

# Discrete Optimization

Quentin Louveaux

ULg - Institut Montefiore

2016

# Polynomial running time algorithms for the max flow problem

- The generic augmenting path algorithm runs in  $\mathcal{O}(mnU)$
- Ways to improve the algorithm and make it run in polynomial time
  - ▶ Augmenting in **large** increments of flows  
→ **Capacity scaling** algorithm
  - ▶ Augment on **shortest paths** in the residual network
  - ▶ Relax the **mass balance constraints** and only augment **locally**  
→ The **push-relabel** algorithm which is the most efficient one!

In order to implement the last two algorithms, we need to rely on **distance labels**.

# Polynomial running time algorithms for the max flow problem

- The generic augmenting path algorithm runs in  $\mathcal{O}(mnU)$
- Ways to improve the algorithm and make it run in polynomial time
  - ▶ Augmenting in **large** increments of flows  
→ **Capacity scaling** algorithm
  - ▶ Augment on **shortest paths** in the residual network
  - ▶ Relax the **mass balance constraints** and only augment **locally**  
→ The **push-relabel** algorithm which is the most efficient one!

In order to implement the last two algorithms, we need to rely on **distance labels**.

# Distance labels

## Definition

A distance function gives a **numeric label** to each node.

The distance function is **valid** if

$$d(t) = 0$$

$$d(i) \leq d(j) + 1 \quad \text{for every arc } (i,j) \text{ in } G(x)$$

## Property

If  $d$  is valid,  $d(i)$  is a **lower bound** on the length of the shortest path from  $i$  to  $t$  in the **residual graph**.

## Corollary

If  $d(s) \geq n$ , there exists no path from  $s$  to  $t$  in the residual graph.

# Distance labels

## Definition

A distance function gives a **numeric label** to each node.

The distance function is **valid** if

$$d(t) = 0$$

$$d(i) \leq d(j) + 1 \quad \text{for every arc } (i,j) \text{ in } G(x)$$

## Property

If  $d$  is valid,  $d(i)$  is a **lower bound** on the length of the shortest path from  $i$  to  $t$  in the **residual graph**.

## Corollary

If  $d(s) \geq n$ , there exists no path from  $s$  to  $t$  in the residual graph.

# Distance labels

## Definition

A distance function gives a **numeric label** to each node.

The distance function is **valid** if

$$d(t) = 0$$

$$d(i) \leq d(j) + 1 \quad \text{for every arc } (i,j) \text{ in } G(x)$$

## Property

If  $d$  is valid,  $d(i)$  is a **lower bound** on the length of the shortest path from  $i$  to  $t$  in the **residual graph**.

## Corollary

If  $d(s) \geq n$ , there exists no path from  $s$  to  $t$  in the residual graph.

## Distance labels

The distance labels are **exact** if each label indicates the **exact length** of the shortest path to  $t$  in the residual graph.

An exact labeling can be determined in  $\mathcal{O}(m)$  by **backward breadth-first search**.

An arc  $(i, j) \in G(x)$  is **admissible** if

$$d(i) = d(j) + 1,$$

otherwise it is **inadmissible**.

An **admissible path** is a path consisting only of admissible arcs.

An admissible path is a **shortest augmenting path** !

## Distance labels

The distance labels are **exact** if each label indicates the **exact length** of the shortest path to  $t$  in the residual graph.

An exact labeling can be determined in  $\mathcal{O}(m)$  by **backward breadth-first search**.

An arc  $(i, j) \in G(x)$  is **admissible** if

$$d(i) = d(j) + 1,$$

otherwise it is **inadmissible**.

An **admissible path** is a path consisting only of admissible arcs.

An admissible path is a **shortest augmenting path** !



## Distance labels

The distance labels are **exact** if each label indicates the **exact length** of the shortest path to  $t$  in the residual graph.

An exact labeling can be determined in  $\mathcal{O}(m)$  by **backward breadth-first search**.

An arc  $(i, j) \in G(x)$  is **admissible** if

$$d(i) = d(j) + 1,$$

otherwise it is **inadmissible**.

An **admissible path** is a path consisting only of admissible arcs.

An admissible path is a **shortest augmenting path** !

# The shortest augmenting path algorithm

- Augmenting on shortest paths in the residual network guarantees a **polynomial running time**!
- We could rerun backward breadth-first search at **each iteration** → very inefficient but runs in  $\mathcal{O}(nm^2)$
- The minimum distance from each node to the sink is **monotonically increasing**.

