

# Optimization I

## Network Optimization

Dr. David Willems

Mathematical Institute  
University of Koblenz-Landau  
Campus Koblenz

Notes

---

---

---

---

---

---

---

### Outline

- 1. Introduction: Definitions and Examples
- 2. Minimum Spanning Tree Problem
- 3. Shortest Path Problems
- 4. Network flow problems
- 5. Dynamic network flow problems

Notes

---

---

---

---

---

---

---

### Introduction: Definitions and Examples

#### Outline

- 1. Introduction: Definitions and Examples
  - 1.1 Graphs
  - 1.2 Path
  - 1.3 Connectedness, Cuts and Trees

Notes

---

---

---

---

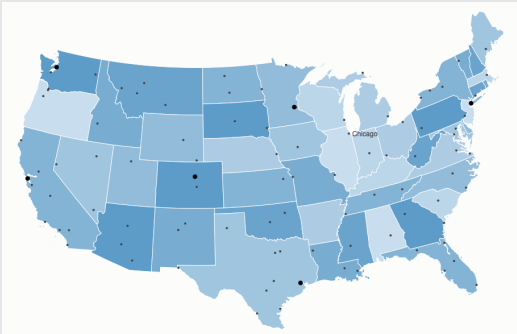
---

---

---

### Traveling salesman problem

#### Example 1.1



Notes

---

---

---

---

---

---

---

Example 1.2



Notes

---

---

---

---

---

---

---

Definition 1.3 (Undirected graph)

An undirected graph is a triple  $G = (V, E, \gamma)$  with a non-empty set of nodes (or vertices)  $V$ , a set of edges (or arcs)  $E$  with  $V \cap E = \emptyset$  and a map

$$\gamma: E \rightarrow \{X: X \subseteq V \text{ with } 1 \leq |X| \leq 2\},$$

that assigns every edge its corresponding endpoints  $\gamma(e) \in V$ .

Example 1.4

→ board

Notes

---

---

---

---

---

---

---

Definition 1.5 (Directed graph)

A directed graph (or short *digraph*) is a quadruple  $G = (V, R, \alpha, \omega)$  with the following properties:

1.  $V$  is a non-empty set of nodes (or vertices).
2.  $R$  is the set of arcs.
3. It holds that  $V \cap R = \emptyset$ .
4.  $\alpha: R \rightarrow V$  and  $\omega: R \rightarrow V$  are maps.  $\alpha(e)$  denotes the head of arc  $e$ ,  $\omega(e)$  is the tail of  $e$ .

Example 1.6

→ board

Notes

---

---

---

---

---

---

---

Definition 1.7 (Loops, parallel edges, simple graphs)

Let  $G = (V, R, \alpha, \omega)$  be a digraph.

- An arc  $r \in R$  is called loop if  $\alpha(r) = \omega(r)$ .
- A graph  $G$  is called loopless if it contains no loops.
- Two arcs  $r_1, r_2 \in R$  are called parallel, if  $\alpha(r_1) = \alpha(r_2)$  and  $\omega(r_1) = \omega(r_2)$ .
- A graph  $G$  is called simple if it contains neither loops nor parallel arcs.

Example 1.8

→ board

Remark 1.9

In the following, we usually assume that  $G$  is simple. In this cases, every arc  $r \in R$  is uniquely identified by the pair  $(\alpha(r), \omega(r))$ .

Notes

---

---

---

---

---

---

---

Definition 1.10

Let  $G = (V, R, \alpha, \omega)$  be a digraph.

- A node  $v \in V$  is called *incident* to an arc  $r \in R$  if  $v \in \{\alpha(r), \omega(r)\}$ .
- Two nodes  $u, v \in V$  are called *adjacent* iff there exists an arc  $r \in R$  that is incident to  $u$  and  $v$ .
- For a node  $v \in V$  we call
  - $\delta_G^+(v) := \{r \in R: \alpha(r) = v\}$  the outgoing arcs of  $v$ .
  - $\delta_G^-(v) := \{r \in R: \omega(r) = v\}$  the ingoing arcs on  $v$ .
  - $N_G^+(v) := \{\omega(r): r \in \delta_G^+(v)\}$  the successor set of  $v$ .
  - $N_G^-(v) := \{\alpha(r): r \in \delta_G^-(v)\}$  the predecessor set of  $v$ .
  - $g_G^+(v) := |\delta_G^+(v)|$  the out-degree of  $v$ .
  - $g_G^-(v) := |\delta_G^-(v)|$  the in-degree of  $v$ .
  - $g_G(v) := g_G^+(v) + g_G^-(v)$  the degree of  $v$ .

Remark 1.11

Usually we drop the index  $G$  if  $G$  is clear from the context.

Example 1.12

→ board

Notes

---

---

---

---

---

---

---

---

Complexity of Algorithms

Definition 1.13 (Landau notation)

Let  $M$  be the set of all real-valued functions  $f: \mathbb{N} \rightarrow \mathbb{R}$ . Every function  $g \in M$  then defines three classes of functions as follows:

$$\mathcal{O}(g) := \{f \in M: \exists c \in \mathbb{R}, n_0 \in \mathbb{N}: \forall n \geq n_0: f(n) \leq c \cdot g(n)\}$$
$$\Omega(g) := \{f \in M: \exists c \in \mathbb{R}, n_0 \in \mathbb{N}: \forall n \geq n_0: f(n) \geq c \cdot g(n)\}$$
$$\Theta(g) := \mathcal{O}(g) \cap \Omega(g)$$

We call a function  $f$  of *polynomial magnitude* in  $n$ , if there exists a polynomial  $g$  (e.g.  $g(n) = n^3$ ) such that  $f \in \mathcal{O}(g)$ .

A function  $f$  grows exponentially in  $n$  if  $f(n)$  cannot be bounded by a polynomial function, e.g.  $g(n) = 2^n$ .

Notes

---

---

---

---

---

---

---

---

Complexity of Algorithms

Model of computation

- For this part of the lecture, we're interested in the running times of the algorithms we're going to develop.
- Our model of computation is the *Unit-Cost RAM* (Random Access Machine); a "machine" that has countable many registers.
- The following operations can be performed in one clock cycle:
  - Reading and writing numbers.
  - Comparison of two registers.
  - Conditional branching.
  - Addition, subtraction, multiplication and division.
- The complexity of an algorithm is a function depending on the input size.

Notes

---

---

---

---

---

---

---

---

Complexity of Algorithms

Model of Computation

- Usually, integers are coded as binary strings.  $p \in \mathbb{N}$  thus has the coding length of  $\lceil \log_2(p+1) \rceil + 1$  bits. We need  $\lceil \log_2(p+1) \rceil$  bits to represent  $|p|$  and an additional bit for the sign.
- A fractional number  $p/q$  with  $q \geq 1$  and  $p, q \in \mathbb{Z}$  coprime has thus the coding length  $\lceil \log_2(p+1) \rceil + \lceil \log_2(q+1) \rceil + 1$ .
- We say an algorithm has *worst-case (time) complexity*  $T$ , if the running time for all inputs of size  $\ell$  can be bounded from above by  $T(\ell)$ .

Notes

---

---

---

---

---

---

---

---

Definition 1.14 (Adjacency matrix)

Let  $G = (V, E)$  be a digraph and  $n = |V|$ . Then, the *adjacency matrix*  $A = (a_{ij})_{n \times n} \in \mathbb{R}^{n \times n}$  is a matrix with entries

$$a_{ij} = |\{r \in R: \alpha(r) = v_i \text{ and } \omega(r) = v_j\}|.$$

For undirected graphs  $H$  we analogously define the adjacency matrix as

$$a_{ij} = |\{e \in E: \gamma(e) = \{v_i, v_j\}\}|.$$

Example 1.15

→ board

Observation 1.16

For undirected graphs the adjacency matrix is symmetric.

Notes

Definition 1.17 (Incidence matrix)

Let  $G = (V, R)$  be a loop-less digraph,  $n = |V|$  and  $R = \{r_1, \dots, r_m\}$ . Then, the *incidence matrix*  $I \in \mathbb{R}^{n \times m}$  is a matrix with

$$i_{kl} := \begin{cases} 1 & \text{if } \alpha(r_l) = v_k \\ -1 & \text{if } \omega(r_l) = v_k \\ 0 & \text{otherwise.} \end{cases}$$

For undirected graphs we define

$$i_{kl} := \begin{cases} 1 & \text{if } v_k \in \gamma(e_l) \\ 0 & \text{otherwise.} \end{cases}$$

Example 1.18

→ board

Notes

Remark 1.19

Note that the rows of the incidence matrix  $I$  are linearly dependent for every digraph  $G$ . If the matrix obtained by removing a row from  $I$  has full rank, it is called *full rank incidence matrix*.

Remark 1.20

In the case of an digraph  $G$ , the incidence matrix  $I$  is *unimodular*, that means every quadratic submatrix has determinant  $+1, -1$  or  $0$ .

Observation 1.21

Let  $G = (V, E)$  be a (di)graph with  $|V| = n$  and  $|E| = m$ .

- The space needed to save the adjacency matrix is  $\Theta(n^2)$  and is independent of the number of edges in  $G$ .
- The space needed to save the incidence matrix of  $G$  is  $\Theta(nm)$ .

Notes

Comparison of graph storage techniques

The following table gives the time complexity for three elementary graph operations depending on their storage:

1. Is there an arc from  $v$  to  $w$ ? (short:  $(v, w) \in R$ ?)
2. What is the out-degree of  $v$ ? (short:  $g^+(v) = ?$ )
3. Exists a node with in-degree  $g^-(v) = 0$ ? (short:  $\exists v: g^-(v) = 0$ ?)

