \mathcal{T})$ is defined as

$$\text{cut}(\mathcal{S}, \mathcal{T}) = \sum_{(i,j) \in \mathcal{S} \times \mathcal{T}} c(i,j) \; .$$



Assume distinct nodes $s, t \in \mathcal{V}$, a cut $(\mathcal{S}, \mathcal{T})$ is called $s-t$ *cut* if $s \in \mathcal{S}$ and $t \in \mathcal{T}$.

Graph cut

Assume a *weighted directed graph* $G = (\mathcal{V}, \mathcal{E}, c)$

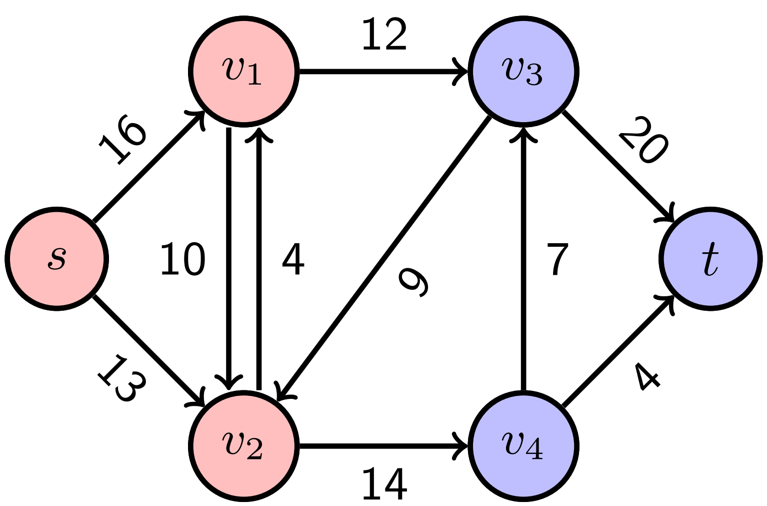
- $\mathcal{V} = \{1, \ldots, n\}$ is a finite set of nodes,
- $\mathcal{E} \subseteq \{(i,j) \in \mathcal{V} \times \mathcal{V} \mid i \neq j\}$ is the set of edges,
- $c : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ is a weight function. (For any $(i,j) \notin \mathcal{E}$, $c(i,j) = 0$.)

A *cut* $(\mathcal{S}, \mathcal{T})$ of G is a *disjoint* partition of $\mathcal{V}$ into $\mathcal{S}$ and $\mathcal{T} = \mathcal{V} \backslash \mathcal{S}$.

The *capacity* of the cut $(\mathcal{S}, \mathcal{T})$ is defined as

$$\text{cut}(\mathcal{S}, \mathcal{T}) = \sum_{(i,j) \in \mathcal{S} \times \mathcal{T}} c(i,j) \ .$$

Assume distinct nodes $s, t \in \mathcal{V}$, a cut $(\mathcal{S}, \mathcal{T})$ is called $s - t$ *cut* if $s \in \mathcal{S}$ and $t \in \mathcal{T}$.



The *minimum* $s - t$ *cut problem* is to find an $s - t$ cut with the lowest cost.

Assume a *weighted directed graph* $G = (\mathcal{V}, \mathcal{E}, c)$

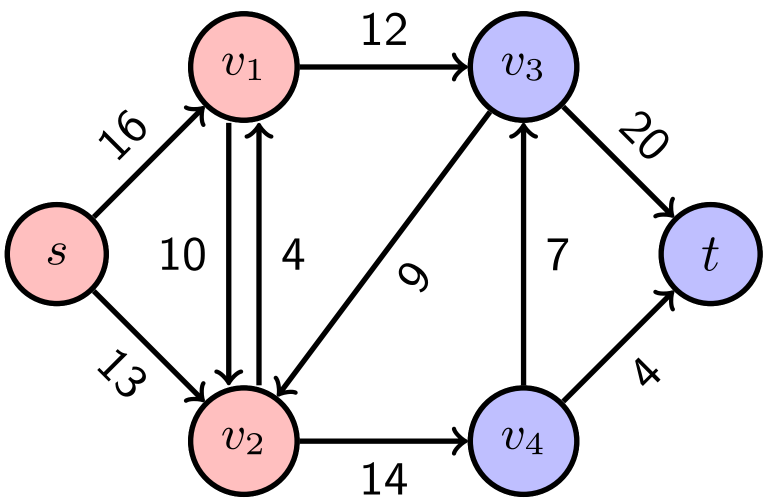
- $\mathcal{V} = \{1, \dots, n\}$ is a finite set of nodes,
- $\mathcal{E} \subseteq \{(i, j) \in \mathcal{V} \times \mathcal{V} \mid i \neq j\}$ is the set of edges,
- $c : \mathcal{V} \times \mathcal{V} \to \mathbb{R}$ is a weight function. (For any $(i, j) \notin \mathcal{E}$, $c(i, j) = 0$.)

A *cut* $(\mathcal{S}, \mathcal{T})$ of G is a *disjoint* partition of $\mathcal{V}$ into $\mathcal{S}$ and $\mathcal{T} = \mathcal{V} \backslash \mathcal{S}$.

The *capacity* of the cut $(\mathcal{S}, \mathcal{T})$ is defined as

$$\text{cut}(\mathcal{S}, \mathcal{T}) = \sum_{(i,j) \in \mathcal{S} \times \mathcal{T}} c(i, j) \; .$$

Assume distinct nodes $s, t \in \mathcal{V}$, a cut $(\mathcal{S}, \mathcal{T})$ is called $s - t$ *cut* if $s \in \mathcal{S}$ and $t \in \mathcal{T}$.



The *minimum* $s - t$ *cut problem* is to find an $s - t$ cut with the lowest cost.

<u>*Example*</u>: $\text{cut}(\mathcal{S}, \mathcal{T}) = c(v_1, v_3) + c(v_2, v_4) = 12 + 14 = 26$.

Flow network and flow

Let $G = (\mathcal{V}, \mathcal{E}, c)$ be a *directed weighted graph* with **non-negative** edge weights.

Flow network and flow

Let $G = (\mathcal{V}, \mathcal{E}, c)$ be a *directed weighted graph* with **non-negative** edge weights. Given two distinct nodes, a **source** s and a **sink** t, we call $(\mathcal{V}, \mathcal{E}, c, s, t)$ a *flow network*

Flow network and flow

Let $G = (\mathcal{V}, \mathcal{E}, c)$ be a *directed weighted graph* with **non-negative** edge weights. Given two distinct nodes, a **source** s and a **sink** t, we call $(\mathcal{V}, \mathcal{E}, c, s, t)$ a *flow network*

Let $(\mathcal{V}, \mathcal{E}, c, s, t)$ be a flow network. A function $f : \mathcal{E} \to \mathbb{R}^+$ is called a *flow* if it satisfies the following two properties:

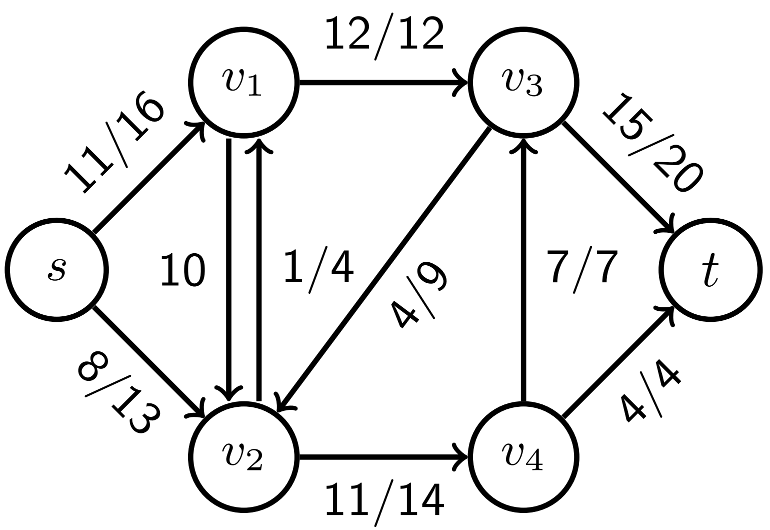
Flow network and flow

Let $G = (\mathcal{V}, \mathcal{E}, c)$ be a *directed weighted graph* with **non-negative** edge weights. Given two distinct nodes, a **source** s and a **sink** t, we call $(\mathcal{V}, \mathcal{E}, c, s, t)$ a *flow network*

Let $(\mathcal{V}, \mathcal{E}, c, s, t)$ be a flow network. A function $f : \mathcal{E} \to \mathbb{R}^+$ is called a *flow* if it satisfies the following two properties:

1. $f(i,j) \leqslant c(i,j)$ for all $(i,j) \in \mathcal{E}$.
2. For all $i \in \mathcal{V} \backslash \{s,t\}$

$$\sum_{(i,j)\in\mathcal{E}} f(i,j) = \sum_{(j,i)\in\mathcal{E}} f(j,i) \, .$$



The edges are labeled by $f(i,j)/c(i,j)$.

Only positive $f(i,j)$ are shown.

The value of a flow

The *value* of a flow f is defined as

$$|f| \stackrel{\Delta}{=} \sum_{(s,i)\in\mathcal{E}} f(s,i) = -\sum_{(i,t)\in\mathcal{E}} f(i,t)\ .$$

The value of a flow

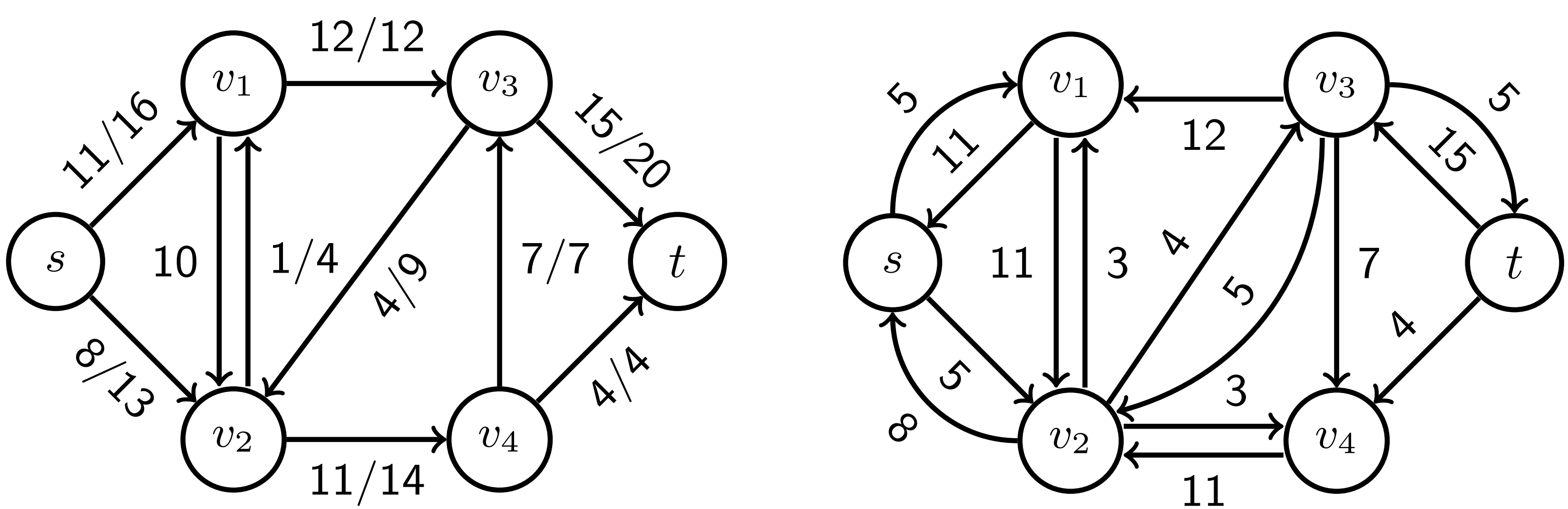
The *value* of a flow f is defined as

$$|f| \triangleq \sum_{(s,i)\in\mathcal{E}} f(s,i) = -\sum_{(i,t)\in\mathcal{E}} f(i,t)\ .$$

The *maximum-flow problem* is to find a *flow* f with the highest cost for a given flow network G

Residual network

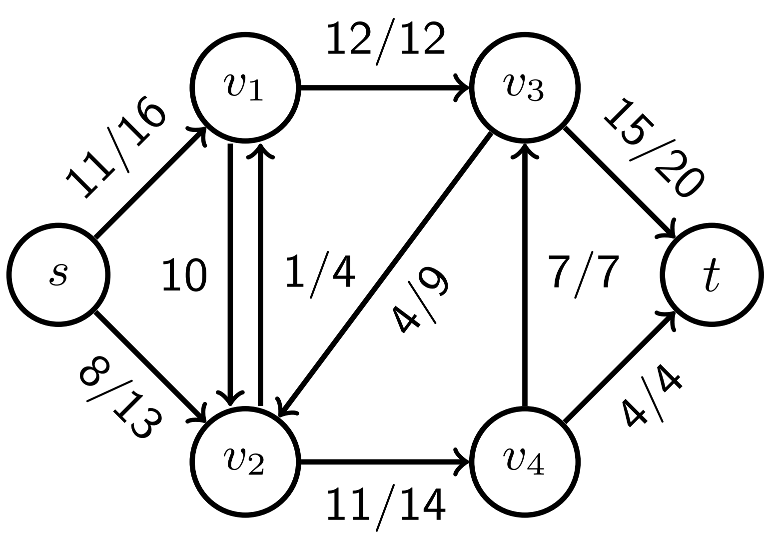
Let $G = (\mathcal{V}, \mathcal{E}, c, s, t)$ be a *flow network* and let f be a *flow* in G. The *weighted directed graph* $G_f = (\mathcal{V}, \mathcal{E}_f, c_f)$ is called *residual network* of G induced by f, where