# The shortest augmenting path algorithm

- Perform an **initial labeling** by backward breadth-first-search
- Repeat : Perform an **advance operation** (finding an admissible arc from the current last node in the admissible path)
- If we find an augmenting path, then we augment along it !
- If at some node  $i$ , we do not find any admissible arc, we **relabel** the node

$$\min\{d(j) + 1 \mid (i, j) \in \delta(\{i\}) \text{ and } r_{ij} > 0\}$$

and remove the node  $i$  from the current admissible path.

# The shortest augmenting path algorithm

- Perform an **initial labeling** by backward breadth-first-search
- Repeat : Perform an **advance operation** (finding an admissible arc from the current last node in the admissible path)
- If we find an augmenting path, then we augment along it !
- If at some node  $i$ , we do not find any admissible arc, we **relabel** the node

$$\min\{d(j) + 1 \mid (i, j) \in \delta(\{i\}) \text{ and } r_{ij} > 0\}$$

and remove the node  $i$  from the current admissible path.

# The shortest augmenting path algorithm

- Perform an **initial labeling** by backward breadth-first-search
- Repeat : Perform an **advance operation** (finding an admissible arc from the current last node in the admissible path)
- If we find an augmenting path, then we augment along it !
- If at some node  $i$ , we do not find any admissible arc, we **relabel** the node

$$\min\{d(j) + 1 \mid (i, j) \in \delta(\{i\}) \text{ and } r_{ij} > 0\}$$

and remove the node  $i$  from the current admissible path.

# The shortest augmenting path algorithm

- Perform an **initial labeling** by backward breadth-first-search
- Repeat : Perform an **advance operation** (finding an admissible arc from the current last node in the admissible path)
- If we find an augmenting path, then we augment along it !
- If at some node  $i$ , we do not find any admissible arc, we **relabel** the node

$$\min\{d(j) + 1 \mid (i, j) \in \delta(\{i\}) \text{ and } r_{ij} > 0\}$$

and remove the node  $i$  from the current admissible path.

# Correctness of the algorithm

- Every operation maintains a **valid distance labeling**.
- Each relabel strictly increases the distance label of a node.
- The shortest augmenting path algorithm correctly computes a maximum flow.

# Correctness of the algorithm

- Every operation maintains a **valid distance labeling**.
- Each relabel strictly increases the distance label of a node.
- The shortest augmenting path algorithm correctly computes a maximum flow.



# Correctness of the algorithm

- Every operation maintains a **valid distance labeling**.
- Each relabel strictly increases the distance label of a node.
- The shortest augmenting path algorithm correctly computes a maximum flow.

# Complexity of the algorithm

- Between two **relabels**, if an arc becomes inadmissible, it stays inadmissible.
- If the algorithm relabels any node at most  $k$  times, the complexity of finding admissible arcs and relabeling the nodes is  $\mathcal{O}(km)$  and the algorithm saturates arcs at most  $\frac{km}{2}$  times.
- Each distance label increases at most  $n$  times. The total number of relabeling is  $n^2$  and the total number of augment operations is  $\frac{nm}{2}$ .

The shortest augmenting path algorithm runs in  $\mathcal{O}(n^2m)$ .

# Complexity of the algorithm

- Between two **relabels**, if an arc becomes inadmissible, it stays inadmissible.
- If the algorithm relabels any node at most  $k$  times, the complexity of finding admissible arcs and relabeling the nodes is  $\mathcal{O}(km)$  and the algorithm saturates arcs at most  $\frac{km}{2}$  times.
- Each distance label increases at most  $n$  times. The total number of relabeling is  $n^2$  and the total number of augment operations is  $\frac{nm}{2}$ .

The shortest augmenting path algorithm runs in  $\mathcal{O}(n^2m)$ .

# Complexity of the algorithm

- Between two **relabels**, if an arc becomes inadmissible, it stays inadmissible.
- If the algorithm relabels any node at most  $k$  times, the complexity of finding admissible arcs and relabeling the nodes is  $\mathcal{O}(km)$  and the algorithm saturates arcs at most  $\frac{km}{2}$  times.
- Each distance label increases at most  $n$  times. The total number of relabeling is  $n^2$  and the total number of augment operations is  $\frac{nm}{2}$ .

The shortest augmenting path algorithm runs in  $\mathcal{O}(n^2m)$ .

# Complexity of the algorithm

- Between two **relabels**, if an arc becomes inadmissible, it stays inadmissible.
- If the algorithm relabels any node at most  $k$  times, the complexity of finding admissible arcs and relabeling the nodes is  $\mathcal{O}(km)$  and the algorithm saturates arcs at most  $\frac{km}{2}$  times.
- Each distance label increases at most  $n$  times. The total number of relabeling is  $n^2$  and the total number of augment operations is  $\frac{nm}{2}$ .