Storage	Memory	$(v, w) \in R$ ?	$g^+(v) = ?$	$\exists v: g^-(v) = 0$ ?
Adjacency matrix	$\Theta(n^2)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$
Incidence matrix	$\Theta(nm)$	$\mathcal{O}(m)$	$\mathcal{O}(m)$	$\mathcal{O}(nm)$

Notes

**Definition 1.22 (Paths, cycles)**

- Let  $G = (V, R)$  be a digraph. A *path* in  $G$  is a finite sequence

$$P = (v_0, r_1, v_1, \dots, r_k, v_k)$$

- with  $k \geq 0$ , such that  $v_0, v_1, \dots, v_k \in V$  and  $r_1, \dots, r_k \in R$  and  $\alpha(r_i) = v_{i-1}$  as well as  $\omega(r_i) = v_i$  for  $i = 1, \dots, k$ .
- We define the *starting node* of  $P$  as  $\alpha(P) = v_0$  and the *end node* as  $\omega(P) = v_k$ .
- We sometimes call  $P$  a  $v_0$ - $v_k$ -path.
- The *length*  $|P|$  of  $P$  is the number of traversed arcs in the path.
- If  $\alpha(P) = \omega(P)$  and  $k \geq 1$  we call  $P$  a *cycle*.
- A path is called *elementary* if it contains—except for the case, that start and end coincide—no node is traversed more than once.

**Example 1.23**

→ board

Notes

---

---

---

---

---

---

---

---

**Definition 1.24 (Connectedness and cuts)**

Let  $G = (V, E)$  be a graph.

- $G$  is called *connected* if for all  $v_1, v_2 \in G$  there exists an  $v_1$ - $v_2$ -path in  $G$ .
- A *disconnecting set of edges* is a subset  $Q \subseteq E$  such that  $G \setminus Q := (V, E \setminus Q)$  is unconnected.
- A *cut*  $Q$  is minimal disconnecting set of edges, i.e. any subset of  $Q$  is not disconnecting.
- A cut can be written in the form  $Q = (X, \bar{X})$  as the set of edges with one endpoint in  $X$  and one endpoint in  $\bar{X}$ . ( $X, \bar{X} \subseteq V$ ,  $\bar{X} = V \setminus X$ )
- Let  $s, t \in V$  and  $s \neq t$ , then a *s-t-cut* is a cut such that  $s \in X$  and  $t \in \bar{X}$ .

**Example 1.25**

→ board

Notes

---

---

---

---

---

---

---

---

18

**Definition 1.26 (Subgraphs and trees)**

Let  $G = (V, E)$  be a graph.

- A *subgraph/subdigraph* of  $G$  is a graph/digraph  $G' = (V', E')$  with  $V' \subseteq V$  and  $E' \subseteq E$ .
- A *spanning subgraph/subdigraph* of  $G$  is a subgraph/subdigraph  $G' = (V', E')$  with  $V' = V$ .
- A graph/digraph is called *acyclic* if it contains no cycle/dicycle.
- A *tree* is a connected, acyclic graph.
- A *spanning tree* is a connected, acyclic, spanning subgraph of  $G$ .
- We call vertex  $v \in V_T$  *leaf* of a tree  $T = (V_T, E_T)$  if there exists exactly one edge in  $e \in E_T$ , which has  $v$  as an endpoint.

**Example 1.27**

→ board

Notes

---

---

---

---

---

---

---

---

19

**Theorem 1.28**

*Equivalent characterizations for trees* Let  $G = (V, E, \gamma)$  be an undirected graph. Then, the following statements are equivalent:

- $G$  is a tree.
- $G$  contains no elementary cycle, but every supergraph of  $G$  with same node-set contains an elementary cycle.
- For every pair  $u, v \in V$  exists exactly one elementary path  $P$  with  $\alpha(P) = u$  and  $\omega(P) = v$ .
- $G$  is connected and for every edge  $e \in E$  the graph  $G' = (V, E \setminus \{e\}, \gamma)$  is not connected.
- $G$  is connected and  $|E| = |V| - 1$ .
- $G$  contains no elementary cycle and  $|E| = |V| - 1$ .

Notes

---

---

---

---

---

---

---

---

20

Minimum Spanning Tree Problem

Outline

- 2. Minimum Spanning Tree Problem
- 2.1 Kruskal’s Algorithm
- 2.2 Algorithm of Prim
- 2.3 IP Formulation

Notes

---

---

---

---

---

---

---

Minimum Spanning Tree Problem (MST)

Given a graph  $G = (V, E)$  and cost coefficients  $c_{ij} \forall (i, j) \in E$  on the edges, find a *minimum spanning tree (MST)*  $T = (V, E(T))$  of  $G$ , i. e.

$$\min_{T \in \mathcal{T}} c(T) = \min_{T \in \mathcal{T}} \sum_{e \in E(T)} c_e$$

where  $\mathcal{T}$  denotes the set of all spanning trees of  $G$ .

Notes

---

---

---

---

---

---

---

Minimum Spanning Tree Problem (MST)

Example 2.1

→ board

Remark 2.2

The MST problem has many applications in infrastructure design problems or occurs as a subproblem, e.g. design of communication networks, gas networks, highways.

Remark 2.3

In the following, we assume that the graph  $G = (V, E)$  is simple. This is no restriction, since a spanning tree never contains loops and in the case of parallel edges only the cheapest edge is included.

Additionally, we assume that  $G$  is connected.

Notes

---

---

---

---

---

---

---

Minimum Spanning Tree Problem (MST)

Definition 2.4

We call a subset  $F \subseteq E$  of edges of  $G$  *error free*, if there exists a MST  $F^*$  with  $F \subseteq E(F^*)$ . An edge  $e \in E$  is called *promising* for  $F$  if  $F \cup \{e\}$  is error free too.

Theorem 2.5

Let  $F \subseteq E$  be error free and  $(A, B)$  a cut in  $G$  with  $\delta(A) \cap F = \emptyset$ . If  $e \in \delta(A)$  is an edge with minimum weight in  $\delta(A)$ , then  $e$  is promising for  $F$ .

Proof.

→ board



Corollary 2.6

Let  $F \subseteq E$  be error free and  $U$  be a connected component of  $(V, F)$ . If  $e$  is an edge with minimum weight in  $\delta(U)$ , then  $e$  is promising for  $F$ .

Proof.

Follows directly from Theorem 2.5, since  $\delta(U) \cap F = \emptyset$ .



Notes

---

---

---

---

---

---

---

## Kruskal's algorithm

## 25

[illegible]

## Kruskal's algorithm

☐

26

[illegible]

## Kruskal's algorithm

## 27

[illegible]

## Kruskal's algorithm

## 28

[illegible]

The algorithm of Prim

Algorithm 2: The algorithm of Prim

**Input** : An undirected graph  $G = (V, E, \gamma)$  with edge weights  
 $c: E \rightarrow \mathbb{R}$

**Output**: The set of edges  $E_T$  of a MST

1 Choose  $s \in V$  arbitrarily, set  $E_T = \emptyset$  and  $S := \{s\}$

2 **while**  $S \neq V$  **do**

3     Choose an edge  $(u, v) \in \delta(S)$  with minimum cost. Let  $u \in S$  and  
           $v \in V \setminus S$

4     Set  $E_T := E_T \cup \{(u, v)\}$  and  $S := S \cup \{v\}$

5 **return**  $E_T$

Notes

The algorithm of Prim

Example 2.12

→ board

Theorem 2.13

For a connected graph  $G = (V, E, \gamma)$ , the algorithm of Prim computes a MST.

Proof.

Follows from Corollary 2.6 via induction, since  $S$  is always a connected component of  $G$ . □

Notes

Complexity of Prim’s Algorithm

- In every iteration one edge is added to the tree ( $\leadsto n - 1$  iterations).
- Every iteration takes  $\mathcal{O}(n)$  time to find the minimum cost edge.
- This leads to a worst-case running time of  $\mathcal{O}(n^2)$  for a naive implementation.
- more efficient if implemented using a Fibonacci-heap for the not connected vertices sorted wrt. to the cost of adding to the tree:  $\mathcal{O}(m + n \log n)$

Notes

Comparison

Algorithm	run time	Properties
Prim	$\mathcal{O}(n^2)$ [ $\mathcal{O}(m + n \log n)$ eff. Implem.]	<ul style="list-style-type: none"><li>• Iteratively increase the node set <math>S</math> by adding a cost minimal edge</li><li>• For dense graphs faster than Kruskal.</li><li>• Efficient implementation: <i>Fibonacci-heap</i></li></ul>
Kruskal	$\mathcal{O}(mn)$ [ $\mathcal{O}(m\alpha(n))$ eff. Implem.]	<ul style="list-style-type: none"><li>• Add iteratively a cost minimal edge which does not form a cycle</li><li>• For thin graphs faster than Prim.</li><li>• The sorting algorithm determines the complexity</li></ul>

Notes



IP Formulation

Let  $G = (V, E)$  be a graph with  $|V| = n$  nodes. We model MST as an integer program:

(Binary) decision variables  $x_{ij}$  with

$$x_{ij} := \begin{cases} 1 & \text{if } (i, j) \in E(T) \\ 0 & \text{otherwise} \end{cases}$$

and cost coefficients  $C_{ij}$  with

$$C_{ij} := \begin{cases} c_{ij} & \text{if } (i, j) \in E \\ \infty & \text{otherwise} \end{cases}$$

Notes

---

---

---

---

---

---

---

IP Formulation

$$\begin{aligned} \min \quad & \sum_{i=1}^n \sum_{j=1}^n C_{ij} x_{ij} \\ \text{s. t.} \quad & \sum_{i=1}^n \sum_{j=1}^n x_{ij} = n - 1 \\ & \sum_{i \in S} \sum_{\substack{j \in S \\ i < j}} x_{ij} \leq |S| - 1 \quad \forall S \subset V \\ & x_{ij} \in \{0, 1\} \\ \text{or} \quad & x_{ij} \in [0, 1] \\ \text{or} \quad & x_{ij} \geq 0 \end{aligned}$$

Notes

---

---

---

---

---

---

---

IP Formulation

Remark 2.14

- The previous IP formulation can be solved as an LP, since every extreme point of the feasible polyhedron is integral.
- $x \leq 1$  follows from second constraint with  $|S| = 2$
- Drawback: this formulation has an exponential number of constraints!