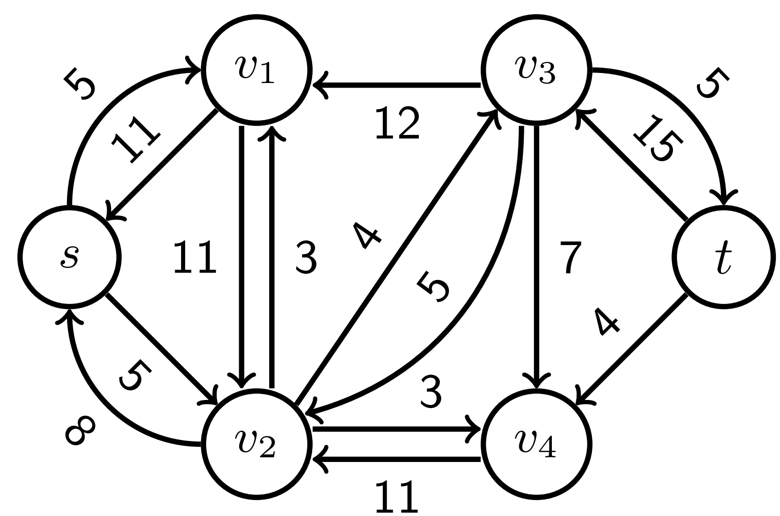


Residual network

Let $G = (\mathcal{V}, \mathcal{E}, c, s, t)$ be a *flow network* and let f be a *flow* in G. The *weighted directed graph* $G_f = (\mathcal{V}, \mathcal{E}_f, c_f)$ is called *residual network* of G induced by f, where

$$c_f(i,j) = c(i,j) - f(i,j)$$

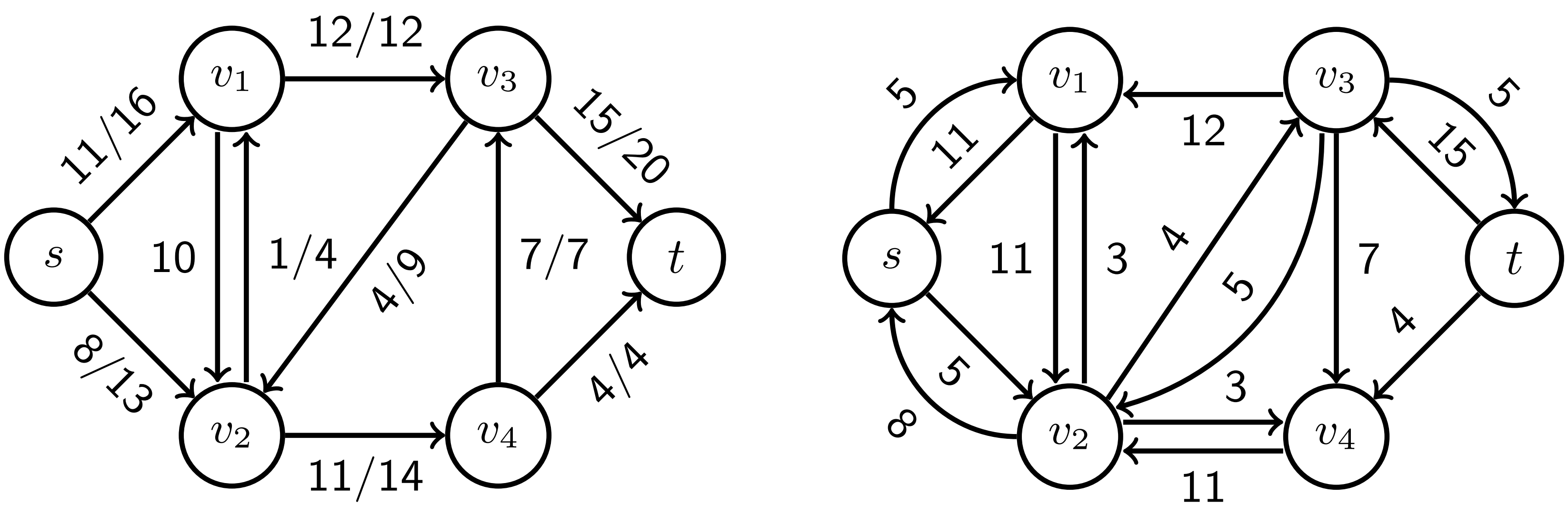




Residual network

Let $G = (\mathcal{V}, \mathcal{E}, c, s, t)$ be a *flow network* and let f be a *flow* in G. The *weighted directed graph* $G_f = (\mathcal{V}, \mathcal{E}_f, c_f)$ is called *residual network* of G induced by f, where

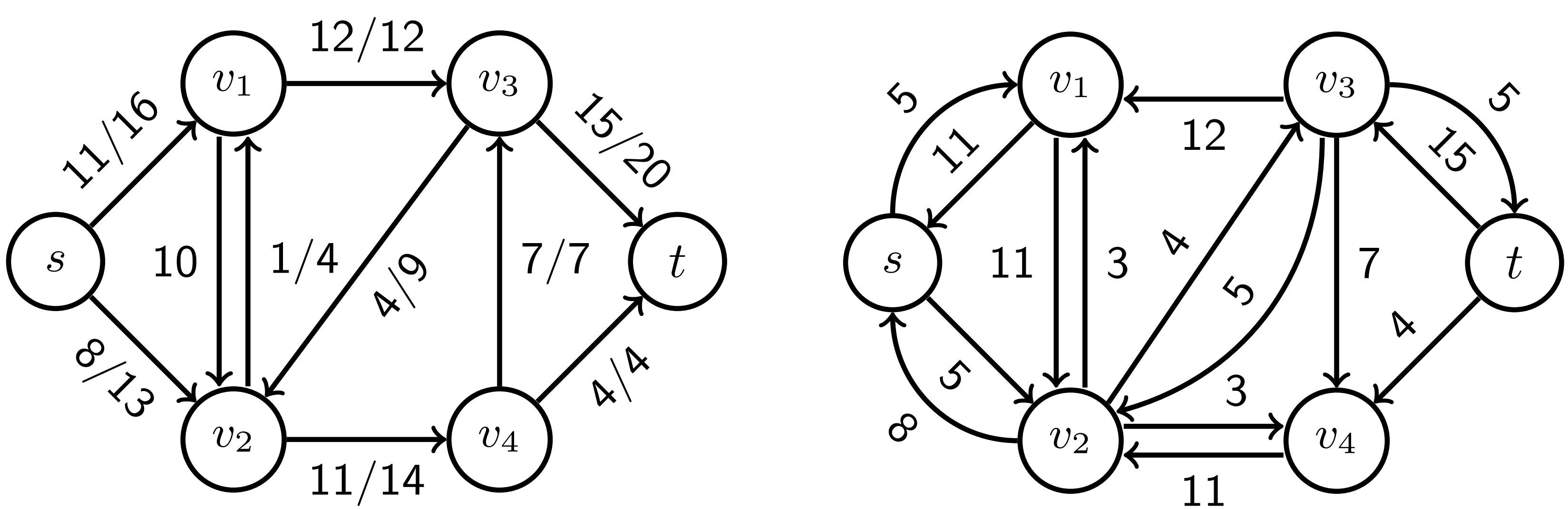
$$c_f(i,j) = c(i,j) - f(i,j)$$
$$\mathcal{E}_f = \{(i,j) \in \mathcal{V} \times \mathcal{V} : c_f(i,j) > 0\}$$



Residual network

Let $G = (\mathcal{V}, \mathcal{E}, c, s, t)$ be a *flow network* and let f be a *flow* in G. The *weighted directed graph* $G_f = (\mathcal{V}, \mathcal{E}_f, c_f)$ is called *residual network* of G induced by f, where

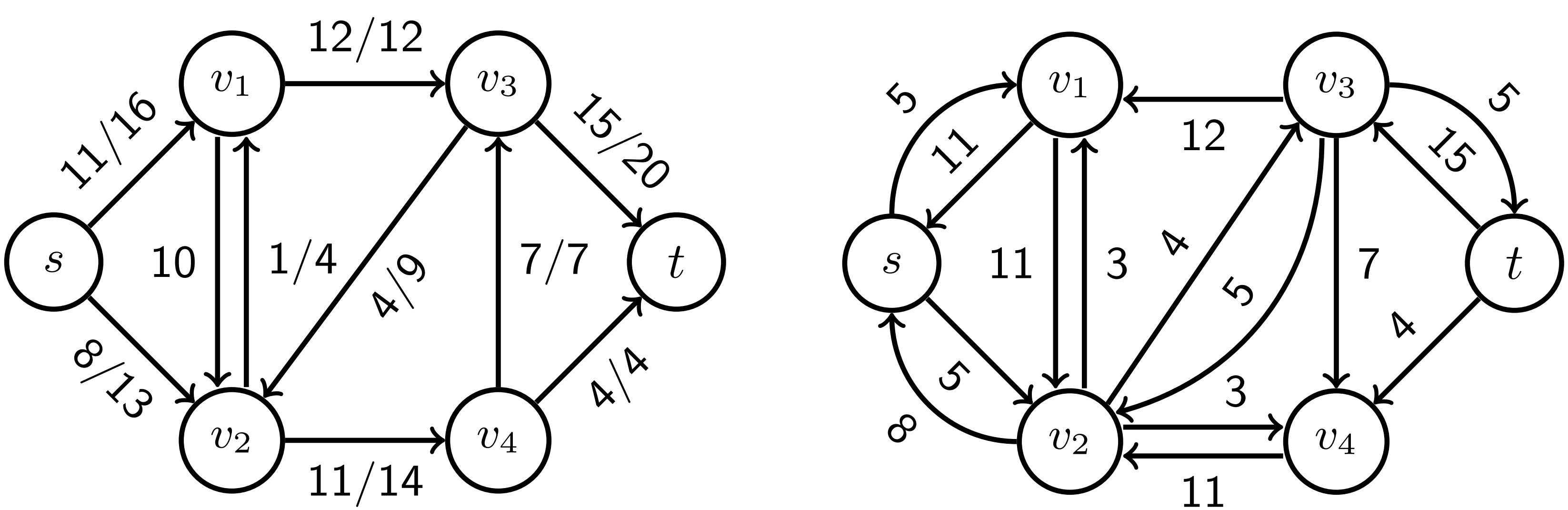
$$c_f(i,j) = c(i,j) - f(i,j)$$
$$\mathcal{E}_f = \{(i,j) \in \mathcal{V} \times \mathcal{V} : c_f(i,j) > 0\}$$



Residual network

Let $G = (\mathcal{V}, \mathcal{E}, c, s, t)$ be a *flow network* and let f be a *flow* in G. The *weighted directed graph* $G_f = (\mathcal{V}, \mathcal{E}_f, c_f)$ is called *residual network* of G induced by f, where

$$c_f(i,j) = c(i,j) - f(i,j)$$
$$\mathcal{E}_f = \{(i,j) \in \mathcal{V} \times \mathcal{V} : c_f(i,j) > 0\}$$



A path p from s to t in G_f is called an *augmenting path*.

Max-flow–min-cut theorem

Let f be a *flow* in a *flow network* $G = (\mathcal{V}, \mathcal{E}, c, s, t)$. Then the following conditions are equivalent:

1) f is a maximal flow in G
2) The residual graph G_f contains no augmenting paths
3) $|f| = \text{cut}(\mathcal{S}, \mathcal{T})$ for some $s - t$ cut of G

Max-flow–min-cut theorem

Let f be a *flow* in a *flow network* $G = (\mathcal{V}, \mathcal{E}, c, s, t)$. Then the following conditions are equivalent:

1) f is a maximal flow in G
2) The residual graph G_f contains no augmenting paths
3) $|f| = \text{cut}(\mathcal{S}, \mathcal{T})$ for some $s - t$ cut of G

In general, for **any** flow f in G the following holds:

$$|f| = \sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{T}} f(i, j)$$

Max-flow–min-cut theorem

Let f be a *flow* in a *flow network* $G = (\mathcal{V}, \mathcal{E}, c, s, t)$. Then the following conditions are equivalent:

1) f is a maximal flow in G
2) The residual graph G_f contains no augmenting paths
3) $|f| = \text{cut}(\mathcal{S}, \mathcal{T})$ for some $s - t$ cut of G

In general, for **any** flow f in G the following holds:

$$|f| = \sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{T}} f(i,j) \leqslant \sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{T}} c(i,j)$$

Max-flow–min-cut theorem

Let f be a *flow* in a *flow network* $G = (\mathcal{V}, \mathcal{E}, c, s, t)$. Then the following conditions are equivalent:

1) f is a maximal flow in G
2) The residual graph G_f contains no augmenting paths
3) $|f| = \text{cut}(\mathcal{S}, \mathcal{T})$ for some $s - t$ cut of G

In general, for **any** flow f in G the following holds:

$$|f| = \sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{T}} f(i,j) \leqslant \sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{T}} c(i,j) = \text{cut}(\mathcal{S}, \mathcal{T})$$

Max-flow–min-cut theorem

Let f be a *flow* in a *flow network* $G = (\mathcal{V}, \mathcal{E}, c, s, t)$. Then the following conditions are equivalent:

1) f is a maximal flow in G
2) The residual graph G_f contains no augmenting paths
3) $|f| = \text{cut}(\mathcal{S}, \mathcal{T})$ for some $s - t$ cut of G

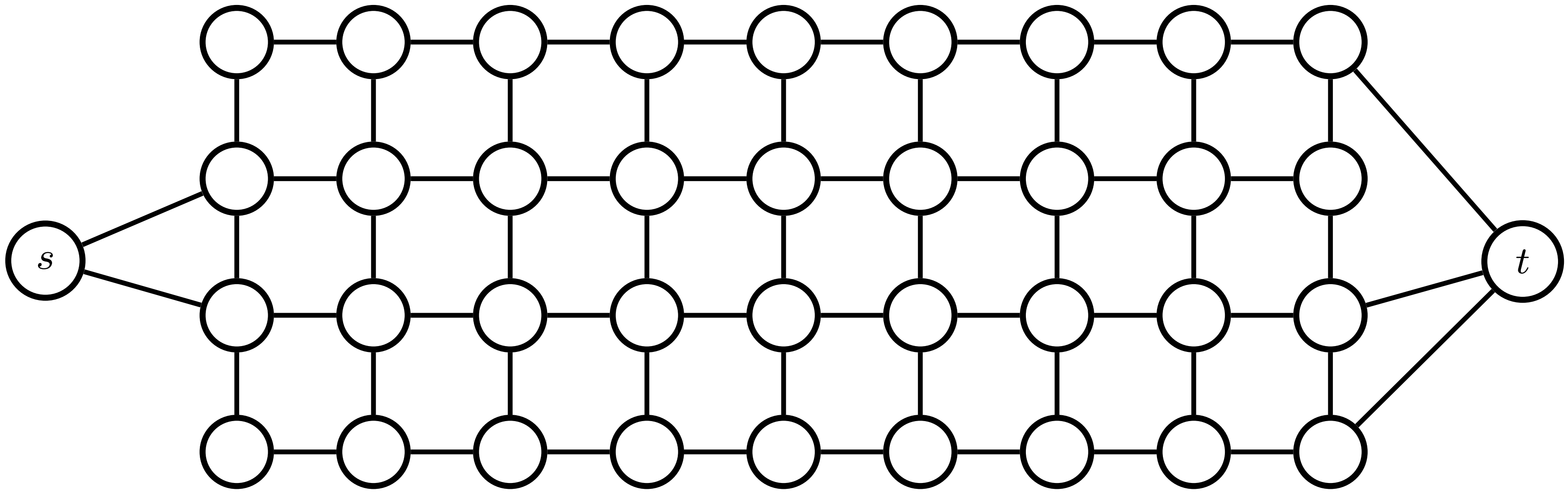
In general, for **any** flow f in G the following holds:

$$|f| = \sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{T}} f(i,j) \leqslant \sum_{i \in \mathcal{S}} \sum_{j \in \mathcal{T}} c(i,j) = \text{cut}(\mathcal{S}, \mathcal{T})$$