The shortest augmenting path algorithm runs in  $\mathcal{O}(n^2m)$ .

# Preflow-push algorithm

Drawback of the augmenting path algorithm : the expensive operation of **sending flow** along a path.

Many of these augmentations may share **the same subpath** !

Idea of preflow-push : augment along **arcs** and we therefore have to **relax the flow balance constraints**.

# Preflows

## Definition

A **preflow** is a function  $x : A \rightarrow \mathbb{R}_+$  such that

$$\sum_{j|(j,i) \in A} x_{ji} - \sum_{j|(i,j) \in A} x_{ij} \geq 0. \quad \text{for all } i \in V \setminus \{s, t\}.$$

## Definition

The **excess** at a node  $i$  of a given preflow  $x$  is given by

$$e(i) := \sum_{j|(j,i) \in A} x_{ji} - \sum_{j|(i,j) \in A} x_{ij}.$$

A node with positive excess is said to be **active** because we need to recover the mass-balance constraint at some point.

## Definition

A **preflow** is a function  $x : A \rightarrow \mathbb{R}_+$  such that

$$\sum_{j|(j,i) \in A} x_{ji} - \sum_{j|(i,j) \in A} x_{ij} \geq 0. \quad \text{for all } i \in V \setminus \{s, t\}.$$

## Definition

The **excess** at a node  $i$  of a given preflow  $x$  is given by

$$e(i) := \sum_{j|(j,i) \in A} x_{ji} - \sum_{j|(i,j) \in A} x_{ij}.$$

A node with positive excess is said to be **active** because we need to recover the mass-balance constraint at some point.



## Definition

A **preflow** is a function  $x : A \rightarrow \mathbb{R}_+$  such that

$$\sum_{j|(j,i) \in A} x_{ji} - \sum_{j|(i,j) \in A} x_{ij} \geq 0. \quad \text{for all } i \in V \setminus \{s, t\}.$$

## Definition

The **excess** at a node  $i$  of a given preflow  $x$  is given by

$$e(i) := \sum_{j|(j,i) \in A} x_{ji} - \sum_{j|(i,j) \in A} x_{ij}.$$

A node with positive excess is said to be **active** because we need to recover the mass-balance constraint at some point.

# Preflow-push algorithm

- Compute exact **distance labels**  $d(i)$
- Push flows along all arcs emanating from  $s$  :  $x_{sj} := u_{sj}$
- $d(s) := n$
- If a node is active, **push-relabel**
  - ▶ If  $(i, j)$  is admissible, push  $\delta := \min\{e(i), r_{ij}\}$
  - ▶ Else  $d(i) := \min\{d(j) + 1 \mid (i, j) \in \delta(\{i\}) \text{ and } r_{ij} > 0\}$

# Preflow-push algorithm

- Compute exact **distance labels**  $d(i)$
- Push flows along all arcs emanating from  $s : x_{sj} := u_{sj}$
- $d(s) := n$
- If a node is active, **push-relabel**
  - ▶ If  $(i, j)$  is admissible, push  $\delta := \min\{e(i), r_{ij}\}$
  - ▶ Else  $d(i) := \min\{d(j) + 1 \mid (i, j) \in \delta(\{i\}) \text{ and } r_{ij} > 0\}$

# Preflow-push algorithm

- Compute exact **distance labels**  $d(i)$
- Push flows along all arcs emanating from  $s$  :  $x_{sj} := u_{sj}$
- $d(s) := n$
- If a node is active, **push-relabel**
  - ▶ If  $(i, j)$  is admissible, push  $\delta := \min\{e(i), r_{ij}\}$
  - ▶ Else  $d(i) := \min\{d(j) + 1 \mid (i, j) \in \delta(\{i\}) \text{ and } r_{ij} > 0\}$

# Preflow-push algorithm

- Compute exact **distance labels**  $d(i)$
- Push flows along all arcs emanating from  $s : x_{sj} := u_{sj}$
- $d(s) := n$
- If a node is active, **push-relabel**
  - ▶ If  $(i, j)$  is admissible, push  $\delta := \min\{e(i), r_{ij}\}$
  - ▶ Else  $d(i) := \min\{d(j) + 1 \mid (i, j) \in \delta(\{i\}) \text{ and } r_{ij} > 0\}$