Notes

---

---

---

---

---

---

---

Shortest Path Problems

Outline

- 3. Shortest Path Problems
  - 3.1 Shortest Path Problem
  - 3.2 Shortest path problem with negative costs
  - 3.3 The all-pair shortest path problem

Notes

---

---

---

---

---

---

---

Shortest Path Problem

Assumptions for this part:

- A simple digraph  $G = (V, R)$ . As for spanning trees, this is no loss of generality.
- A function  $c: R \rightarrow \mathbb{R}$  that assigns *weights* or *costs* to every arc in the graph.

Definition 3.1

Let  $G = (V, R)$  be a digraph and  $c$  be a cost function as above. The *length* (or *weight* or *cost*)  $c(P)$  of a path  $P = (v_0, r_1, v_1, \dots, r_k, v_k)$  in  $G$  is defined by

$$c(P) := \sum_{i=1}^k c(r_i).$$

The length of a path  $P = (v_0)$  without arcs is thus  $c(P) = 0$ .

Notes

---

---

---

---

---

---

---

---

Shortest (Di-)Path Problem

Shortest (Di-)Path Problem

Let  $s \in V$ . For all vertices  $i \in V \setminus \{s\}$  compute a *shortest path*, i.e. a path  $P_{si}$  with *minimum cost/length/weight*

$$c(P_{si}) = \sum_{r \in P_{si}} c(r).$$

Notes

---

---

---

---

---

---

---

---

Definition 3.2

The *distance*  $\text{dist}_c(u, v, G)$  of two nodes  $u, v \in V$  with respect to the weight function  $c$  is given by

$$\text{dist}_c(u, v, G) := \inf \{ c(P) : P \text{ is a path from } u \text{ to } v \text{ in } G \}.$$

Remark 3.3

As usual, we set  $\inf(\emptyset) = +\infty$ .

Remark 3.4

1. If  $u \neq v$  and  $v$  is not reachable from  $u$ , we thus have  $\text{dist}_c(u, v) = +\infty$ .
2. If  $u$  and  $v$  coincide there are only two possibilities: Either it holds that  $\text{dist}_c(u, u) = 0$  or  $\text{dist}_c(u, u) = -\infty$ .

Example 3.5

→ board.

Notes

---

---

---

---

---

---

---

---

Observation 3.6

Let  $P$  be a shortest path from  $u$  to  $v$  and  $w$  is traversed by  $P$ . The partial path  $P_{uw}$  from  $u$  to  $w$  is a shortest  $u$ - $w$ -path. Analogously for  $P_{wv}$ .

Proof.

→ board.



Lemma 3.7

Let  $s \in V$ . Then it holds that

$$\text{dist}_c(s, v) \leq \text{dist}_c(s, u) + c(u, v) \text{ for all } (u, v) \in R.$$

Proof.

→ board.



Notes

---

---

---

---

---

---

---

---

**Definition 3.8 (Reduced costs, potential)**

Let  $p: V \rightarrow \mathbb{R}$  be a node weighting. We define the *reduced costs*  $c^p: R \rightarrow \mathbb{R}$  by

$$c^p(u, v) := c(u, v) + p(u) - p(v).$$

The node weighting is called *potential* for  $c$ , if for all  $(u, v) \in R$  it holds that  $c^p(u, v) \geq 0$ , thus

$$p(v) \leq p(u) + c(u, v) \text{ for all } (u, v) \in R.$$

**Observation 3.9**

If for  $s \in V$  all the values  $\text{dist}_c(s, v)$  are finite, then according to Lemma 3.7,  $p(v) := \text{dist}_c(s, v)$  is a potential.

41

Notes

---

---

---

---

---

---

---

---

Let  $P = (v_0, \dots, v_k)$  be a path from  $v_0$  to  $v_k$ . Then it holds that

$$\begin{aligned} c^p(P) &= \sum_{i=0}^{k-1} (c(v_i, v_{i+1}) + p(v_i) - p(v_{i+1})) \\ &= \sum_{i=0}^{k-1} c(v_i, v_{i+1}) + \sum_{i=0}^{k-1} (p(v_i) - p(v_{i+1})) \\ &= c(P) + p(v_0) - p(v_k), \end{aligned}$$

since the second sum is a telescoping sum.

**Observation 3.10**

- In the transition from  $c$  to  $c^p$ , the length of *every* way from  $v_0$  to  $v_k$  differs by the number  $p(v_0) - p(v_k)$  that is independent of the way.
- If  $P$  is a cycle it holds that  $v_0 = v_k$  and thus  $c(P) = c^p(P)$ .

42

Notes

---

---

---

---

---

---

---

---

**Observation 3.11 (Transition to reduced costs)**

Let  $p: V \rightarrow \mathbb{R}$  be a node weighting.

1.  $P$  is a shortest  $u$ - $v$ -path w.r.t.  $c$  if and only if  $P$  is a shortest  $u$ - $v$ -path w.r.t.  $c^p$ .
2. For every path  $P$  in  $G$  it holds that  $c(P) = c^p(P) + p(\omega(P)) - p(\alpha(P))$ .
3. For every cycle  $C$  in  $G$  it holds that  $c(C) = c^p(C)$ .

If we are given a potential  $p$  in  $G$ , it holds that  $c^p(u, v) \geq 0$  for all  $(u, v) \in R$ .

According to Observation 3.11 we can lead back the problem of finding a shortest path with respect to  $c$  to the problem of finding a shortest path with respect to the reduced costs  $c^p$ .

43

Notes

---

---

---

---

---

---

---

---

**Theorem 3.12**

Let  $G = (V, R)$  be a simple digraph and  $c: R \rightarrow \mathbb{R}$  be a weight function. A potential  $p$  in  $G$  exists if and only if there is no cycle with negative length in  $G$ . If  $c: R \rightarrow \mathbb{Z}$  is integral, we can choose  $p$  to be integral too.

**Proof.**

→ board. □

**Definition 3.13 (Shortest path tree)**

Let  $G$  be as above. A shortest path tree rooted in  $s$  is a tree  $T = (V', R')$  with the following properties:

1.  $V'$  is the set of nodes that is reachable from  $s$ .
2. For all  $v \in V' \setminus \{s\}$  the unique path from  $s$  to  $v$  in  $T$  is a shortest path from  $s$  to  $v$  in  $G$ .

44

Notes

---

---

---

---

---

---

---

---

Remark 3.14

A shortest path tree is not necessarily unique.

Example 3.15

→ board.

Theorem 3.16

Let  $s \in V$  be in such a way that every reachable cycle in  $G$  has non-negative weight. Then a shortest path tree emanating from  $s$  exists in  $G$ .

Proof.

→ board. □

Notes

---

---

---

---

---

---

---

---

The algorithms that we’re going to look at maintain a *predecessor graph*  $G_\pi$  that is defined as follows:

Definition 3.17

Let  $G_\pi = (V_\pi, R_\pi)$  with

$$V_\pi := \{v \in V: \pi[v] \neq \text{nil}\} \cup \{s\}$$
$$R_\pi := \{(\pi[v], v) \in R: v \in V_\pi \wedge v \neq s\}.$$

We will show that  $G_\pi$  is a shortest path tree with respect to  $s$ .

Notes

---

---

---

---

---

---

---

---

The algorithms that we’re going to look at share the same initialization `INIT` from Algorithm 3.

Algorithm 3: Initialization for shortest path algorithms

INIT( $G, s$ )

1

forall  $v \in V$  do

2      $d[v] := +\infty$

3      $\pi[v] := \text{nil}$

4  $d[s] := 0$

The values  $d[v]$  for  $v \in V$  are upper bounds for  $\text{dist}_c(s, v)$  that are adjusted during the runtime.

Notes

---

---

---

---

---

---

---

---

The algorithms also share the following method `Test` shown in Algorithm 4.

Algorithm 4: Algorithm that checks for an arc  $(u, v) \in R$ , whether it can be used to find a shorter path from  $s$  to  $v$  than the previously known with length  $d[v]$

TEST( $u, v$ )

1

if  $d[v] > d[u] + c(u, v)$  then

2      $d[v] := d[u] + c(u, v)$

3      $\pi[v] := u$

By calling `Test( $u, v$ )` for an arc  $(u, v) \in R$  it is checked whether the arc  $(u, v)$  can be used to construct a shorter way from  $s$  to  $v$  than the current upper bound  $d[v]$ .

Notes

---

---

---

---

---

---

---

---

**Theorem 3.18**

An algorithm is initialized with Algorithm 3 and performs an arbitrary number of test steps (Algorithm 4). Then the following holds during the runtime of the algorithm:

1. For the reduced costs  $c^d$  with respect to the node weighting  $d$  it holds that  $c^d(u, v) \leq 0$  for all  $(u, v) \in R_\pi$ .
2.  $d[v] \geq \text{dist}_c(s, v)$  for all  $v \in V$ .
3. Every cycle in  $G_\pi$  has negative length with respect to  $c$  and  $c^d$ .
4. If no cycle with negative length in  $G$  is reachable from  $s$ , the predecessor graph  $G_\pi = (V_\pi, R_\pi)$  is a tree with root  $s$  and  $G_\pi$  contains for every  $v \in V_\pi \setminus \{s\}$  a path from  $s$  to  $v$  with length at most  $d[v]$ .

**Proof.**

→ board. □

Notes

---

---

---

---

---

---

---

---

**Corollary 3.19**

If an algorithm after a number of iterations generates  $d[v] \leq \text{dist}_c(s, v)$  for every  $v \in V$ , it holds that  $d[v] = \text{dist}_c(s, v)$  for all  $v \in V$  and  $G_\pi$  is a shortest path tree rooted in  $s$ .

**Proof.**

→ board. □

According to Theorem 3.18 and Corollary 3.19 the “only” thing we have to do is to find a suitable sequence of test-steps, such that  $d[v] = \text{dist}_c(s, v)$  holds.