Hence $|f| = \text{cut}(\mathcal{S}, \mathcal{T})$ is maximal (equivalently $\text{cut}(\mathcal{S}, \mathcal{T})$ is minimal)

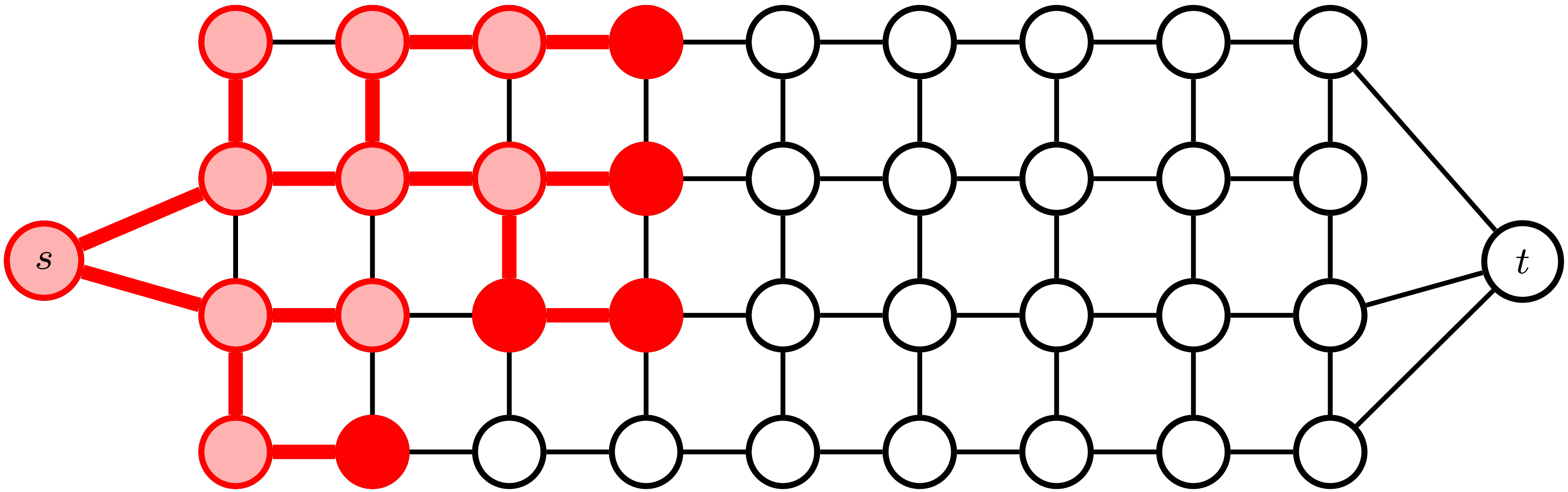
Boykov–Kolmogorov algorithm

Boykov–Kolmogorov algorithm



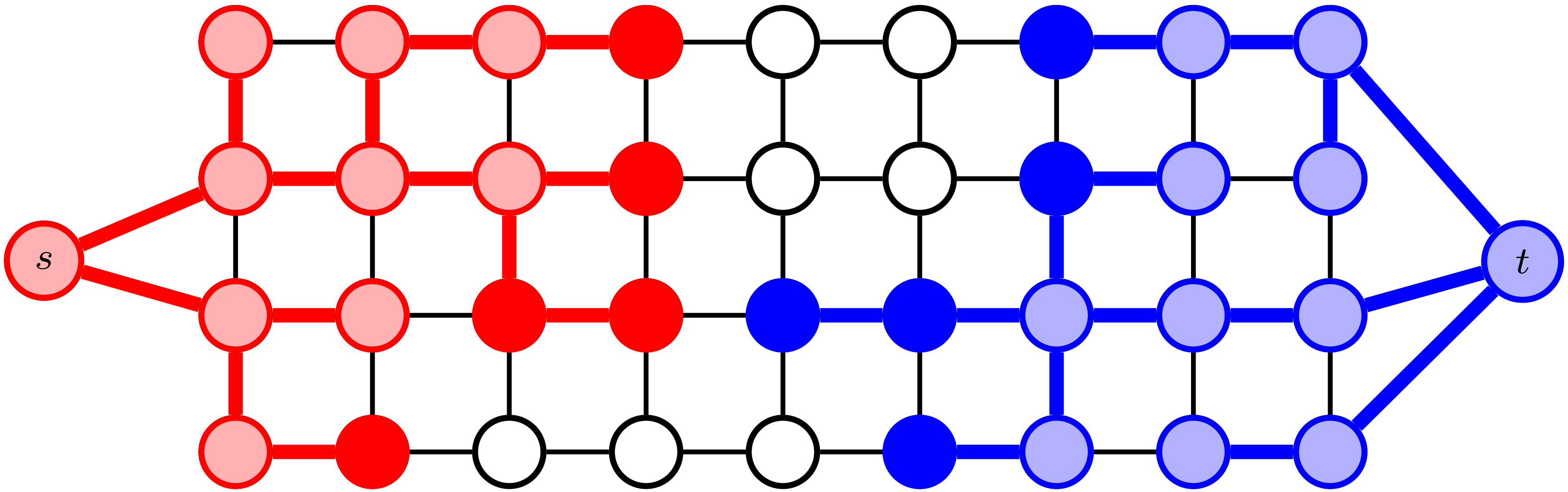
- Main idea: Never start building an *augmenting path* from scratch

Boykov–Kolmogorov algorithm



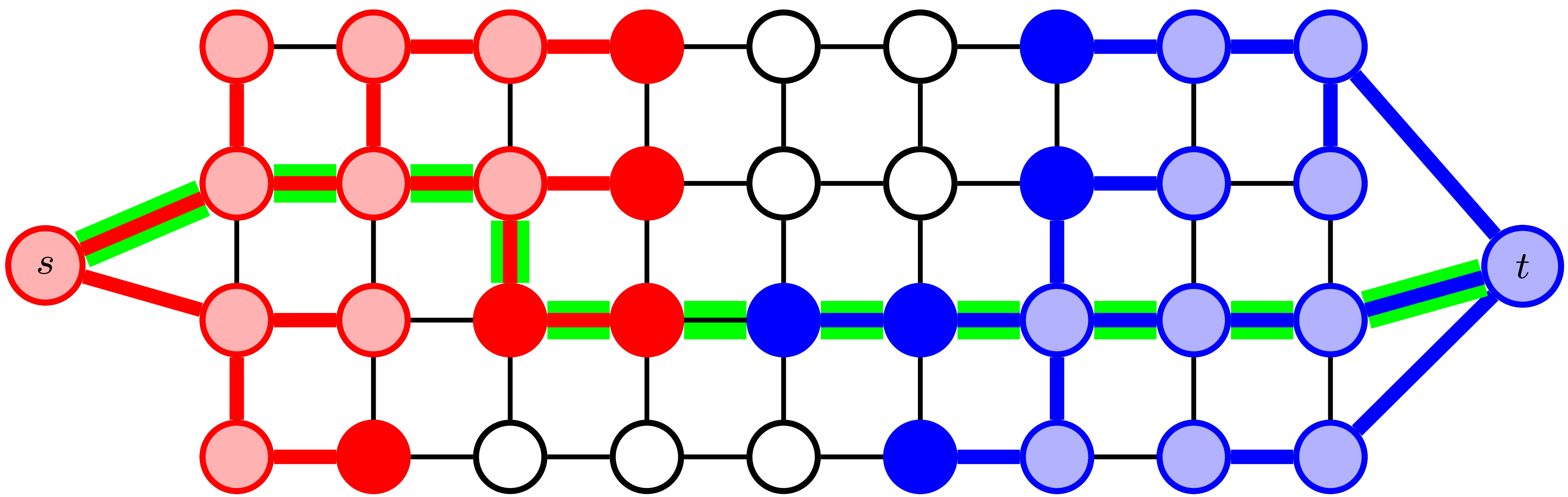
- Main idea: Never start building an *augmenting path* from scratch

Boykov–Kolmogorov algorithm



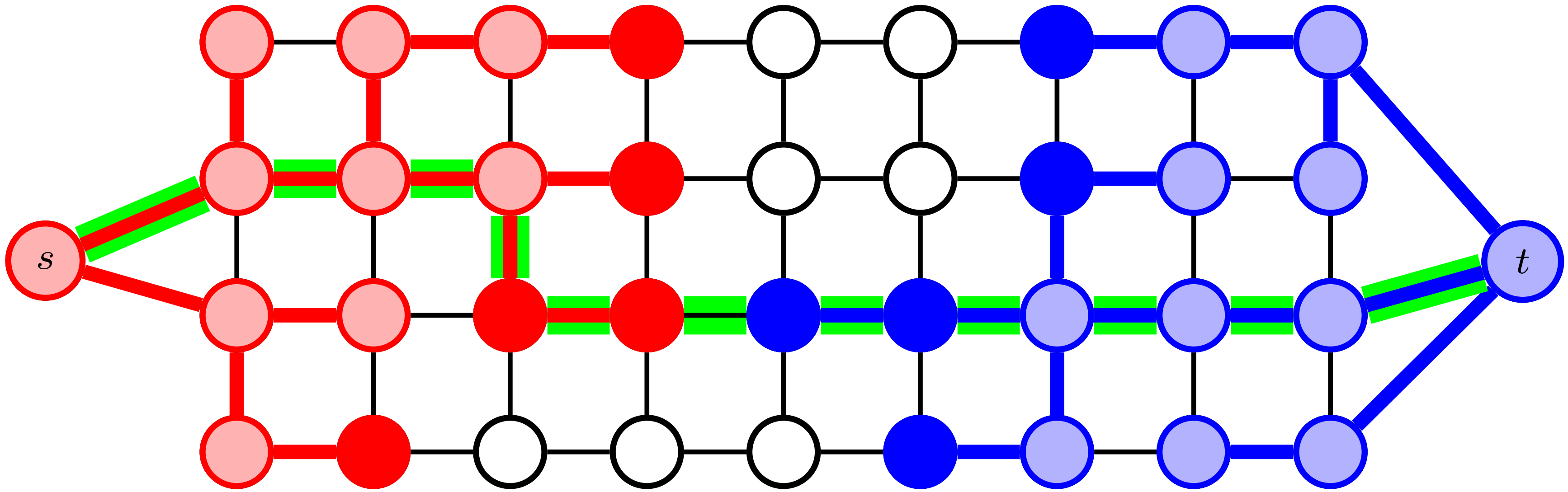
- Main idea: Never start building an *augmenting path* from scratch
- Two non-overlapping search trees S and T with roots at the terminals
- The edges of the trees are *non-saturated*, i.e. $f(i,j) < c(i,j)$

Boykov–Kolmogorov algorithm



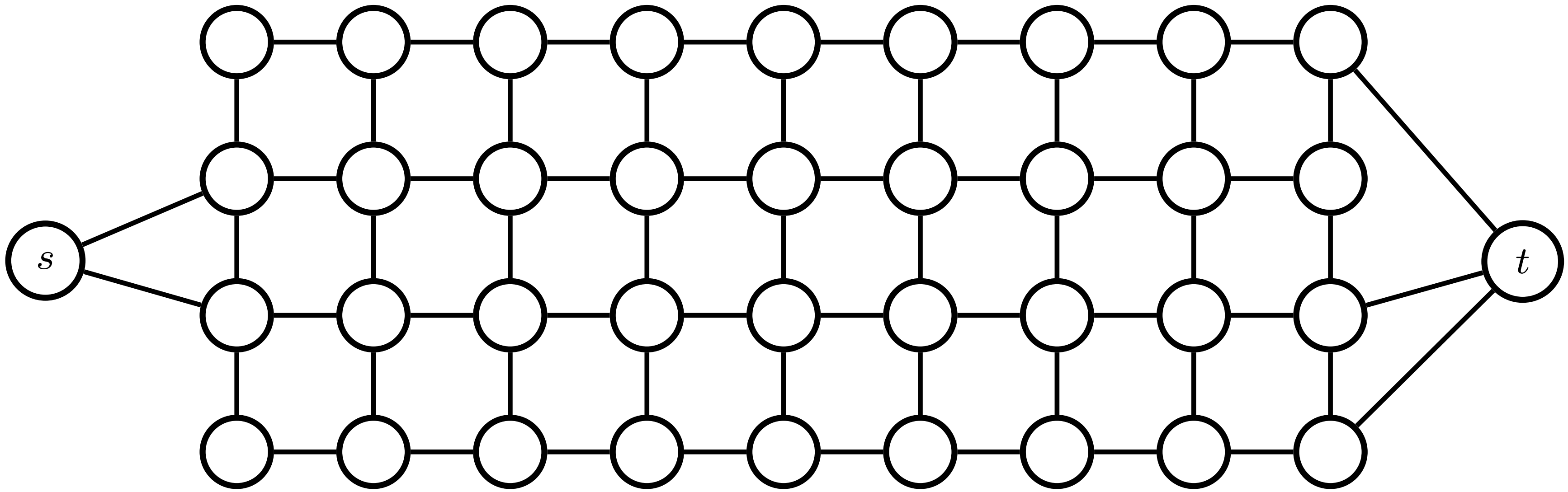
- Main idea: Never start building an *augmenting path* from scratch
- Two non-overlapping search trees S and T with roots at the terminals
- The edges of the trees are *non-saturated*, i.e. $f(i,j) < c(i,j)$
- Active nodes:
- Passive nodes:
- Free nodes:

Boykov–Kolmogorov algorithm



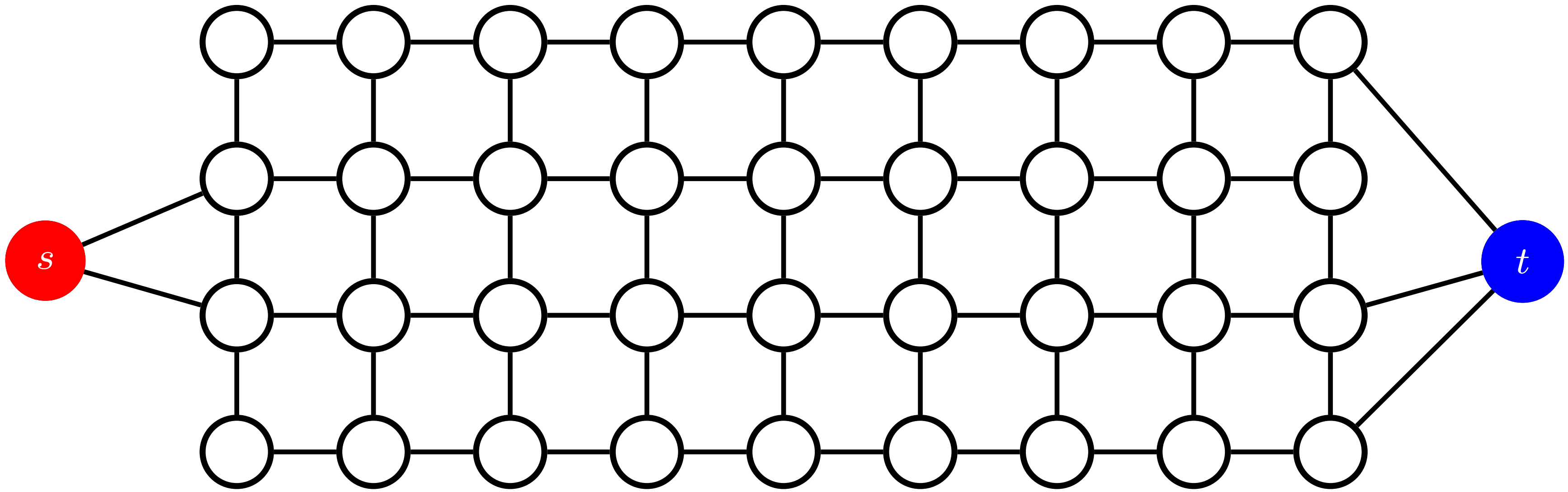

```
while true do
    grow S or T to find an augmenting path P from s to t
    if P = ∅ then
        terminate
    end if
    augment on P
    adopt orphans
end while
```

Growth stage



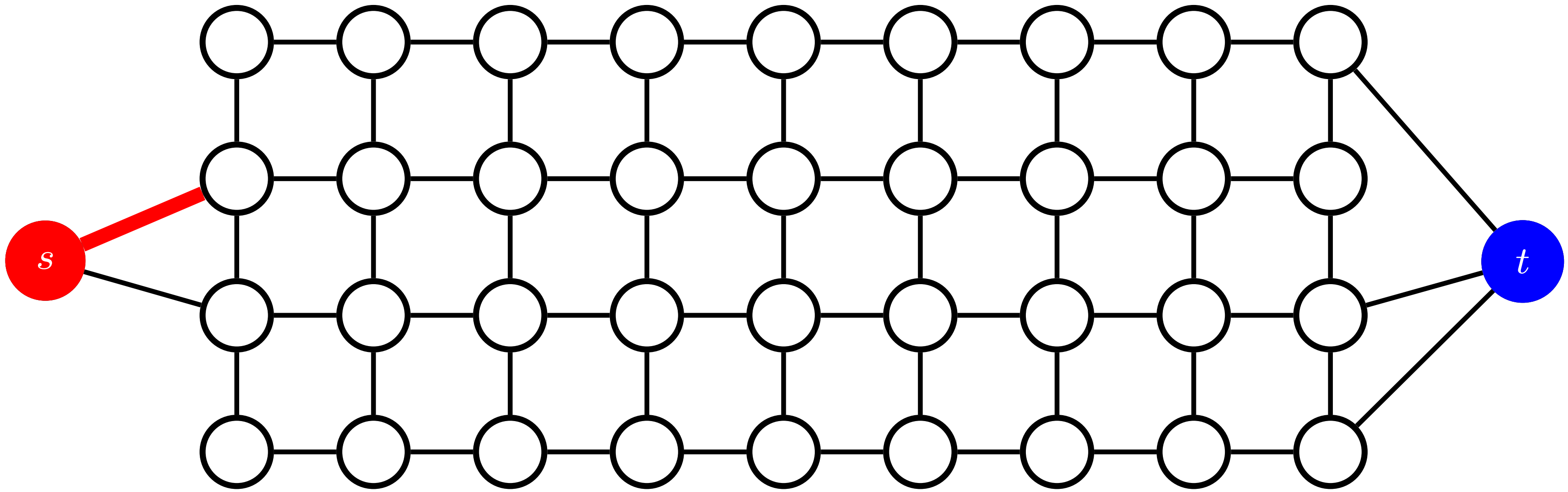
- The *active nodes* explore adjacent edges and acquire new children from a set of *free nodes*
- The newly acquired nodes become *active* members of the corresponding search trees
- The *active node* becomes *passive*, when all of its neighbors are explored
- If an *active node* encounters a neighboring node belonging to the opposite tree, the growth stage terminates

Growth stage



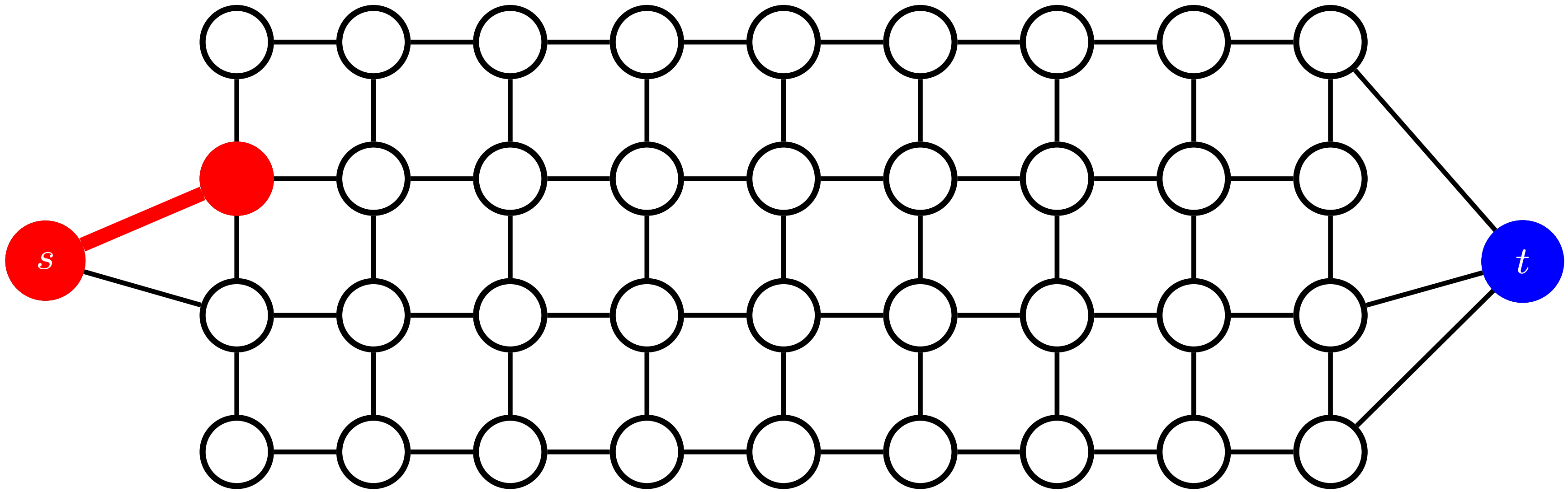
- The *active nodes* explore adjacent edges and acquire new children from a set of *free nodes*
- The newly acquired nodes become *active* members of the corresponding search trees
- The *active node* becomes *passive*, when all of its neighbors are explored
- If an *active node* encounters a neighboring node belonging to the opposite tree, the growth stage terminates

Growth stage



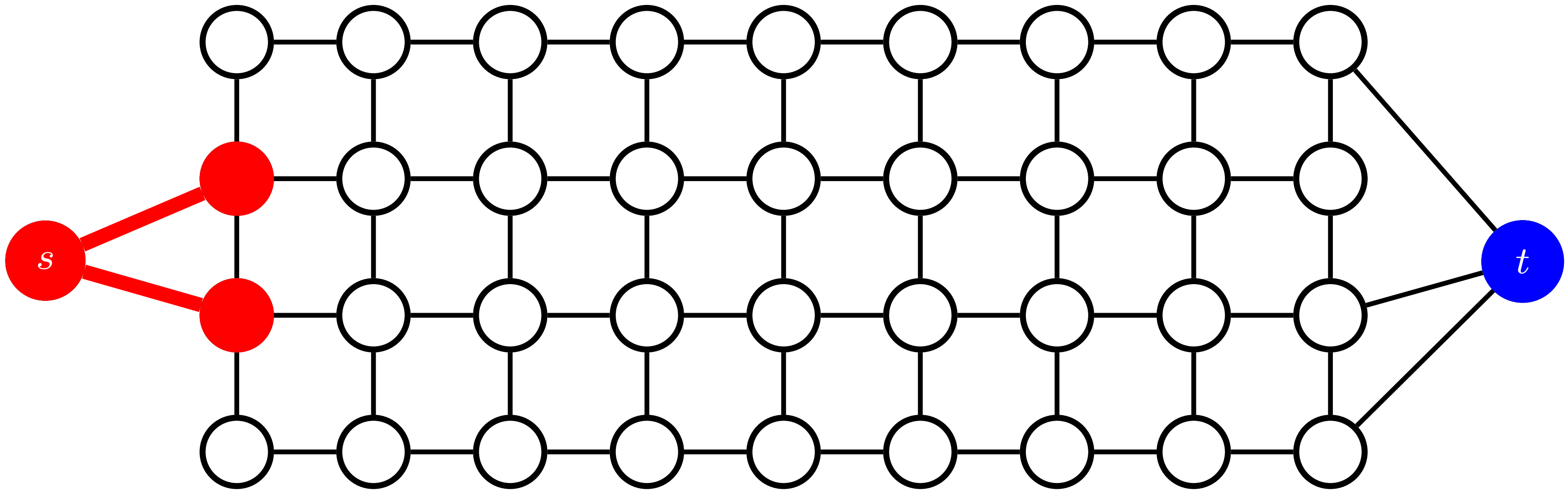
- The *active nodes* explore adjacent edges and acquire new children from a set of *free nodes*
- The newly acquired nodes become *active* members of the corresponding search trees
- The *active node* becomes *passive*, when all of its neighbors are explored
- If an *active node* encounters a neighboring node belonging to the opposite tree, the growth stage terminates

Growth stage



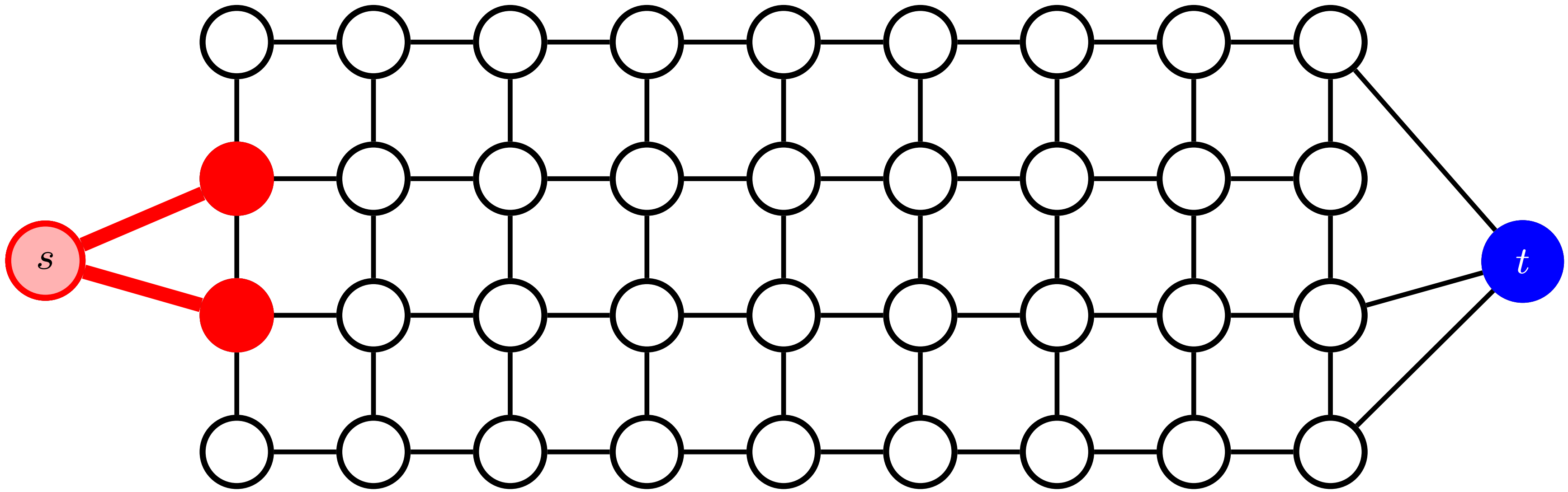
- The *active nodes* explore adjacent edges and acquire new children from a set of *free nodes*
- The newly acquired nodes become *active* members of the corresponding search trees
- The *active node* becomes *passive*, when all of its neighbors are explored
- If an *active node* encounters a neighboring node belonging to the opposite tree, the growth stage terminates