# Preflow-push algorithm

- Compute exact **distance labels**  $d(i)$
- Push flows along all arcs emanating from  $s$  :  $x_{sj} := u_{sj}$
- $d(s) := n$
- If a node is active, **push-relabel**
  - ▶ If  $(i, j)$  is admissible, push  $\delta := \min\{e(i), r_{ij}\}$
  - ▶ Else  $d(i) := \min\{d(j) + 1 \mid (i, j) \in \delta(\{i\}) \text{ and } r_{ij} > 0\}$

# Preflow-push algorithm

- Compute exact **distance labels**  $d(i)$
- Push flows along all arcs emanating from  $s$  :  $x_{sj} := u_{sj}$
- $d(s) := n$
- If a node is active, **push-relabel**
  - ▶ If  $(i, j)$  is admissible, push  $\delta := \min\{e(i), r_{ij}\}$
  - ▶ Else  $d(i) := \min\{d(j) + 1 \mid (i, j) \in \delta(\{i\}) \text{ and } r_{ij} > 0\}$

# Complexity of the algorithm

- Distance labels are always **valid** during the algorithm
- At any stage, a node  $i$  with excess  $e(i) > 0$  has a path from  $i$  to  $s$  in the residual network
- For each  $i$ ,  $d(i) < 2n$
- The total number of **relabels** is  $2n^2$
- The algorithm performs at most  $nm$  **saturating** pushes.
- The algorithm performs at most  $n^2m$  **nonsaturating** pushes
- The algorithm runs in  $\mathcal{O}(n^2m)$



# Complexity of the algorithm

- Distance labels are always **valid** during the algorithm
- At any stage, a node  $i$  with excess  $e(i) > 0$  has a path from  $i$  to  $s$  in the residual network
- For each  $i$ ,  $d(i) < 2n$
- The total number of **relabels** is  $2n^2$
- The algorithm performs at most  $nm$  **saturating** pushes.
- The algorithm performs at most  $n^2m$  **nonsaturating** pushes
- The algorithm runs in  $\mathcal{O}(n^2m)$

# Complexity of the algorithm

- Distance labels are always **valid** during the algorithm
- At any stage, a node  $i$  with excess  $e(i) > 0$  has a path from  $i$  to  $s$  in the residual network
- For each  $i$ ,  $d(i) < 2n$
- The total number of **relabels** is  $2n^2$
- The algorithm performs at most  $nm$  **saturating** pushes.
- The algorithm performs at most  $n^2m$  **nonsaturating** pushes
- The algorithm runs in  $\mathcal{O}(n^2m)$

# Complexity of the algorithm

- Distance labels are always **valid** during the algorithm
- At any stage, a node  $i$  with excess  $e(i) > 0$  has a path from  $i$  to  $s$  in the residual network
- For each  $i$ ,  $d(i) < 2n$
- The total number of **relabels** is  $2n^2$
- The algorithm performs at most  $nm$  **saturating** pushes.
- The algorithm performs at most  $n^2m$  **nonsaturating** pushes
- The algorithm runs in  $\mathcal{O}(n^2m)$

# Complexity of the algorithm

- Distance labels are always **valid** during the algorithm
- At any stage, a node  $i$  with excess  $e(i) > 0$  has a path from  $i$  to  $s$  in the residual network
- For each  $i$ ,  $d(i) < 2n$
- The total number of **relabels** is  $2n^2$
- The algorithm performs at most  $nm$  **saturating** pushes.
- The algorithm performs at most  $n^2m$  **nonsaturating** pushes
- The algorithm runs in  $\mathcal{O}(n^2m)$

# Complexity of the algorithm

- Distance labels are always **valid** during the algorithm
- At any stage, a node  $i$  with excess  $e(i) > 0$  has a path from  $i$  to  $s$  in the residual network
- For each  $i$ ,  $d(i) < 2n$
- The total number of **relabels** is  $2n^2$
- The algorithm performs at most  $nm$  **saturating** pushes.
- The algorithm performs at most  $n^2m$  **nonsaturating** pushes
- The algorithm runs in  $\mathcal{O}(n^2m)$

# Complexity of the algorithm

- Distance labels are always **valid** during the algorithm
- At any stage, a node  $i$  with excess  $e(i) > 0$  has a path from  $i$  to  $s$  in the residual network
- For each  $i$ ,  $d(i) < 2n$
- The total number of **relabels** is  $2n^2$
- The algorithm performs at most  $nm$  **saturating** pushes.
- The algorithm performs at most  $n^2m$  **nonsaturating** pushes
- The algorithm runs in  $\mathcal{O}(n^2m)$