Notes

---

---

---

---

---

---

---

---

**Algorithm 5:** Dijkstra’s algorithm

Dijkstra( $G, c, s$ )

**Input** : A directed graph  $G = (V, R)$  with non-negative arc weight function  $c: R \rightarrow \mathbb{R}^+$  and a node  $s \in V$

**Output:** For all  $v \in V$  the distance  $\text{dist}_c(s, v)$  and a tree of shortest paths emanating from  $s$

```
1 INIT( $G, s$ )
2 PERM := {} // PERM is the set of permanently marked nodes
3 while PERM  $\neq V$  do
4   Choose  $u \in Q := V \setminus \text{PERM}$  with minimal key  $d[u]$ .
5   PERM := PERM  $\cup \{u\}$ 
6   forall adjacent nodes  $v$  of  $u$  with  $v \notin \text{PERM}$  do
7     | TEST( $u, v$ )
8 return  $d[]$  and  $G_\pi$  //  $G_\pi$  is spanned by the predecessor pointers  $\pi$ 
```

Notes

---

---

---

---

---

---

---

---

**Example 3.20 (SPP with nonnegative cost)**

→ board

**Theorem 3.21**

After termination of Dijkstra’s algorithm, it holds that  $d[v] = \text{dist}_c(s, v)$  for all  $v \in V$  and  $G_\pi$  is a shortest path tree rooted in  $s$ .

**Proof.**

→ board. □

**Example 3.22 (SPP with general cost)**

→ board

Notes

---

---

---

---

---

---

---

---

**Theorem 3.23**

A naïve implementation of Dijkstra’s algorithm solves the shortest dipath problem with nonnegative costs in  $\mathcal{O}(n^2)$  time.

**Proof.**

→ board □

**Remark 3.24**

Using more sophisticated data structures (Fibonacci-Heaps similar as in the algorithm of Prim), Dijkstra’s algorithm can be implemented with a worst-case running time of  $\mathcal{O}(m + n \log n)$ .

Notes

As we have seen in Example 3.22, Dijkstra’s algorithm fails if we allow negative weights on the arcs in  $G$ .

The following algorithm by Bellman and Ford (Algorithm 6) is capable of dealing with negative weights and is able to detect negative cycles in  $G$ .

The algorithm works in  $n$  steps. In every step every arc is checked exactly once.

**Lemma 3.25**

At the end of step  $k = 1, 2, \dots, n - 1$  in Algorithm 6 it holds for all  $v \in V$  that

$$d[v] \leq \min\{c(P) : P \text{ is a path from } s \text{ to } v \text{ with at most } k \text{ arcs}\}.$$

**Proof.**

→ board. □

Notes

**Algorithm 6:** Algorithm of Bellman and Ford

BELLMAN-FORD( $G, c, s$ )

**Input** : A directed graph  $G = (V, R)$  with arc weight function  $c: R \rightarrow \mathbb{R}$  and a node  $s \in V$

**Output:** For all  $v \in V$  the distance  $\text{dist}_c(s, v)$  and a tree of shortest paths emanating from  $s$

```
1 INIT( $G, s$ )
2 for  $k := 1, \dots, n - 1$  do
3   for all  $(u, v) \in R$  do
4     TEST( $u, v$ )
5 return  $d[]$  and  $G_\pi$  //  $G_\pi$  is spanned by the predecessor pointers  $\pi$ 
```

Notes

**Example 3.26**

→ board.

**Theorem 3.27**

1. If  $G$  contains no cycle of negative length that is reachable from  $s$ , at the termination of Algorithm 6 it holds that  $d[v] = \text{dist}_c(s, v)$  for all  $v \in V$ . Moreover,  $G_\pi$  is a shortest path tree rooted in  $s$ .
2. The algorithm of Bellman and Ford can be implemented with a worst-case running time of  $\mathcal{O}(mn)$ .

**Proof.**

→ board. □

Notes

For the last theorem we assumed that no cycle with negative length is reachable from  $s$ . In the following, we show how to modify the algorithm of Bellman and Ford to detect cycles with negative length.

Our modification is that, at the end of the algorithm, we iterate over all arcs once again and check the condition  $d[v] \leq d[u] + c(u, v)$ . If it holds that  $d[v] > d[u] + c(u, v)$  for one arc, we declare this as a certificate for a cycle of negative length.

**Example 3.28**  
→ board.

Notes

---

---

---

---

---

---

---

**Algorithm 7:** Algorithm that tests if  $G$  contains a cycle with negative length

TEST-NEGATIVE-CYCLE( $G, c, d$ )  
**Input** : A directed graph  $G = (V, R)$  with arc weight function  $c: R \rightarrow \mathbb{R}$  and the distance values  $d$  of the algorithm of Bellman and Ford  
**Output:** The information whether  $G$  contains a cycle of negative length or not

```
1 forall  $(u, v) \in R$  do
2   if  $d[v] > d[u] + c(u, v)$  then
3     return "Yes"
4 return "No"
```

Notes

---

---

---

---

---

---

---

**Theorem 3.29**

Algorithm 7 decides in  $\mathcal{O}(m + n)$  time after the termination of the algorithm of Bellman and Ford whether  $G$  contains a cycle of negative length that is reachable from  $s$  or not.

**Proof.**  
→ board. ☐

**Remark 3.30**

1. We can decide in  $\mathcal{O}(nm + m + n) = \mathcal{O}(nm)$  time whether a graph contains a cycle of negative length or not.
2. In an additional  $\mathcal{O}(n)$  time a cycle with negative length can be constructed if we know an arc  $(u, v) \in R$  with  $d[v] > d[u] + c(u, v)$ .

**Theorem 3.31**

For a directed graph  $G$  we can construct a cycle of negative length in  $\mathcal{O}(nm)$  time or determine that no such cycle exists.

Notes

---

---

---

---

---

---

---

In this section we look at the problem to solve the all-pair shortest path problem. In addition to the assumptions at the beginning of this chapter we assume that no cycle with negative length exists in  $G$  (this can be checked in  $\mathcal{O}(mn)$  time).

In principle, we could determine all distances  $\text{dist}_c(u, v)$  by applying  $n$ -times the algorithm of Bellman and Ford; once for every node  $v \in V$  as start node  $s$ . This leads to a worst-case complexity of  $\mathcal{O}(n^2m)$ .

In the following, we show how to do better with the concept of *dynamic programming*.

Notes

---

---

---

---

---

---

---

The idea of the algorithm of Floyd and Warshall is based on a recursion for the distance values.

We number the nodes  $V = \{v_1, \dots, v_n\}$  in an arbitrary manner and define for  $v_i, v_j \in V$  and  $k = 0, 1, \dots, n$  the value  $d_k(v_i, v_j)$  as the length (w.r.t.  $c$ ) of a shortest path from  $v_i$  to  $v_j$ , that traverses only the nodes  $\{v_i, v_j, v_1, \dots, v_k\}$ .

If no such path exists, we set  $d_k(v_i, v_j) := +\infty$ . The distance  $\text{dist}_c(v_i, v_j)$  then equals  $d_n(v_i, v_j)$ .

For  $k = 0$  the path may only traverse  $v_i$  and  $v_j$ , it thus holds that

$$d_0(v_i, v_j) := \begin{cases} c(v_i, v_j) & \text{if } (v_i, v_j) \in R \\ \infty & \text{else} \end{cases}.$$

61

Notes

---

---

---

---

---

---

---

Let  $P$  be a shortest path from  $v_i$  to  $v_j$  that only traverses  $\{v_i, v_j, v_1, \dots, v_k\}$ . Since, by assumption,  $G$  contains no cycle with negative length, we may assume that  $P$  is an elementary path.

If  $v_{k+1}$  is not traversed by  $P$ , it is  $c(P) = d_k(v_i, v_j)$ . If  $v_{k+1}$  is contained in  $P$ , we can split up  $P$  into two partial paths  $P_{v_i, v_{k+1}}$  from  $v_i$  to  $v_{k+1}$  and  $P_{v_{k+1}, v_j}$  from  $v_{k+1}$  to  $v_j$ . Then it holds that  $c(P_{v_i, v_{k+1}}) = d_k(v_i, v_{k+1})$  and  $c(P_{v_{k+1}, v_j}) = d_k(v_{k+1}, v_j)$ .

So for  $k \geq 1$  we get the following recursion

$$d_k(v_i, v_j) = \min \{d_k(v_i, v_j), \quad d_k(v_i, v_{k+1}) + d_k(v_{k+1}, v_j)\}.$$

This leads to the following algorithm...

62

Notes

---

---

---

---

---

---

---

**Algorithm 8:** Algorithm of Floyd and Warshall

```
FLOYD-WARSHALL( $G, c$ )
Input : A directed graph  $G = (V, R)$  with arc weight function  $c: R \rightarrow \mathbb{R}$ 
Output: For all  $u, v \in V$  the distance  $D_n[u, v] = \text{dist}_c(u, v)$ 

1 forall  $v_i, v_j \in V$  do
2    $D_0[v_i, v_j] := +\infty$ 
3 forall  $(v_i, v_j) \in R$  do
4    $D_0[v_i, v_j] := c(v_i, v_j)$ 
5 for  $k = 0, \dots, n-1$  do
6   for  $i = 1, \dots, n$  do
7     for  $j = 1, \dots, n$  do
8        $D_{k+1}[v_i, v_j] := \min\{D_k[v_i, v_j], D_k[v_i, v_{k+1}] + D_k[v_{k+1}, v_j]\}$ 
9 return  $D_n[]$ 
```

63

Notes

---

---

---

---

---

---

---

**Example 3.32**

→ get the example on the lecture's homepage.

**Theorem 3.33**

If  $G$  contains no cycle of negative length, the algorithm of Floyd and Warshall solves the all-pair shortest path problem in  $\mathcal{O}(n^3)$  time.