Growth stage



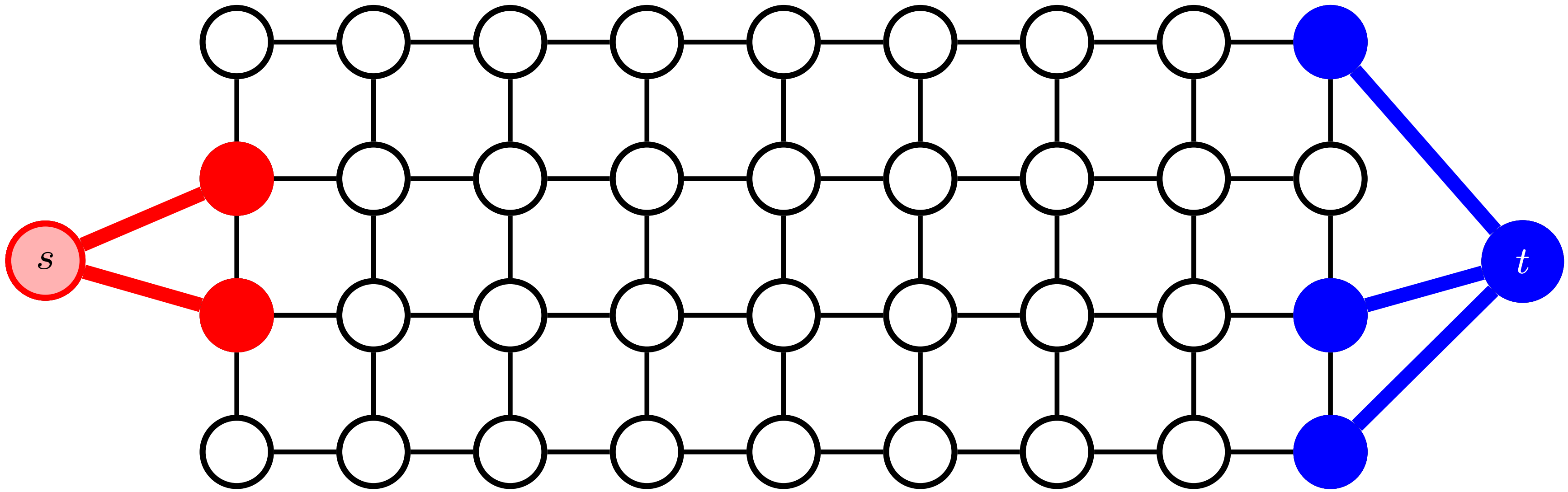
- The *active nodes* explore adjacent edges and acquire new children from a set of *free nodes*
- The newly acquired nodes become *active* members of the corresponding search trees
- The *active node* becomes *passive*, when all of its neighbors are explored
- If an *active node* encounters a neighboring node belonging to the opposite tree, the growth stage terminates

Growth stage



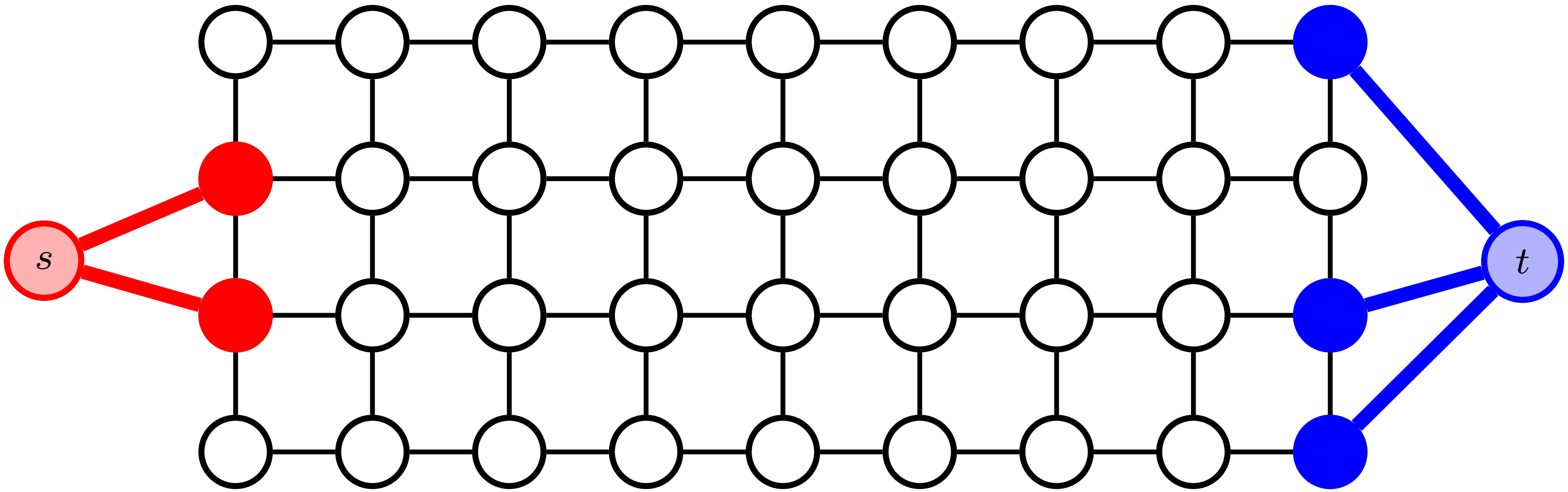
- The *active nodes* explore adjacent edges and acquire new children from a set of *free nodes*
- The newly acquired nodes become *active* members of the corresponding search trees
- The *active node* becomes *passive*, when all of its neighbors are explored
- If an *active node* encounters a neighboring node belonging to the opposite tree, the growth stage terminates

Growth stage



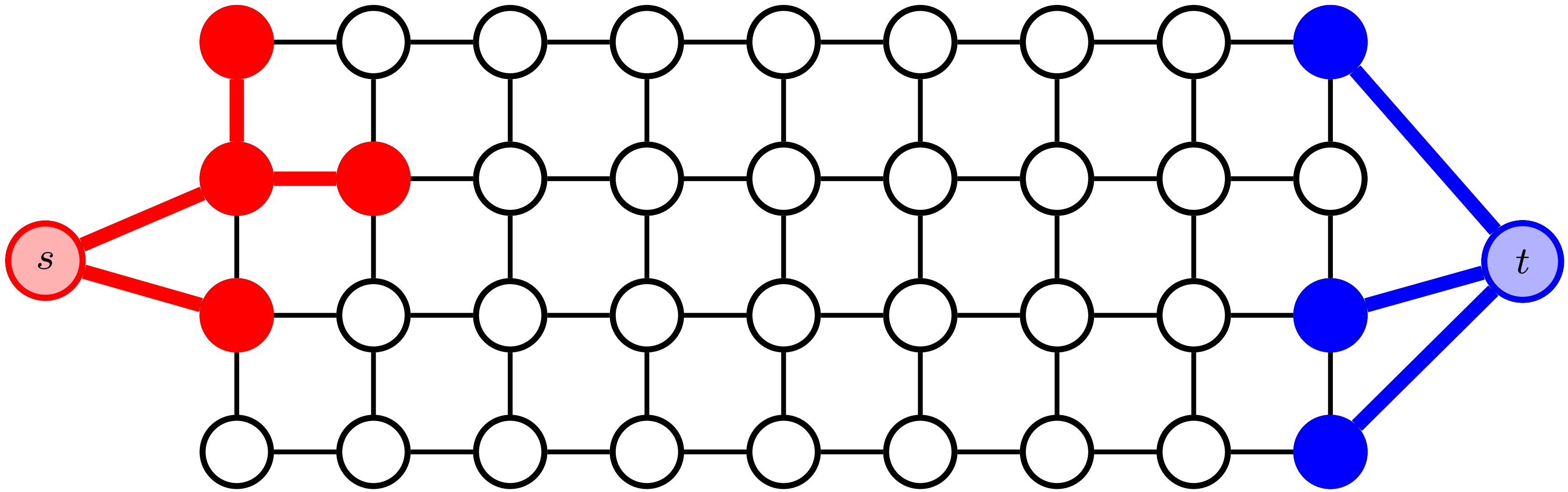
- The *active nodes* explore adjacent edges and acquire new children from a set of *free nodes*
- The newly acquired nodes become *active* members of the corresponding search trees
- The *active node* becomes *passive*, when all of its neighbors are explored
- If an *active node* encounters a neighboring node belonging to the opposite tree, the growth stage terminates

Growth stage



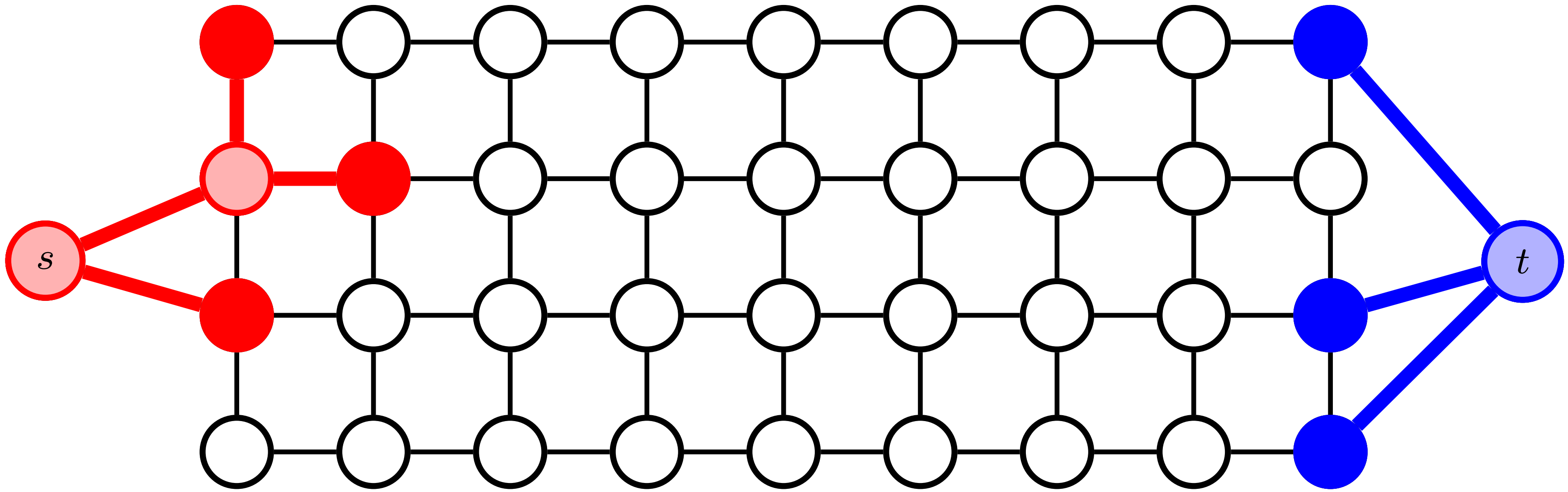
- The *active nodes* explore adjacent edges and acquire new children from a set of *free nodes*
- The newly acquired nodes become *active* members of the corresponding search trees
- The *active node* becomes *passive*, when all of its neighbors are explored
- If an *active node* encounters a neighboring node belonging to the opposite tree, the growth stage terminates

Growth stage



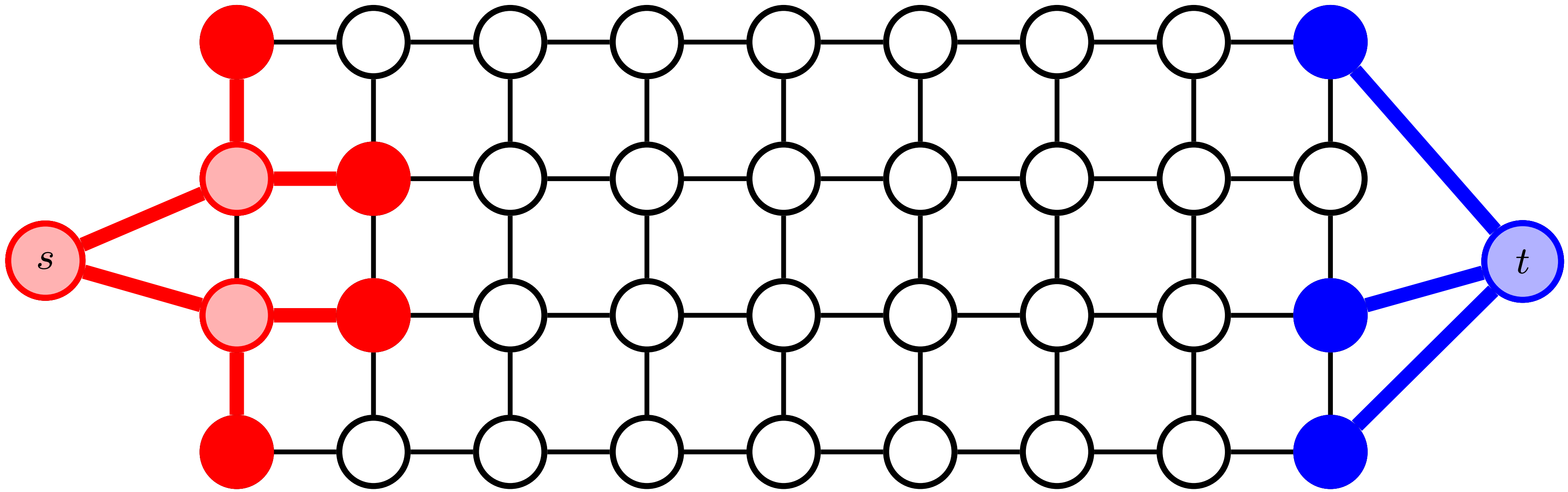
- The *active nodes* explore adjacent edges and acquire new children from a set of *free nodes*
- The newly acquired nodes become *active* members of the corresponding search trees
- The *active node* becomes *passive*, when all of its neighbors are explored
- If an *active node* encounters a neighboring node belonging to the opposite tree, the growth stage terminates

Growth stage



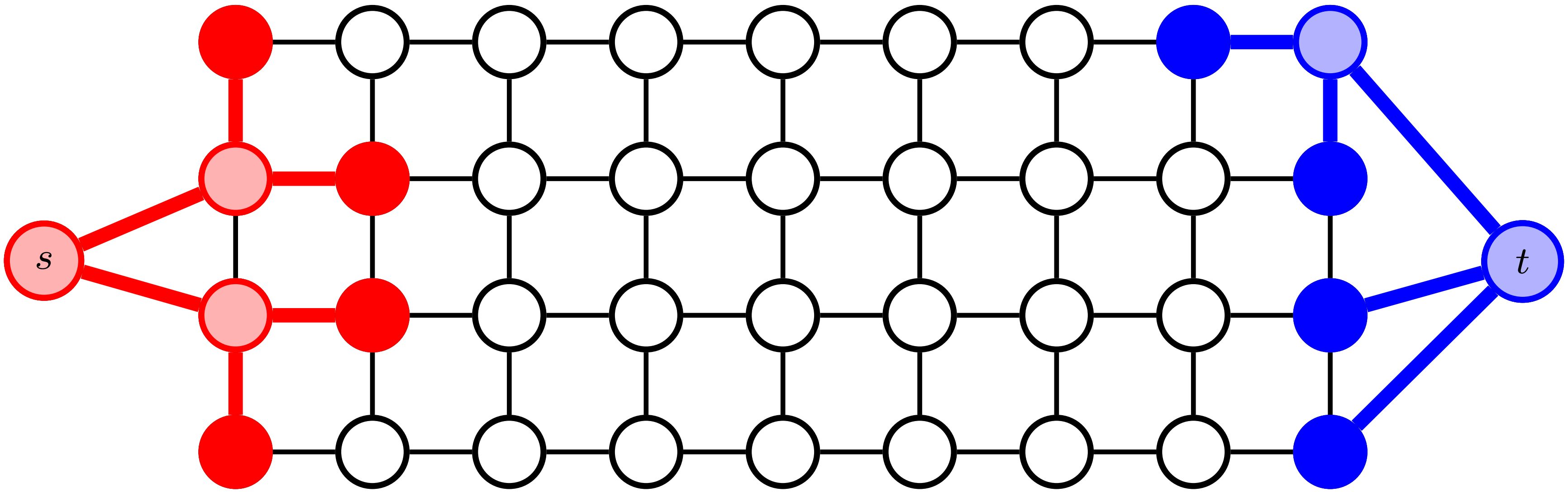
- The *active nodes* explore adjacent edges and acquire new children from a set of *free nodes*
- The newly acquired nodes become *active* members of the corresponding search trees
- The *active node* becomes *passive*, when all of its neighbors are explored
- If an *active node* encounters a neighboring node belonging to the opposite tree, the growth stage terminates