**Proof.**

Obvious. □

64

Notes

---

---

---

---

---

---

---



Outline

- 4. Network flow problems
  - 4.1 Flows and cuts
  - 4.2 Residual networks and flow augmenting paths
  - 4.3 The max-flow-min-cut theorem
  - 4.4 The algorithm of Ford and Fulkerson
  - 4.5 The algorithm of Edmonds and Karp
  - 4.6 Lower bounds and  $b$ -flows
  - 4.7 Flow decomposition
  - 4.8 Min cost flows

Notes

---

---

---

---

---

---

---

Flows and cuts

Assumptions for this chapter

Within this chapter,  $G$  denotes a finite directed graph  $G = (V, R, \alpha, \omega)$  that does not necessarily have to be simple.

Additionally, let  $f: R \rightarrow \mathbb{R}$  be an arbitrary function. We interpret  $f(r)$  as *flow value* on the arc  $r$ .

Notes

---

---

---

---

---

---

---

Flows and cuts

Definition 4.1 (Excess of a node  $v$ )

For some node  $v \in V$ , we denote by

$$f(\delta^+(v)) = \sum_{r \in \delta^+(v)} f(r)$$

the amount of flow leaving  $v$  and by

$$f(\delta^-(v)) = \sum_{r \in \delta^-(v)} f(r)$$

the amount of flow entering node  $v$ .

We denote by

$$\text{excess}_f(v) := f(\delta^-(v)) - f(\delta^+(v))$$

the *excess* of node  $v \in V$ .

Notes

---

---

---

---

---

---

---

Definition 4.2 (Flow, feasible flow, maximum flow)

- Let  $s, t \in V$  with  $s \neq t$ . An  $(s, t)$ -flow is a function  $f: R \rightarrow \mathbb{R}_+$  with

$$\text{excess}_f(v) = 0 \tag{1}$$

for all  $v \in V \setminus \{s, t\}$ .

- The node  $s$  is called *source node*, the node  $t$  is called *sink node* of flow  $f$ . If  $s$  and  $t$  are clear from the context, we shortly say *flow*.
- We call  $\text{val}(f) := \text{excess}_f(t)$  the *flow value* corresponding to  $f$ .
- If  $u: R \rightarrow \mathbb{R}$  is a *capacity function* on the arcs of  $G$ , we say a flow  $f$  is *feasible* if

$$0 \leq f(r) \leq u(r) \tag{2}$$

for all  $r \in R$ .

- A feasible flow is called *maximum  $(s, t)$ -flow* if it has maximum value of all feasible  $(s, t)$ -flows.

Notes

---

---

---

---

---

---

---

Example 4.3

→ board

Remark 4.4

- The conditions  $\text{excess}_f(v) = 0$  are called *flow conservation constraints* or *mass balance constraints*. All nodes except  $s$  and  $t$  are in balance.
- The inequalities  $0 \leq f(r) \leq u(r)$  for all  $r \in R$  are called *capacity constraints*.

Notes

---

---

---

---

---

---

---

Definition 4.5

Let  $(S, T)$  be a cut in  $G$ . We define the *forward part* of a cut by

$$\delta^+(S) := \{r \in R: \alpha(r) \in S \text{ and } \omega(r) \in T\}$$

and the *backward part* of a cut by

$$\delta^-(S) := \{r \in R: \alpha(r) \in T \text{ and } \omega(r) \in S\}.$$

Definition 4.6 (Excess of a cut)

For a set  $S \subseteq V$  we define the excess of  $S$  by

$$\text{excess}_f(S) := f(\delta^-(S)) - f(\delta^+(S)).$$

Notes

---

---

---

---

---

---

---

Lemma 4.7

Let  $f: R \rightarrow \mathbb{R}$  be an arbitrary function and  $S \subseteq V$ . Then it holds that

$$\text{excess}_f(S) = \sum_{v \in S} \text{excess}_f(v).$$

Proof.

→ board.



Notes

---

---

---

---

---

---

---

Lemma 4.8

If  $f$  is an  $(s, t)$ -flow and  $(S, T)$  an  $(s, t)$ -cut, then it holds that

$$\text{val}(f) = f(\delta^+(S)) - f(\delta^-(S)).$$

In particular, it follows that  $\text{excess}_f(t) = -\text{excess}_f(s)$ .

Proof.

→ board.



Notes

---

---

---

---

---

---

---

Definition 4.9 (Capacity of an  $(s, t)$ -cut)

If  $(S, T)$  is an  $(s, t)$ -cut in  $G$  with capacities  $u: R \rightarrow \mathbb{R}_+$  on the arcs, we denote by

$$u(\delta^+(S)) = \sum_{r \in \delta^+(S)} u(r)$$

the *capacity of the cut*  $(S, T)$ .

We say  $(S, T)$  is a *minimum  $(s, t)$ -cut*, if it has minimum capacity among all  $(s, t)$ -cuts.

Remark 4.10

Intuitively, the capacity of a cut  $(S, T)$  is an upper bound for the flow value of a feasible flow  $f$ . That this intuition is correct follows from Lemma 4.8.

Notes

---

---

---

---

---

---

---

---

Lemma 4.11

If  $f$  is a feasible  $(s, t)$ -flow and  $(S, T)$  an  $(s, t)$ -cut, then it holds that

$$\text{val}(f) \leq u(\delta^+(S)).$$

Since  $f$  and  $(S, T)$  are arbitrary, it follows that

$$\max_{f \text{ is a feasible } (s, t)\text{-flow in } G} \text{val}(f) \leq \min_{(S, T) \text{ is an } (s, t)\text{-cut in } G} u(\delta^+(S)).$$

Notes

---

---

---

---

---

---

---

---

Corollary 4.12

If  $f$  is a feasible flow and  $(S, T)$  a cut in  $G$  such that the flow value  $\text{val}(f)$  equals the capacity  $u(\delta^+(S))$  of the cut, then  $f$  is a maximum flow and  $(S, T)$  is a minimum cut.

Notes

---

---

---

---

---

---

---

---

Example 4.13

→ board.

Definition 4.14 (Residual network)

Let  $f$  be a feasible flow in  $G$  and  $l, u$  be lower and upper capacity bounds with  $0 \leq l(r) \leq u(r)$  for all arcs  $r \in R$  (we also allow  $u(r) = +\infty$ ).

The *residual network*  $G_f = (V, R_f, \alpha', \omega')$  has the same node set as  $G$ . The set of arcs is defined as follows:

- If  $r \in R$  and  $f(r) < u(r)$ ,  $G_f$  contains an arc  $+r$  with  $\alpha'(+r) = \alpha(r)$  and  $\omega'(+r) = \omega(r)$  and *residual capacity*  $u_f(+r) := u(r) - f(r)$ .
- If  $r \in R$  and  $f(r) > l(r)$ ,  $G_f$  contains an arc  $-r$  with  $\alpha'(-r) = \omega(r)$  and  $\omega'(-r) = \alpha(r)$  and *residual capacity*  $u_f(-r) := f(r) - l(r)$ .

Notes

---

---

---

---

---

---

---

---

Residual networks and flow augmenting paths

Example 4.15

→ board.

Remark 4.16

We have introduced the “signs” for the arcs in  $G_f$  in order to make clear that these arrows are in the residual network. On the other hand, for example  $-r$  emphasizes that this arc is an “flipped” version of  $r$ .

In the following we are using  $\sigma$  as a place holder for the sign, thus every arc in  $G_f$  can be written as  $\sigma r$  for all  $r \in R$ .

We denote by  $-\sigma r$  the inverse arc, thus  $-r$  for  $+r$  and vice versa.

Notes

---

---

---

---

---

---

---

Residual networks and flow augmenting paths

Definition 4.17 (Flow augmenting paths)

A path  $P$  from  $s$  to  $t$  in  $G_f$  is called *flow augmenting path* for the flow  $f$ . The *residual capacity*

$$\Delta(P) := \min_{\sigma r \in P} u_f(\sigma r)$$

of  $P$  is the minimum of the residual capacities of its arcs.

Example 4.18

→ board.

Observation 4.19

If there exists a flow augmenting path for some flow  $f$ ,  $f$  cannot be a maximum flow.

Notes

---

---

---

---

---

---

---

Residual networks and flow augmenting paths

In Lemma 4.11 we have shown that the capacity of an  $(s, t)$ -cut is an upper bound for the flow value of every feasible  $(s, t)$ -flow. We re-formulate this result with the help of residual networks.

Lemma 4.20

If  $f$  is a feasible  $(s, t)$ -flow and  $f'$  and maximum  $(s, t)$ -flow, it holds that

$$\text{val}(f') \leq \text{val}(f) + u_f(\delta_{G_f}^+(S))$$

for every  $(s, t)$ -cut  $(S, T)$  in  $G_f$ .

Proof.

→ board.



Notes

---

---

---

---

---

---

---

The max-flow-min-cut theorem

Observation 4.19 yields a necessary condition for the maximality of some flow  $f$ : there must not exist a flow augmenting path. We’re going to show that this condition is also sufficient.

Let  $f^*$  be a maximum  $(s, t)$ -flow. According to Observation 4.19 there does not exist a flow augmenting path for  $f^*$ . This means, that  $t$  is not reachable from  $s$  in  $G_f$  and the two sets

$$S := \{v \in V: v \text{ is reachable from } s \text{ in } G_f\}$$
$$T := \{v \in V: v \text{ is reachable from } t \text{ in } G_f\}$$

are non-empty.

Thus the two sets define a cut  $(S, T)$ .

Notes

---

---

---

---

---

---

---

The max-flow-min-cut theorem

Let  $r \in \delta^+(S)$  be an arc in the forward part of the cut.

Then it holds that  $f^*(r) = u(r)$ , because otherwise  $+r$  would be an arc in  $G_{f^*}$  and  $\omega(r)$  would be reachable from  $s$  in  $G_{f^*}$ , in contradiction to  $\omega(r) \in T$  (we have  $v \in S$  and thus by definition of  $S$  the node  $v$  is reachable from  $s$  in  $G_{f^*}$ ).

This shows that

$$f^*(\delta^+(S)) = u(\delta^+(S)). \tag{3}$$

Notes

---

---

---

---

---

---

---

The max-flow-min-cut theorem

Similarly, for every arc  $r \in \delta^-(S)$  it must hold that  $f^*(r) = 0$ , because otherwise  $-r$  would be an arc in  $G_{f^*}$  and  $\omega(r)$  as well as  $\alpha(r)$  would be reachable from  $s$ .

Thus it holds that

$$f^*(\delta^-(S)) = 0. \tag{4}$$

Combining (3) and (4) results in

$$\begin{aligned} u(\delta^+(S)) &= f^*(\delta^+(S)) - f^*(\delta^-(S)) \\ &\stackrel{*}{=} \text{val}(f^*), \end{aligned}$$

where we used Lemma 4.8 at  $\star$ .

Notes

---

---

---

---

---

---

---

The max-flow-min-cut theorem

Because of Corollary 4.12  $f^*$  is a maximum flow and at the same time  $(S, T)$  is a minimum cut, that is a cut with minimum capacity.

This leads to the following, famous theorem by Ford and Fulkerson.

Theorem 4.21 (Max-flow-min-cut theorem)

In a directed graph  $G$  with capacities  $u: R \rightarrow \mathbb{R}_+$  the value of a maximum  $(s, t)$ -flow is equal to the capacity of a minimum  $(s, t)$ -cut:

$$\max_{f \text{ is a feasible } (s, t)\text{-flow in } G} \text{val}(f) = \min_{(S, T) \text{ is an } (s, t)\text{-cut in } G} u(\delta^+(S)).$$

Notes

---

---

---

---

---

---

---

The max-flow-min-cut theorem

At the same time, we have proven the following theorem.

Theorem 4.22 (Augmenting path theorem)

A feasible  $(s, t)$ -flow  $f$  is a maximum flow if and only if there exists no flow augmenting path, in other words, there exists no path from  $s$  to  $t$  in the residual network  $G_f$ .

Notes

---

---

---

---

---

---

---

The algorithm of Ford and Fulkerson

Assumptions for this part

For the algorithmic part, we assume that  $G$  is connected and simple. The assumption that  $G$  is simple eases the notation, but again is no restriction: A bunch of parallel arcs  $r_1, \dots, r_k$  can be merged to one single arc with capacity  $\sum_{i=1}^k u(r_k)$ .

Also the assumption of connectedness is not a limitation as we otherwise solve the problem in the component that contains  $s$  and  $t$  (if no such component exists, the maximum flow is obviously 0).

Notes

---

---

---

---

---

---

---

---

The algorithm of Ford and Fulkerson

Theorem 4.22 motivates an obvious idea for a simple algorithm to solve the maximum flow problem.

We start with the zero flow  $f \equiv 0$  and as long as there are paths from  $s$  to  $t$  in  $G_f$  we augment  $f$  along such a path and update the residual network  $G_f$ .

If the capacities are integral, this means  $u: R \rightarrow \mathbb{N}$ , Algorithm 9 augments the flow in every step by an integer, since if the flow is integral, the residual capacities are integral too.

Moreover, the flow is increased at least by 1 unit. Let  $U := \max\{u(r) : r \in R\}$ . The cut  $(s, V \setminus \{s\})$  has at most  $n - 1$  arcs with capacity at most  $U$ . Thus the cut has capacity at most  $(n - 1)U$ .

Notes

---

---

---

---

---

---

---

---

The algorithm of Ford and Fulkerson

Algorithm 9: Generic algorithm based on flow augmenting paths

FORD-FULKERSON( $G, u, s, t$ )

**Input** : A simple directed graph  $G = (V, R)$ , a non-negative capacity function  $u: R \rightarrow \mathbb{R}$ , two nodes  $s, t \in V$ .

**Output**: A maximum  $(s, t)$ -flow  $f$  (for a "clever" choice of flow augmenting paths)

1 Set  $f(r) = 0$  for all  $r \in R$

2 **while** there exists a path from  $s$  to  $t$  in  $G_f$  **do**

3     Choose such a path  $P$

4     Set  $\Delta := \min\{u_f(\sigma r) : \sigma r \in P\}$      // residual capacity of path  $P$

5     Augment  $f$  along  $P$  by  $\Delta$  units

6     Update  $G_f$

Notes

---

---

---

---

---

---

---

---

The algorithm of Ford and Fulkerson

Example 4.23

→ board.

Theorem 4.24

*If all capacities are integral, Algorithm 9 terminates after  $\mathcal{O}(nU)$  augmenting steps and  $\mathcal{O}((m + n)nU)$  time with an integral maximum flow. Here,  $U := \max\{u(r) : r \in R\}$  denotes the maximum capacity in the graph.*

Example 4.25

→ board.

Theorem 4.26 (Integrality theorem)

*If all capacities are integral, there exists an integral maximum flow.*

Notes

---

---

---

---

---

---

---

---

The algorithm of Ford and Fulkerson

Example 4.27

→ board.

Theorem 4.28

If all capacities are rational numbers, Algorithm 9 terminates after a finite number of steps.

Proof.

→ Multiply all capacities with the lowest common denominator  $K \rightsquigarrow$  solve problem with integral capacities and divide by  $K$ . □

Notes

---

---

---

---

---

---

---

The algorithm of Ford and Fulkerson

Theorem 4.29

For irrational capacities, it may happen that Algorithm 9 does not terminate after a finite number of steps and the “limit flow” (the limit of generated flows) is not even a maximum flow.

Proof.

→ board. □

Notes

---

---

---

---

---

---

---

It remains an open question in Algorithm 9 how to choose the flow augmenting path. For the previous theoretical results, it sufficed to use *any* flow augmenting path.

If the flow augmentation occurs along a shortest  $s$ - $t$ -path in  $G_f$  in Algorithm 9, this leads to the following algorithm by Edmonds and Karp.

Notes

---

---

---

---

---

---

---

The algorithm of Edmonds and Karp

Algorithm 10: Algorithm of Edmonds and Karp

EDMONDS-KARP-MAXFLOW( $G, u, s, t$ )

Input : A simple directed graph  $G = (V, R)$ , a non-negative capacity function  $u: R \rightarrow \mathbb{R}$ , two nodes  $s, t \in V$ .

Output: A maximum  $(s, t)$ -flow  $f$ .

1 Set  $f(r) = 0$  for all  $r \in R$

2 while there exists a path from  $s$  to  $t$  in  $G_f$  do

3     Choose such a shortest path  $P$

4     Set  $\Delta := \min\{u_f(\sigma r) : \sigma r \in P\}$  // residual capacity of path  $P$

5     Augment  $f$  along  $P$  by  $\Delta$  units

6     Update  $G_f$

Notes

---

---

---

---

---

---

---

The algorithm of Edmonds and Karp

Lemma 4.30

Let  $G = (V, R)$  be a directed graph. We again denote by  $\text{dist}(s, t, G)$  the length of a shortest path from  $s$  to  $t$  in  $G$  and by  $R_{st}(G)$  the set of all arcs of  $G$ , that are part of shortest  $s$ - $t$ -paths. Let  $R_{st}(G)^{-1} := \{r^{-1} : r \in R_{st}(G)\}$ , where  $r^{-1}$  denotes the inverse arc of  $r$ .

Then it holds for the graph  $G'$ , that emerges from adding all arcs from  $R_{st}(G)^{-1}$  to  $G$ :

$$\begin{aligned}\text{dist}(s, t, G') &= \text{dist}(s, t, G) \\ R_{st}(G') &= R_{st}(G)\end{aligned}$$

Proof.

→ board. □

Notes

---

---

---

---

---

---

---

---

The algorithm of Edmonds and Karp

Theorem 4.31

Let  $G$  be a network with integer, rational or real capacities. The algorithm of Edmonds and Karp terminates after  $\mathcal{O}(nm)$  iterations with a maximum flow. The overall complexity of the algorithm is  $\mathcal{O}(nm^2)$ .

Proof.

→ board. □

Notes

---

---

---

---

---

---

---

---

Lower bounds and  $b$ -flows

In some applications it makes sense to introduce additional lower bounds  $l: R \rightarrow \mathbb{R}_+$  in addition to the upper bounds  $u: R \rightarrow \mathbb{R}_+$  on every arc  $r \in R$ .

Finding a maximum flow in this context means to find a maximum flow that also obeys the condition  $l(r) \leq f(r) \leq u(r)$  on every arc in  $G$ .

Assumption

Let  $G = (V, R, \alpha, \omega)$  be a finite directed graph,  $l: R \rightarrow \mathbb{R}_+$  be a function that assigns a lower bound on the capacities to every arc  $r \in R$  and  $u: R \rightarrow \mathbb{R}_+$  be a function that assigns an upper bound on the capacities to every arc  $r \in R$  with  $0 \leq l(r) \leq u(r)$  for all  $r \in R$ .

Does such a flow always exist?

Notes

---

---

---

---

---

---

---

---

Lower bounds and  $b$ -flows

Example 4.32

→ board.

We have defined the excess of a node  $v \in V$  as

$$\text{excess}_f(v) := f(\delta^-(v)) - f(\delta^+(v))$$

for some flow function  $f$ .

For an  $(s, t)$ -flow we required the flow to obey the flow conservation constraint  $\text{excess}_f(v) = 0$  for all  $v \in V \setminus \{s, t\}$ . Thus it holds

$$\text{excess}_f(v) = \begin{cases} -\text{val}(f) & \text{if } v = s \\ \text{val}(f) & \text{if } v = t \\ 0 & \text{else} \end{cases}$$

Notes

---

---

---

---

---

---

---

---



Lower bounds and  $b$ -flows

Definition 4.33 ( $b$ -flow, circulation)

Let  $b: V \rightarrow \mathbb{R}$  be a node labeling. A function  $f: R \rightarrow \mathbb{R}$  is called  $b$ -flow in  $G$  if

$$\text{excess}_f(v) = b(v) \quad \text{for all } v \in V.$$

For the special case  $b(v) = 0$  for all nodes  $v \in V$ , we call this 0-flow a *circulation*.