Growth stage



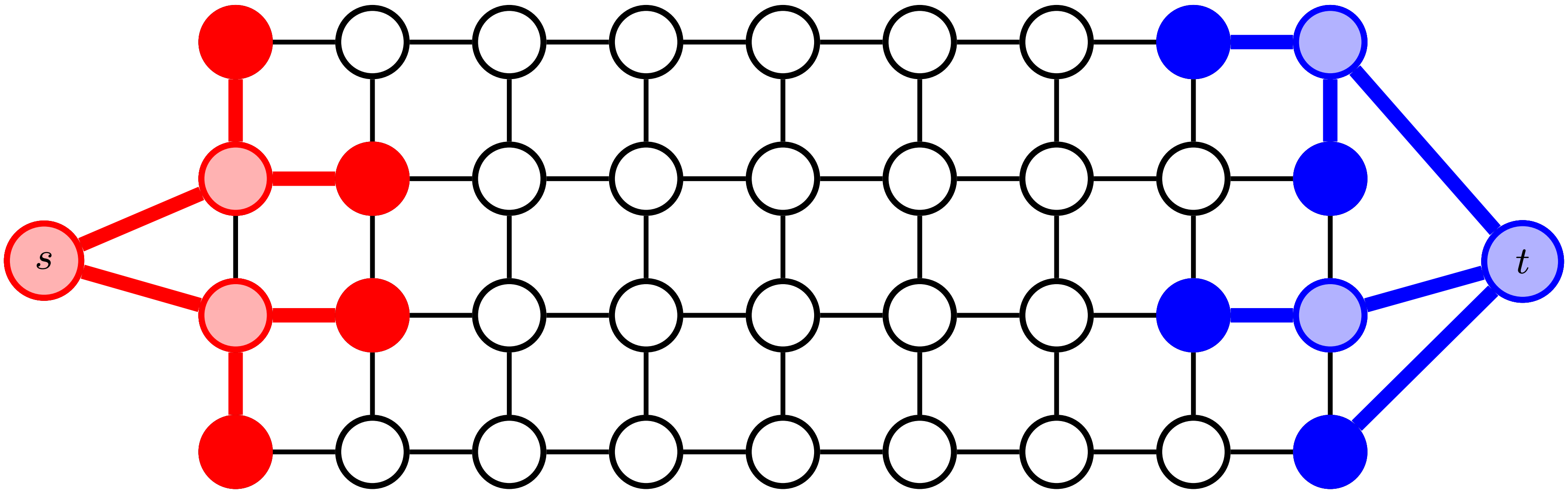
- The *active nodes* explore adjacent edges and acquire new children from a set of *free nodes*
- The newly acquired nodes become *active* members of the corresponding search trees
- The *active node* becomes *passive*, when all of its neighbors are explored
- If an *active node* encounters a neighboring node belonging to the opposite tree, the growth stage terminates

Growth stage



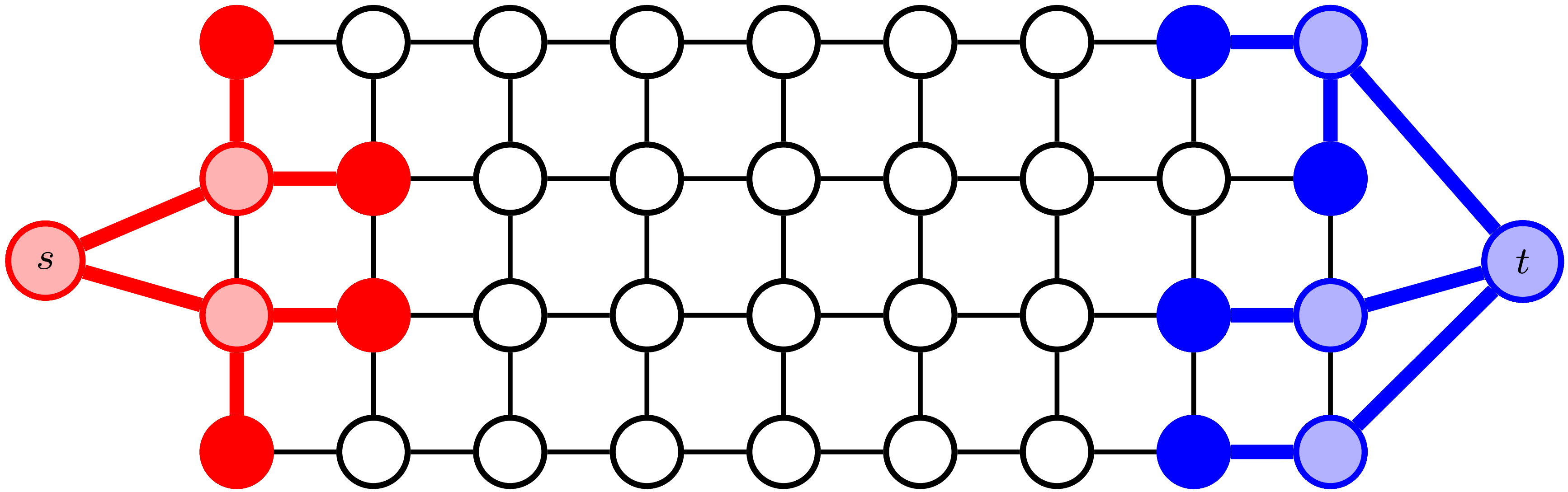
- The *active nodes* explore adjacent edges and acquire new children from a set of *free nodes*
- The newly acquired nodes become *active* members of the corresponding search trees
- The *active node* becomes *passive*, when all of its neighbors are explored
- If an *active node* encounters a neighboring node belonging to the opposite tree, the growth stage terminates

Growth stage



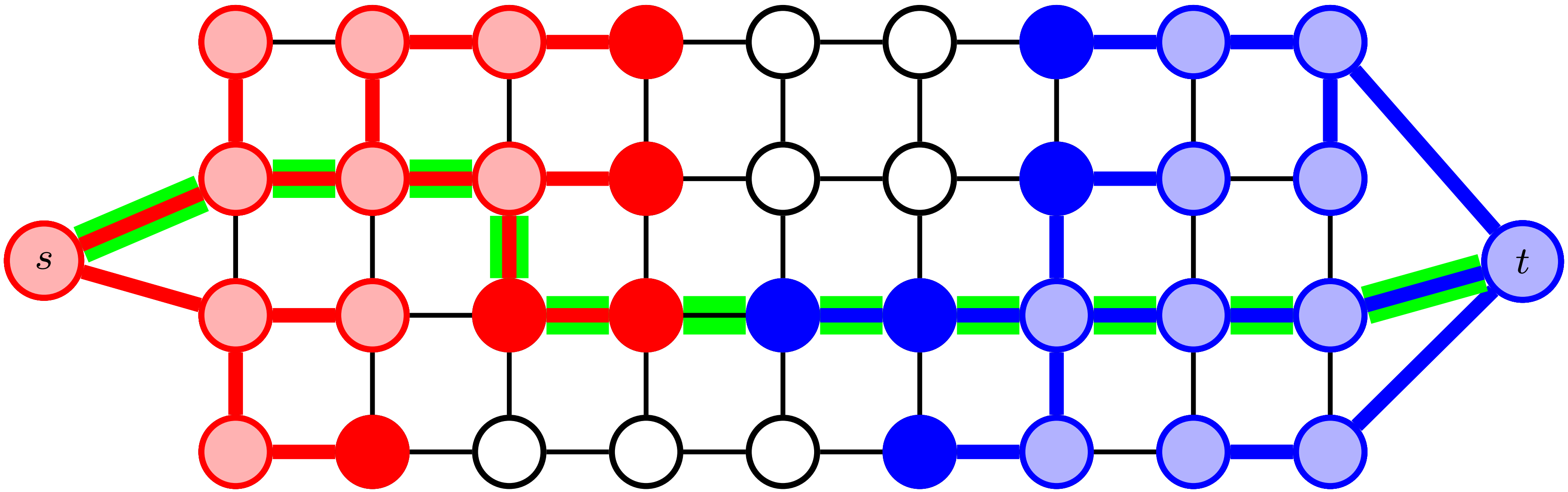
- The *active nodes* explore adjacent edges and acquire new children from a set of *free nodes*
- The newly acquired nodes become *active* members of the corresponding search trees
- The *active node* becomes *passive*, when all of its neighbors are explored
- If an *active node* encounters a neighboring node belonging to the opposite tree, the growth stage terminates

Growth stage



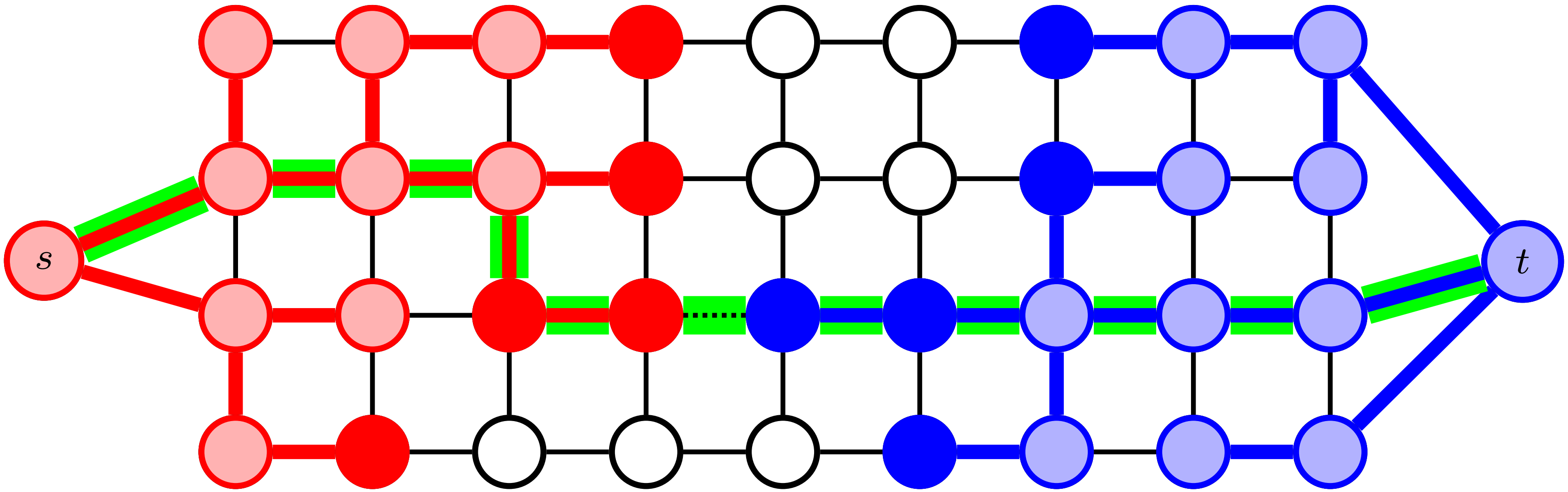
- The *active nodes* explore adjacent edges and acquire new children from a set of *free nodes*
- The newly acquired nodes become *active* members of the corresponding search trees
- The *active node* becomes *passive*, when all of its neighbors are explored
- If an *active node* encounters a neighboring node belonging to the opposite tree, the growth stage terminates

Growth stage



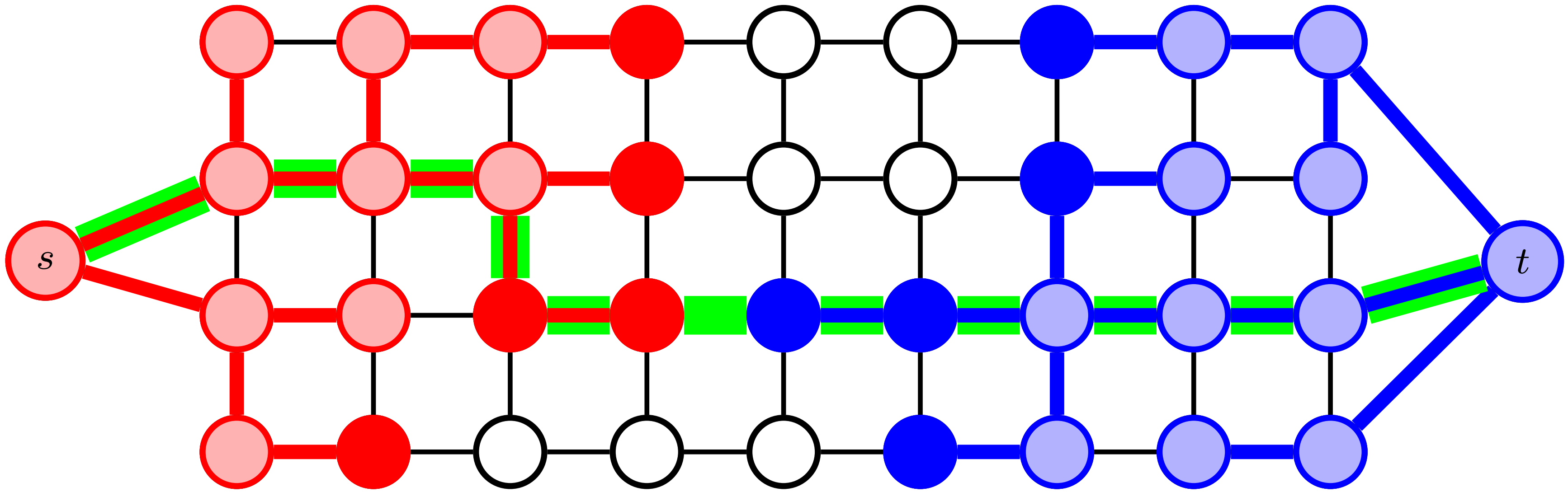
- The *active nodes* explore adjacent edges and acquire new children from a set of *free nodes*
- The newly acquired nodes become *active* members of the corresponding search trees
- The *active node* becomes *passive*, when all of its neighbors are explored
- If an *active node* encounters a neighboring node belonging to the opposite tree, the growth stage terminates