If  $l$  and  $u$  are capacity bounds as in the previous assumption, we say a  $b$ -flow  $f$  or a circulation  $\beta$  is feasible if  $l(r) \leq f(r), \beta(r) \leq u(r)$  for all  $r \in R$ .

Notes

---

---

---

---

---

---

---

---

Lower bounds and  $b$ -flows

Remark 4.34

Every  $(s, t)$ -flow is a  $b$ -flow with  $b(s) = -\text{val}(f)$ ,  $b(t) = \text{val}(f)$  and  $b(v) = 0$  for all  $v \in V \setminus \{s, t\}$ .

Furthermore, every function  $h: R \rightarrow \mathbb{R}_+$  is a  $b$ -flow for  $b(v) := \text{excess}_h(v)$ .

A necessary condition for the existence of a  $b$ -flow is  $\sum_{v \in V} b(v) = 0$ , since for every function  $f: R \rightarrow \mathbb{R}_+$  it holds that  $\sum_{v \in V} \text{excess}_f(v) = 0$ .

Since every circulation  $\beta$  in  $G$  is an  $(s, t)$ -flow for an arbitrary choice of  $s, t \in V$  with flow value  $\text{val}(\beta) = 0$ , Lemma 4.8 yields the following corollary:

Corollary 4.35

For a circulation  $\beta$  in  $G$  and an  $(S, T)$  cut it holds that

$$\beta(\delta^+(v)) = \beta(\delta^-(v)).$$

Notes

---

---

---

---

---

---

---

---

Lower bounds and  $b$ -flows

We have seen that a flow with lower and upper bound does not necessarily exist. How can we decide whether such a flow exists, and, if one exists, calculate a feasible  $b$ -flow?

Let  $G = (V, R, \alpha, \omega)$  be a directed graph with lower and upper bound  $l, u: R \rightarrow \mathbb{R}_+$ . We construct a super graph  $G' = (V', R', \alpha', \omega')$  in the following way: we add two new nodes  $s'$  and  $t'$  to this graph.

There exists one arc from  $s'$  to every node  $v \in V$  and one arc from every  $v \in V$  to the new node  $t'$ .

$$\begin{aligned} V' &= V \cup \{s', t'\} \\ R' &= R \cup \{(s', v) : v \in V\} \cup \{(v, t') : v \in V\} \end{aligned}$$

Notes

---

---

---

---

---

---

---

---

Lower bounds and  $b$ -flows

We set the lower bound  $l'(r') := 0$  for all arcs  $r' \in R'$  and define the upper bound as follows:

$$\begin{aligned} u'(r) &:= u(r) - l(r) && \text{for all } r \in R \\ u'(s', v) &:= \sum_{r \in \delta^-(v)} l(r) = l(\delta^-(v)) && \text{for all } v \in V, \text{ "minimum inflow"} \\ u'(v, t') &:= \sum_{r \in \delta^+(v)} l(r) = l(\delta^+(v)) && \text{for all } v \in V, \text{ "minimum outflow"} \end{aligned}$$

For a node  $v \in V$  we denote by  $\delta^+(v)$  the arcs emanating from  $v$  in  $G$  and by  $\delta_{G'}^+(v) = \delta^+(v) \cup \{(v, t')\}$  the corresponding arcs in  $G'$ . We also use the corresponding notion of  $\delta^-(v)$  for the incoming arcs.

Notes

---

---

---

---

---

---

---

---

Lower bounds and *b*-flows

Lemma 4.36

Let  $f: R \rightarrow \mathbb{R}$  be an arbitrary function. Then it holds for every function  $f': R' \rightarrow \mathbb{R}$  with the property

$f'(r) = f(r) - l(r)$   
 $f'(r') = u'(r')$

for all  $r \in R$   
for all  $r' \in R' \setminus R$

that  $f'(\delta_{G'}^+(v)) = f(\delta^+(v))$  and  $f'(\delta_{G'}^-(v)) = f(\delta^-(v))$  for all  $v \in V$ .

Proof.

→ board. ☐

Notes

---

---

---

---

---

---

---

Lower bounds and *b*-flows

Theorem 4.37

The exists a feasible circulation in  $G$  with respect to the lower and upper bounds  $l$  and  $u$  if and only if the maximum  $(s', t')$ -flow in  $G'$  has the value  $F := \sum_{r \in R} l(r)$ .

Proof.

→ board. ☐

Corollary 4.38

Let  $G$  be as in the previous assumption. By computing a maximum flow we either compute a feasible circulation with respect to the lower and upper bounds  $l$  and  $u$  in  $G$  or determine that no such circulation exists.

If such a circulation exists and  $l$  and  $u$  are integral, the so found circulation is integral too.

Notes

---

---

---

---

---

---

---

Flow decomposition

Definition 4.39 (Path flow, circulation)

A path flow  $f$  along a simple path  $P$  in  $G$  (with flow value  $\delta > 0$ ) is an  $(\alpha(P), \omega(P))$ -flow that vanishes on all arcs  $r \notin R(P)$ , that is so to say

$$f(r) = \begin{cases} \delta & \text{if } r \in R(P) \\ 0 & \text{otherwise.} \end{cases}$$

Analogously we define a circulation  $\beta$  along an elementary cycle  $C$  in  $G$  as

$$\beta(r) = \begin{cases} \delta & \text{if } r \in R(C) \\ 0 & \text{otherwise.} \end{cases}$$

Notes

---

---

---

---

---

---

---

Flow decomposition

Let  $\mathcal{P}$  be the set of all simple paths in  $G$  that are no cycles and  $\mathcal{C}$  be the set of all elementary cycles in  $G$ .

If  $f_P$  for some  $P \in \mathcal{P}$  is a path flow and  $\beta_C$  for some  $C \in \mathcal{C}$  is a circulation, we can construct an arc label  $f$  by

$$f(r) := \sum_{\substack{r \in P \\ P \in \mathcal{P}}} f_P(r) + \sum_{\substack{r \in C \\ C \in \mathcal{C}}} \beta_C(r).$$

It follows, that for all  $v \in V$

$$\text{excess}_f(v) = \sum_{P \in \mathcal{P}} \text{excess}_{f_P}(v)$$

holds. This allows us to construct a *b*-flow from path flows and circulations.

The following theorem shows that the opposite is also true.

Notes

---

---

---

---

---

---

---

Flow decomposition

Theorem 4.40 (Flow decomposition theorem)

Every  $b$ -flow  $f: R \rightarrow \mathbb{R}_+$  can be expressed as the sum of  $m + n$  path flows and circulations with the following properties:

- 1. For every path flow  $f_P$   $P$  is a path from a node  $v$  with  $b(v) < 0$  to a node  $u$  with  $b(u) > 0$ .
- 2. In the linear combination there occur at most  $m$  circulations.

If  $f$  is integral, all path flows and circulations are also integral.

Proof.

→ board. □

Example 4.41

→ board.

Notes

---

---

---

---

---

---

---

---

Flow decomposition

Remark 4.42

The method described in the proof of Theorem 4.40 can be implemented with a worst-case time complexity of  $\mathcal{O}((n + m)^2)$ . A path or a cycle can be found in  $\mathcal{O}(m + n)$  time and there occur at most  $n + m$  path flows and cycles.

Remark 4.43

The result further shows that  $\mathcal{O}(n + m)$  flow augmenting paths are sufficient to construct a maximum flow. This is substantially less than the bound of  $\mathcal{O}(nm)$  iterations in the algorithm of Edmonds and Karp.

However, it is not clear how to turn this into an efficient algorithm, since we already assumed both the knowledge of a maximum flow and that of a flow decomposition.

Notes

---

---

---

---

---

---

---

---

Min cost flows

Assumption for this part

In the following let  $G = (V, R, \alpha, \omega)$  be a finite graph and  $l, u$  be functions that assign a lower and upper capacities to the arcs of  $G$ , where it holds that  $0 \leq l(r) \leq u(r)$  for all  $r \in R$ . We also allow the case  $u(r) = +\infty$  for  $r \in R$ . Let  $b: V \rightarrow \mathbb{R}$  be a function that assigns the desired excess to the nodes. Additionally, let  $c: R \rightarrow \mathbb{R}$  be a function that assigns flow costs to the arcs.

Definition 4.44 (Flow cost)

Let  $c: R \rightarrow \mathbb{R}$  be an arc labeling that we call *cost function*. For some  $b$ -flow  $f$  the *flow costs* are given by

$$c(f) := \sum_{r \in R} c(r) \cdot f(r).$$

We extend the cost function  $c$  on the arcs in the residual network by  $c(+r) := c(r)$  and  $c(-r) := -c(r)$ .

Notes

---

---

---

---

---

---

---

---

Min cost flows

Min cost flow problem

For a given graph  $G$  with capacities  $l, u: R \rightarrow \mathbb{R}_+$ , excess values  $b: V \rightarrow \mathbb{R}$  and flow costs  $c: R \rightarrow \mathbb{R}$ , the problem of finding a feasible  $b$ -flow with minimum flow cost  $c(f)$  is called *min cost flow problem*.

Notes

---

---

---

---

---

---

---

---

Min cost flows

Remark 4.45

In principle, we also allow negative values for the costs on the arcs  $r \in R$ . Additionally, we allow infinite upper flow bounds on the arcs. This allows us to model the maximum flow problem as a special instance of a min cost flow problem.

Let  $G$  be a directed graph with lower and upper flow bounds  $l$  and  $u$  such that  $0 \leq l(r) \leq u(r)$  for all  $r \in R$  and  $s, t \in V$  be the two nodes for those we want to compute a maximum  $(s, t)$ -flow. We extend  $G$  to  $G'$  by adding a new arc  $r_{ts}$  from  $t$  to  $s$  with cost  $c(r_{ts}) := -1$  and capacities  $l(r_{ts} = 0)$  and  $u(r_{ts}) = +\infty$ . For all other arcs  $r \in R$ , we set  $c(r) = 0$ , for all nodes  $v \in V$  we set  $b(v) = 0$ .