Augmentation stage



- Find the bottleneck capacity Δ on P
- Update the residual graph by pushing flow Δ through P

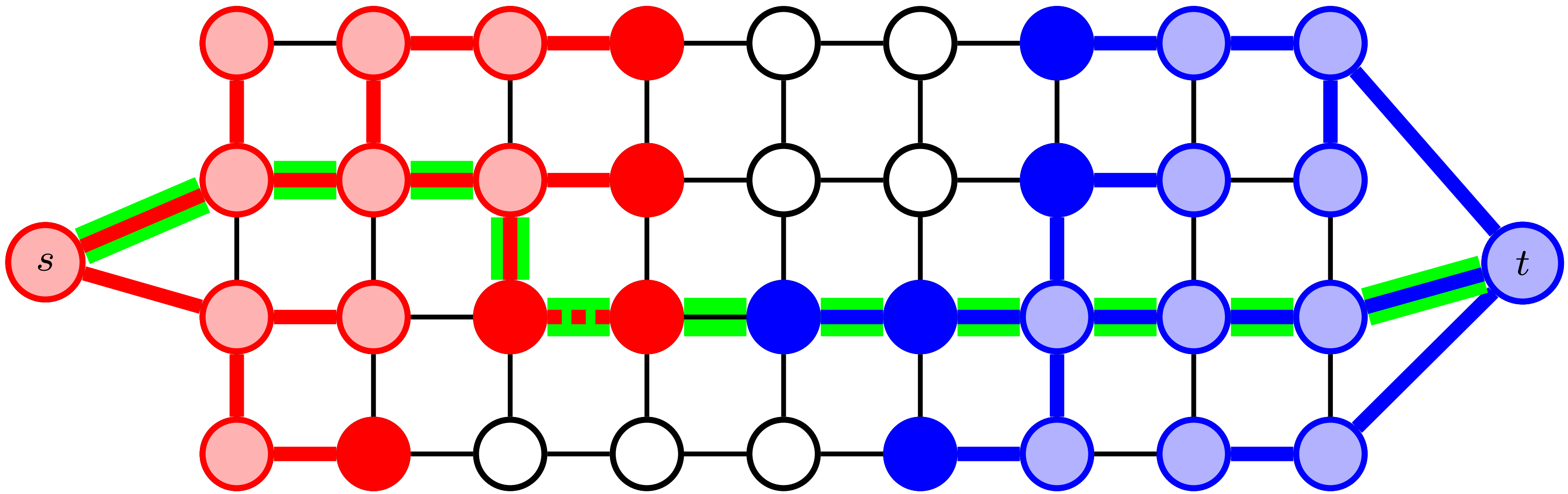
Augmentation stage



- Find the bottleneck capacity Δ on P
- Update the residual graph by pushing flow Δ through P

Adoption stage

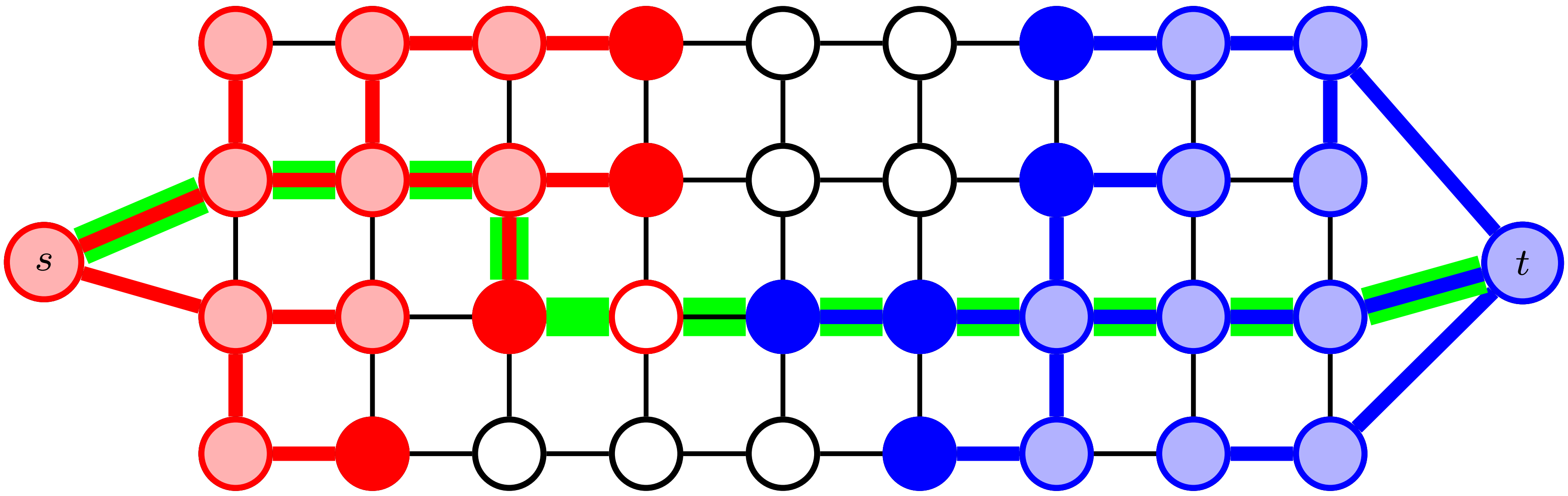
- *Orphan* (○ ○): the nodes such that the edges linking them to their parents are no longer valid (i.e. they are saturated)
- By removing them the search trees S and T may be split into *forests*



We are trying to find a *new valid parent* for p among its neighbors, such that a new parent should belong to the same set, S or T, as the *orphan*

Adoption stage

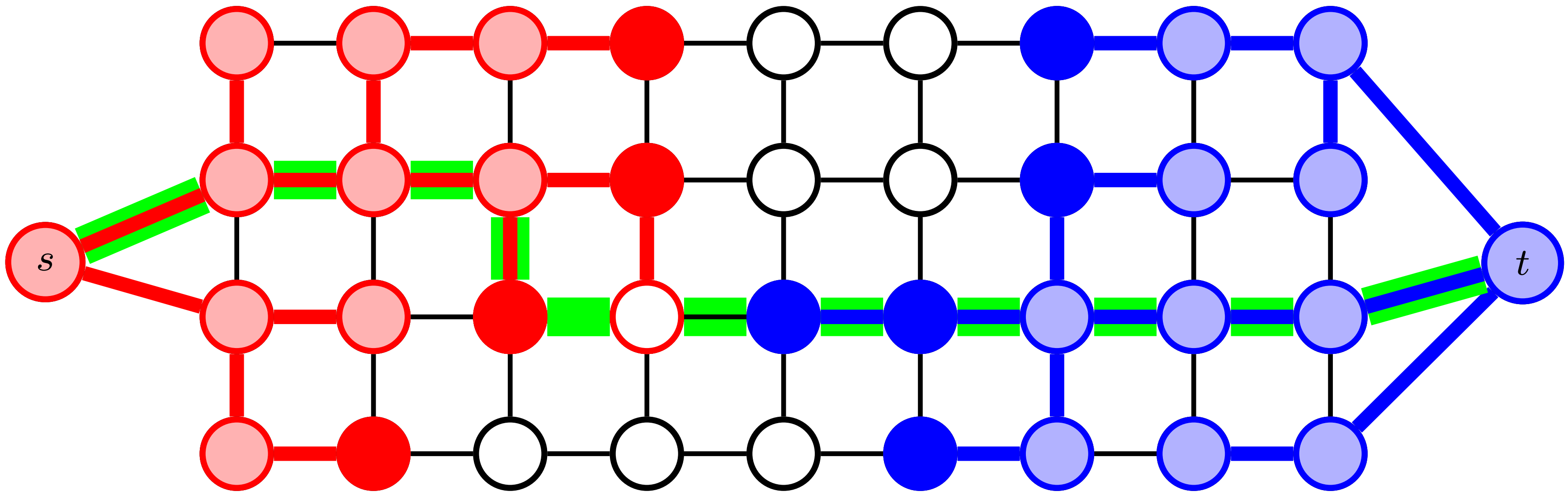
- *Orphan* (○ ○): the nodes such that the edges linking them to their parents are no longer valid (i.e. they are saturated)
- By removing them the search trees S and T may be split into *forests*



We are trying to find a *new valid parent* for p among its neighbors, such that a new parent should belong to the same set, S or T, as the *orphan*

Adoption stage

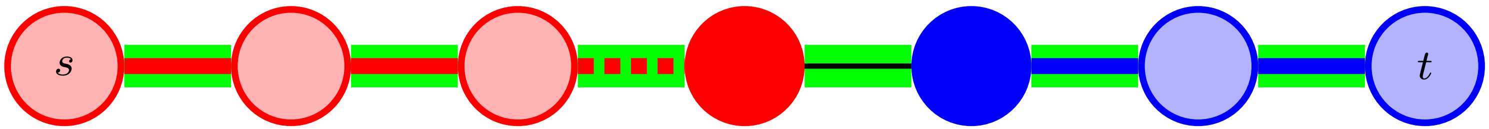
- *Orphan* (○ ○): the nodes such that the edges linking them to their parents are no longer valid (i.e. they are saturated)
- By removing them the search trees S and T may be split into *forests*



We are trying to find a *new valid parent* for p among its neighbors, such that a new parent should belong to the same set, S or T, as the *orphan*

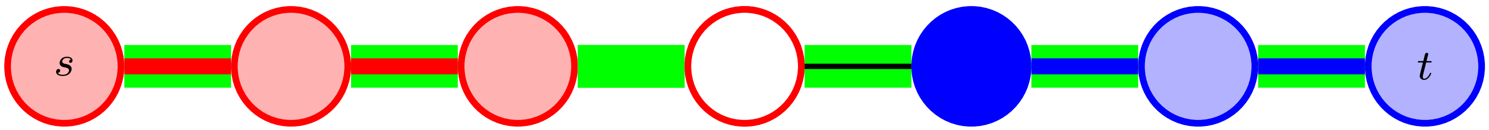
Adoption stage

If an orphan p does not find a valid parent then it becomes a *free node*



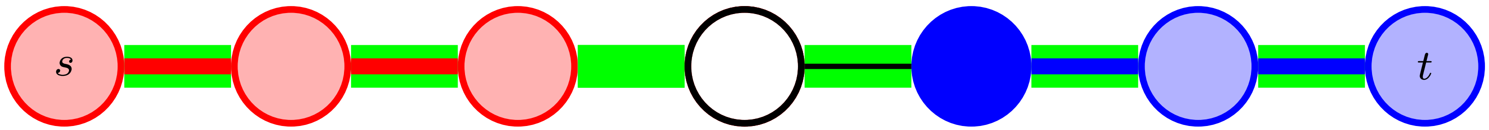
Adoption stage

If an orphan p does not find a valid parent then it becomes a *free node*



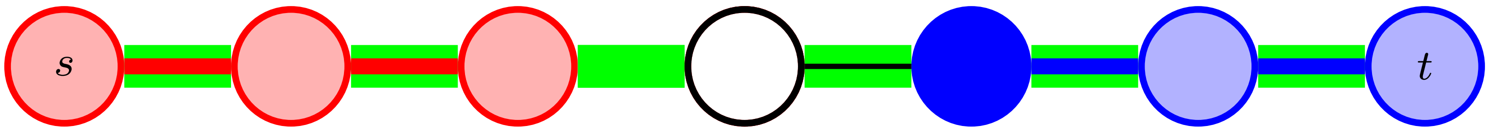
Adoption stage

If an orphan p does not find a valid parent then it becomes a *free node*



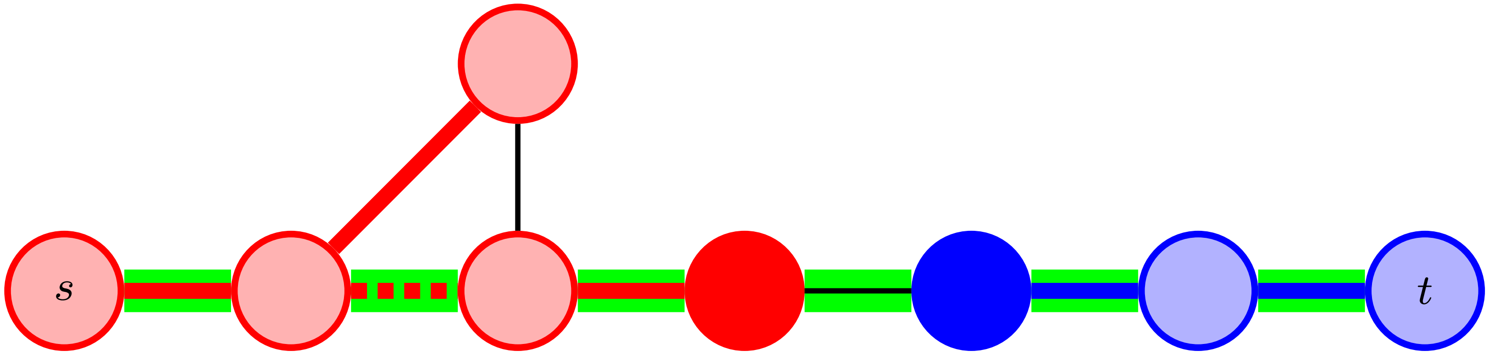
Adoption stage

If an orphan p does not find a valid parent then it becomes a *free node*



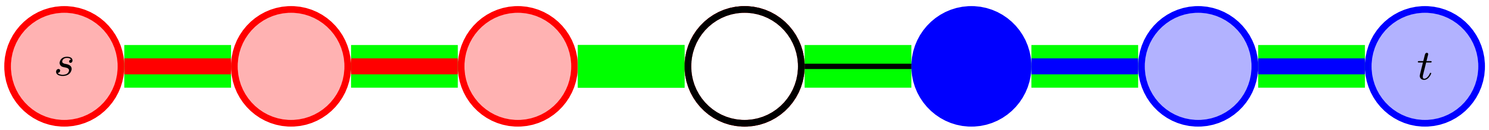
Scan all neighbors q of p such that q belong to the same tree as p:

- if tree $c(q, p) > 0$, add q to the *active set*
- if $\texttt{parent}(q) = p$, add q to the set of *orphans* and set $\texttt{parent}(q) = \varnothing$



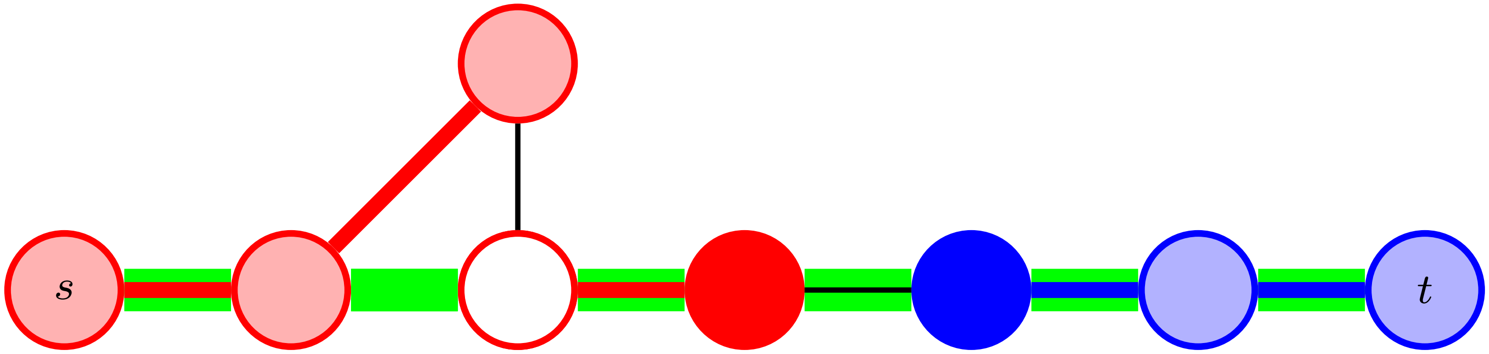
Adoption stage

If an orphan p does not find a valid parent then it becomes a *free node*



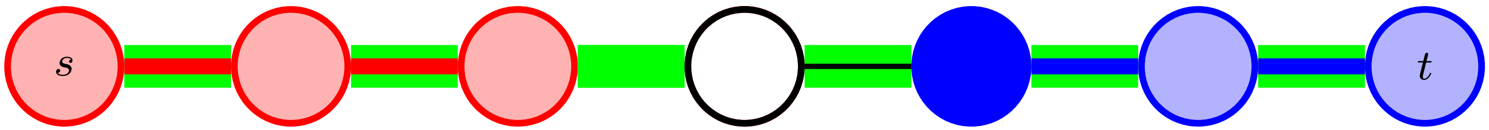
Scan all neighbors q of p such that q belong to the same tree as p:

- if tree $c(q,p) > 0$, add q to the *active set*
- if $\texttt{parent}(q) = p$, add q to the set of *orphans* and set $\texttt{parent}(q) = \emptyset$



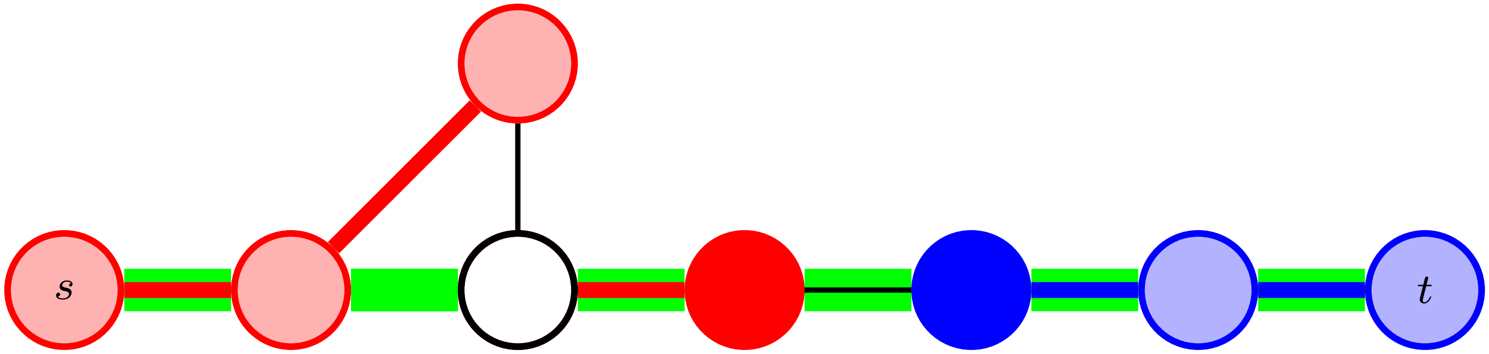
Adoption stage

If an orphan p does not find a valid parent then it becomes a *free node*



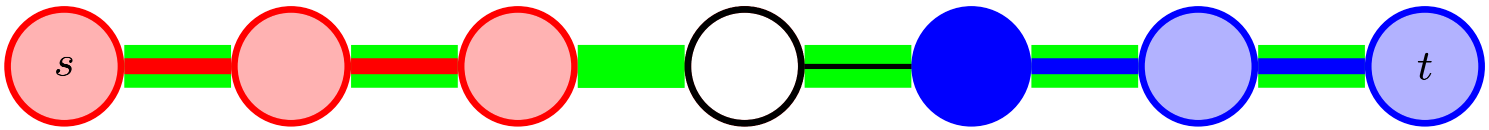
Scan all neighbors q of p such that q belong to the same tree as p:

- if tree $c(q,p) > 0$, add q to the *active set*
- if $\texttt{parent}(q) = p$, add q to the set of *orphans* and set $\texttt{parent}(q) = \emptyset$



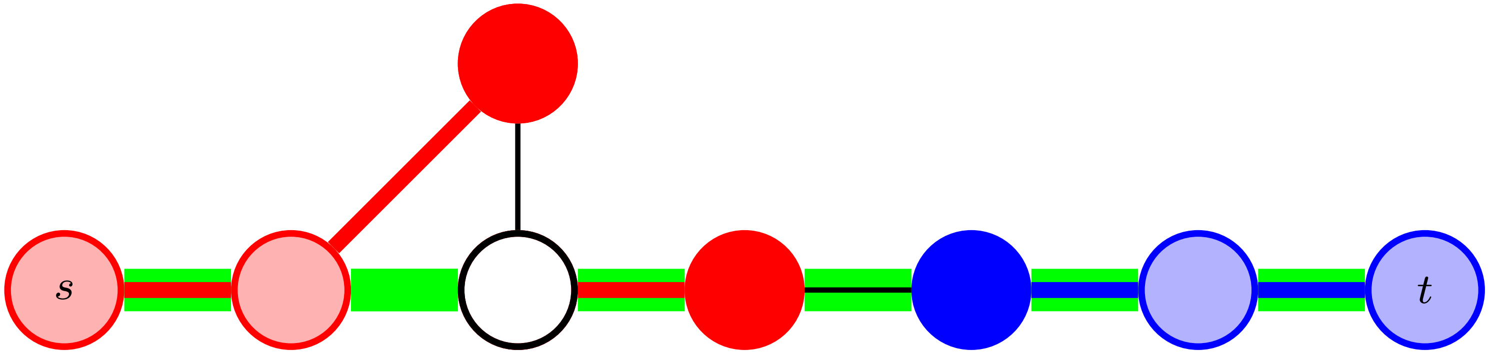
Adoption stage

If an orphan p does not find a valid parent then it becomes a *free node*



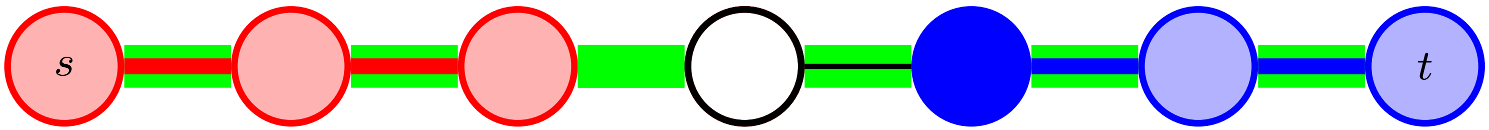
Scan all neighbors q of p such that q belong to the same tree as p:

- if tree $c(q,p) > 0$, add q to the *active set*
- if $\texttt{parent}(q) = p$, add q to the set of *orphans* and set $\texttt{parent}(q) = \varnothing$



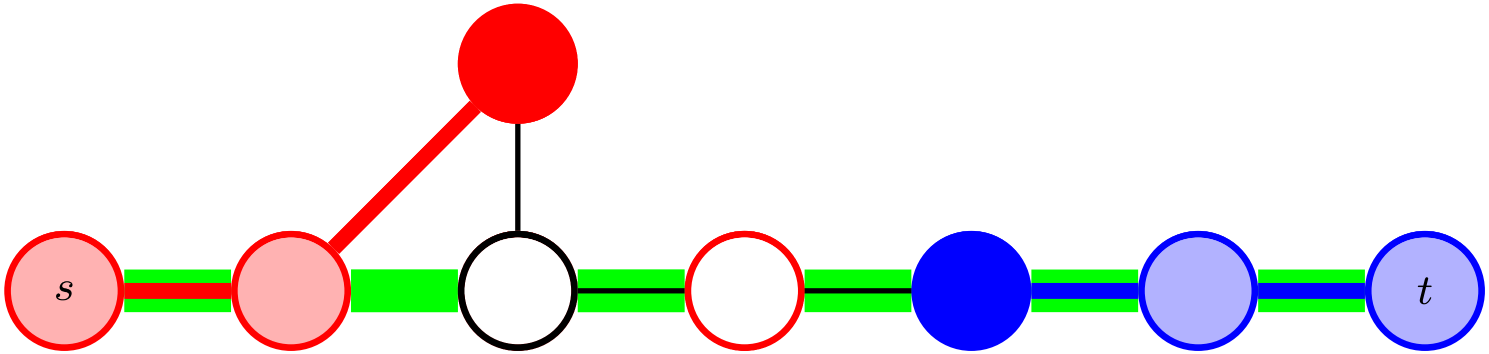
Adoption stage

If an orphan p does not find a valid parent then it becomes a *free node*



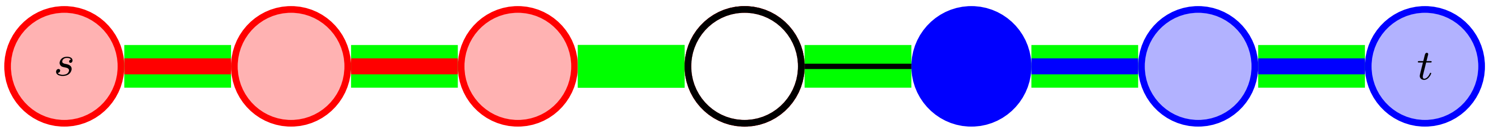
Scan all neighbors q of p such that q belong to the same tree as p:

- if tree $c(q, p) > 0$, add q to the *active set*
- if $\texttt{parent}(q) = p$, add q to the set of *orphans* and set $\texttt{parent}(q) = \emptyset$



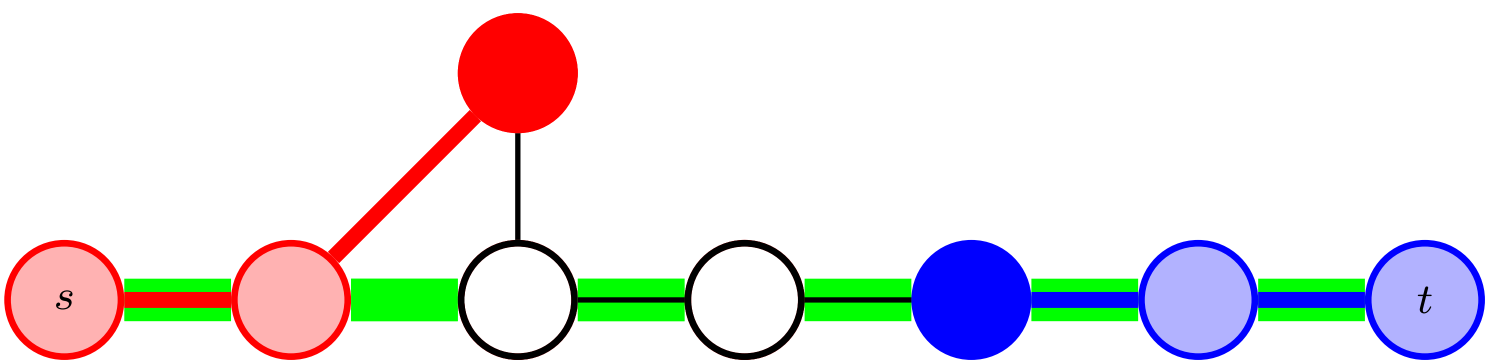
Adoption stage

If an orphan p does not find a valid parent then it becomes a *free node*



Scan all neighbors q of p such that q belong to the same tree as p:

- if tree $c(q,p) > 0$, add q to the *active set*
- if $\texttt{parent}(q) = p$, add q to the set of *orphans* and set $\texttt{parent}(q) = \emptyset$



Summary

- `Boykov-Kolmogorov algorithm` is an *augmented path-based method* with worst case complexity $\mathcal{O}(|\mathcal{E}| \cdot |\mathcal{V}|^2 \cdot |C|)$, where $|C|$ is the value of the minimum cut
- This complexity is worse than complexities of the standard algorithm, however, this algorithm significantly ($\sim$2-10$\times$) outperforms standard algorithms on typical problem instances in vision

Summary

- `Boykov-Kolmogorov algorithm` is an *augmented path-based method* with worst case complexity $\mathcal{O}(|\mathcal{E}| \cdot |\mathcal{V}|^2 \cdot |C|)$, where $|C|$ is the value of the minimum cut
- This complexity is worse than complexities of the standard algorithm, however, this algorithm significantly ($\sim$2-10$\times$) outperforms standard algorithms on typical problem instances in vision

- In many computer vision problems we aim to minimize an em energy function

$$E(\mathbf{y}) = \sum_{i \in \mathcal{V}} E_i(y_i) + \sum_{(i,j \in \mathcal{E})} E_{ij}(y_i, y_j)$$

- As we will see, this is often achieved by solving the *maxFlow problem*