An  $(s, t)$ -flow  $f$  in  $G$  with  $\text{val}(f) = F$  exists if and only if  $G'$  contains a cost minimal  $b$ -flow with cost  $-F$ .

Notes

---

---

---

---

---

---

---

---

Min cost flows

For the calculation of maximum flows, we augmented an  $(s, t)$ -flow along a flow augmenting path  $P$  (that is a path in the residual network) by a value of  $\delta > 0$ . For all arcs  $+r \in P$  we augmented the flow value by  $\delta$ , for all arcs  $-r \in P$  we reduced the flow value by  $\delta$ .

For min cost flows we additionally need the augmentation of a  $b$ -flow along a cycle in  $G_f$ . Analogously we augment the flow along all arcs  $+r \in C$  and reduce the flow along all arcs  $-r \in C$ . Obviously, the excess remains unchanged in all vertices.

Notes

---

---

---

---

---

---

---

---

Min cost flows

Let  $f$  be a  $b$ -flow and  $\beta_C$  be a circulation in  $G_f$  along an elementary cycle  $C = (v_0, \sigma_1 r_1, v_1, \dots, \sigma_k r_k, v_k = v_0)$  in  $G_f$  with flow value  $\delta > 0$ .

Then  $f + \beta_C$ , defined by

$$(f + \beta_C)(r) := \begin{cases} f(r) + \delta & \text{if } r = r_i \text{ and } \sigma_i = + \\ f(r) - \delta & \text{if } r = r_i \text{ and } \sigma_i = - \\ f(r) & \text{otherwise} \end{cases},$$

is again a  $b$  flow in  $G$ . The costs  $c(f + \beta_C)$  are given by  $c(f) + c(\beta_C)$ .

With this notion we can formulate and proof the following theorem:

Notes

---

---

---

---

---

---

---

---

Min cost flows

Theorem 4.46

Let  $f$  and  $f'$  be feasible  $b$ -flows in  $G$  with respect to  $l$  and  $u$ .  $f'$  can be written as the sum of  $f$  and at most  $2m$  circulations  $\beta_{C_1}, \dots, \beta_{C_p}$  in  $G_f$ . It holds that  $c(f') = c(f) + \sum_{i=1}^p c(\beta_{C_i})$ .

Proof.

→ board. □

Theorem 4.47 (Cycle criterion for cost minimal  $b$ -flow)

Let  $G$  be as in the assumption for this part.  $f$  is a cost minimal  $b$ -flow if and only if the residual network  $G_f$  does not contain a cycle of negative length with respect to the cost function  $c$ .

Proof.

→ board. □

Notes

---

---

---

---

---

---

---

---

Min cost flows

Additional assumptions for the algorithmic part

In addition to the previous assumptions we demand the following three properties for the network flow problem:

- 1. There exists a feasible  $b$ -flow with respect to  $l$  and  $u$  in  $G$ . In particular, this means that  $\sum_{v \in V} b(v) = 0$ .
- 2. For all  $u, v \in V$  with  $u \neq v$  there exists a path from  $u$  to  $v$  in  $G$  that consists only of arcs with infinite capacity (therefore, this path also exist in  $G_f$  for an arbitrary  $f: R \rightarrow \mathbb{R}_+$ ).
- 3. It is  $l(r) = 0$  and  $c(r) \geq 0$  for all  $r \in R$ .

None of these points is a loss of generality. In 1. we ensure that a feasible solution exists. This can be tested by the calculation of a maximum flow. 2. can be enforced by adding new arcs with infinite costs. None of these will be used in an optimal solution.

Notes

---

---

---

---

---

---

---

Min cost flows

Theorem 4.47 motivates the following algorithm by Klein to compute min cost flows.

Algorithm 11: Algorithm of Klein to compute min cost flows

```
MINCOSTFLOW-KLEIN( $G, l, u, b, c$ )  
  
  Input : A simple directed graph  $G = (V, R, \alpha, \omega)$  with capacity functions  
           $0 \leq l(r) \leq u(r)$  for all  $r \in R$ , desired excess values  $b: V \rightarrow \mathbb{R}$  and  
          flow costs  $c: R \rightarrow \mathbb{R}_+$ .  
  
  Output: A minimum cost  $b$ -flow  $f$ . If  $l, u$  and  $c$  are integral, so is  $f$ .  
  
  1 Compute a feasible  $b$ -flow  $f$  // Can be done by computing a maximum flow  
  2 while the residual network  $G_f$  contains a negative cycle  $C$  do  
  3   | Let  $\Delta := \min_{\sigma r \in C} c(\sigma r)$  be the minimal residual capacity along  $C$   
  4   | Augment  $f$  along  $C$  by  $\Delta$  // Eliminate the negative cycle  $C$   
  5 return  $f$ 
```

Notes

---

---

---

---

---

---

---

Min cost flows

Example 4.48

→ board

Theorem 4.49

If all excesses  $b$ , capacities  $u$  and costs  $c$  are integral, the algorithm of Klein terminates after  $\mathcal{O}(mUC)$  iterations with an integral min cost  $b$ -flow. Here we denote by  $U := \max\{u(r): r \in R\}$  the maximum capacity and by  $C := \max\{c(r): r \in R\}$  the maximum cost value in  $G$ .

Proof.

→ board.



Corollary 4.50 (Integrality theorem for min cost flows)

If all input data are integral, there exists an integral min cost  $b$ -flow.

Notes

---

---

---

---

---

---

---

Min cost flows

Remark 4.51

Algorithm 11 does not specify how to find an initial start flow and how to find cycles with negative length in the residual network. One can show that cycles  $C$  with minimum average weight  $c(C)/|C|$  lead to polynomial-time algorithms.

Notes

---

---

---

---

---

---

---

**Theorem 4.52 (Potential criterion for min cost b-flows)**  
Let  $G$  be as in the assumptions.  $f$  is a min cost b-flow if and only if there exists a potential in  $G_f$  with respect to the weights  $c$ . That means there exists a function  $p: V \rightarrow \mathbb{R}$  with  $p(v) \leq c(\sigma r) + p(u)$  for all  $\sigma r \in G_f$  with  $u = \alpha(\sigma r)$  and  $v = \omega(\sigma r)$ .

**Proof.**  
→ board. □

An equivalent formulation can be stated with reduced costs.

**Corollary 4.53 (Reduced cost criterion for min cost b-flows)**  
Let  $G$  be as in the assumptions.  $f$  is a min cost b-flow if and only if there exists a node labeling  $p: V \rightarrow \mathbb{R}$  with  $c^p(\sigma r) \geq 0$  for all  $\sigma r \in G_f$ .

Notes

---

---

---

---

---

---

---

---

**Definition 4.54 (Pseudo flow)**  
Let  $G$  be as in the assumptions and  $l \equiv 0$ . A feasible *pseudo flow* in  $G$  is a function  $f: R \rightarrow \mathbb{R}$  with  $0 \leq f(r) \leq u(r)$  for all  $r \in R$ .  
For a pseudo flow we define the *imbalance* of a node  $v \in V$  by  
$$\text{imbal}_f(v) := \text{excess}_f(v) - b(v).$$
  
If  $\text{imbal}_f(v) > 0$  we call  $v$  a *surplus node*. If  $\text{imbal}_f(v) < 0$  we call  $v$  a *deficit node*. A node  $v \in V$  with  $\text{imbal}_f(v) = 0$  is called *satisfied*.

Notes

---

---

---

---

---

---

---

---

For a pseudo flow  $f$  we denote by  $S_f$  and  $D_f$  the set of surplus or deficit nodes. It is  
$$\sum_{v \in V} \text{imbal}_f(v) = \underbrace{\sum_{v \in V} \text{excess}_f(v)}_{=0 \text{ (Lemma)}} - \underbrace{\sum_{v \in V} b(v)}_{=0 \text{ by assumption}} = 0,$$
  
therefore  
$$\sum_{v \in S_f} \text{imbal}_f(v) = - \sum_{v \in D_f} \text{imbal}_f(v). \tag{5}$$
  
(5) yields the following handy observation.

Notes

---

---

---

---

---

---

---

---

**Observation 4.55**  
For a pseudo flow  $f$ , the set of surplus nodes  $S_f$  is empty if and only if the set of deficit nodes  $D_f$  is empty.

Notes

---

---

---

---

---

---

---

---

Algorithm 12: Successive shortest path algorithm

SUCCESSIVE-SHORTEST-PATH( $G, u, b, c$ )

**Input** : A directed graph as in the assumptions.

**Output**: A minimum cost  $b$ -flow  $f$ . If  $l, u$  and  $c$  are integral, so is  $f$ .

1

Set  $f(r) := 0$  for all  $r \in R$  and  $p(v) := 0$  for all  $v \in V$

2

Set  $\text{imbal}_f(v) := -b(v)$  for all  $v \in V$

3

Compute the set of surplus and deficit nodes

$S_f = \{v \in V: \text{imbal}_f(v) > 0\}$

$D_f = \{v \in V: \text{imbal}_f(v) < 0\}$

4

**while**  $S_f \neq \emptyset$  **do**

5

Choose a node  $s \in S_f$  and a node  $t \in D_f$

6

Compute the distances  $d(v) = \text{dist}_{c^p}(s, f, G_f)$  from  $s$  to all other nodes in  $G_f$

with respect to the reduced costs  $c^p$

7

Let  $P$  be a shortest  $s$ - $t$ -path

8

Set  $\Delta := \min\{u_f(\sigma t): \sigma t \in P\}$

9

Update  $p := p + d$

10

$\varepsilon := \min\{\text{imbal}_f(s), -\text{imbal}_f(t), \Delta\}$

11

Augment  $f$  along  $P$  by  $\Delta$

12

Update  $f, G_f, S_f$  and  $D_f$

13

**return**  $f$

121

Notes

Notes

Outline

5. Dynamic network flow problems

5.1

Maximum flows over time

5.2

Earliest arrival flows

5.3

Minimum cost flows over time

5.4

Quickest flows

123

Notes

This section is still under construction.

Notes