Graph Theory

Network Application Diagnostics
B2M32DSAA

Radek Mařík

Czech Technical University
Faculty of Electrical Engineering
Department of Telecommunication Engineering
Prague CZ

October 2, 2025

Outline

Outline

Asymptotic Notation [CLRS09, Erc15]

Let $c, c_1, c_2 \in \mathbb{R}^{>0}$, $n_0, n \in \mathbb{N}$, $f, g \in \mathbb{N} \to \mathbb{R}^+$

Asymptotic upper bound (CZ horní asymptotický odhad)

$f(n) \in O(g(n))$, if $(\exists c > 0)(\exists n_0)(\forall n > n_0) : |f(n)| \leq |c \cdot g(n)|$

Asymptotic lower bound (CZ dolní asymptotický odhad)

$f(n) \in \Omega(g(n))$, if $(\exists c > 0)(\exists n_0)(\forall n > n_0) : |c \cdot g(n)| \leq |f(n)|$

Asymptotic tight bound (CZ optimální asymptotický odhad)

$f(n) \in \Theta(g(n))$, if $\Theta(g(n)) \stackrel{\text{def}}{=} O(g(n)) \cap \Omega(g(n))$
$(\exists c_1, c_2 > 0)(\exists n_0)(\forall n > n_0) : |c_1 \cdot g(n)| < |f(n)| < |c_2 \cdot g(n)|$

NP-Completeness [CLRS09, Erc15]

P and NP

- **P - Polynomial**. Problems that can be solved in polynomial time.
- **NP - Nondeterministic Polynomial**. A problem is in NP if you can in polynomial time by a *certifier* test whether a solution is correct without worrying about how hard it might be to find the solution.
  - Nondeterministic is a fancy way of talking about guessing a solution.
- $P \subseteq NP$ (??? $P = NP$???)

NP-complete and NP-hard

- **NPH - NP-hard**. An NPH problem is a problem which is as hard as any problem in NP
  - An NPH problem does not need to have a certificate.
- **NPC - NP-complete**. A problem is NPC if it is NP and is as hard as any problem in NP
  - A problem A is NPC if it is both NPH and in NP, $NPC = NP \cap NPH$.

Complexity Classes Other Than NP [CLRS09, Erc15]

Complexity classes harder than NP

- **PSPACE**. Problems that can be solved using a reasonable amount of memory
  - defined formally as a polynomial in the input size
  - without regard to how much time the solution takes.
- **EXPTIME**. Problems that can be solved in exponential time.
- **Undecidable**. For some problems, we can prove that there is no algorithm that always solves them, no matter how much time or space is allowed.

Outline

Algebra

- δ_{ij} is the Kronecker delta, which is 1 if $i = j$ and 0 otherwise.
- A **field** (CZ pole, komutativní těleso)is a set on which are defined addition, subtraction, multiplication, and division satisfying the field axioms (commutativity, associativity, a unit).
- $\mathbf{1}$ is the vector $(1, 1, 1, \dots)$.
- The **complex conjugate** (CZ komplexně sdružené číslo) of the complex number $z = x + iy$ is defined to by $\bar{z} = z^* = x - iy$.

Matrix [Lay12, GL13]

- $[\dots]_{ij}$ denotes (i,j) element of a matrix
- The **conjugate** of a matrix $\mathbf{A} = (a_{ij}) \in \mathbb{C}^{n\times m}$ is the matrix $\bar{\mathbf{A}} = (\bar{a}_{ij}) \in \mathbb{C}^{n\times m}$.
- The **trace** of an $n \times n$ ("n by n") square matrix $\mathbf{A}$ is

$$\text{Tr}(\mathbf{A}) = \sum_{i=1}^{n} a_{ii} = a_{11} + a_{22} + \cdots + a_{nn} \tag{1}$$

$$\text{Tr}(\mathbf{A}+\mathbf{B}) = \text{Tr}(\mathbf{A}) + \text{Tr}(\mathbf{B}) \tag{2}$$

$$\text{Tr}(c\mathbf{A}) = c\,\text{Tr}(\mathbf{A}) \tag{3}$$

$$\text{Tr}(\mathbf{A}) = \text{Tr}(\mathbf{A}^T) \tag{4}$$

$$\text{Tr}(\mathbf{AB}) = \text{Tr}(\mathbf{BA}) \tag{5}$$

Matrix Transposition [Wat02, Lay12, GL13]

- The **transpose** of a matrix $\mathbf{A} \in \mathbb{R}^{n\times m}$ ($\mathbb{R}^{n\times m} \to \mathbb{R}^{m\times n}$): $[\mathbf{A}^T]_{ij} = [\mathbf{A}]_{ji}$.
- Let $\mathbf{A}$ and $\mathbf{B}$ denote matrices whose sizes are appropriate for the following sums and products, let r denote any scalar, then
  - $(\mathbf{A}^T)^T = \mathbf{A}$
  - $(\mathbf{A}+\mathbf{B})^T = \mathbf{A}^T + \mathbf{B}^T$
  - $(r\mathbf{A})^T = r\mathbf{A}^T$
  - $(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T$
- The **conjugate transpose** of a matrix $\mathbf{A} \in \mathbb{C}^{n\times m}$: $[\mathbf{A}^*]_{ij} = [\bar{\mathbf{A}}]_{ji}$.
- The square matrix $\mathbf{A}$ is **Hermitian** if $\mathbf{A}^* = \mathbf{A} = \mathbf{A}^H$ and **skew-Hermitian** if $\mathbf{A}^* = -\mathbf{A}$.

Orthogonality [Wat02, GL13]

- A set of vectors $\{x_1, \ldots, x_p\}$ in $\mathbb{R}^n$ is **orthogonal** if $x_i^T x_j = 0$ whenever $i \neq j$ and **orthonormal** if $x_i^T x_j = \delta_{ij}$.
- A matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is said to be **orthogonal** if $\mathbf{A}^T \mathbf{A} = \mathbf{I}$.
- A matrix $\mathbf{A} \in \mathbb{C}^{n \times n}$ is said to be **unitary** if $\mathbf{A}^* \mathbf{A} = \mathbf{I}$.

Matrix Inversion [GL13]

- If $\mathbf{A}$ and $\mathbf{X}$ are in $\mathbb{R}^{n\times n}$ and satisfy $\mathbf{AX} = \mathbf{I}$, then $\mathbf{X}$ is the **inverse** of $\mathbf{A}$ and is denoted by $\mathbf{A}^{-1}$.
  - $(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$
  - $(\mathbf{A}^{-1})^T = (\mathbf{A}^T)^{-1} \equiv \mathbf{A}^{-T}$

Matrix Eigenvalues [GL13]

- The **eigenvalues** of $\mathbf{A} \in \mathbb{C}^{n\times n}$ are zeros of the **characteristic polynomial** $p(x) = det(\mathbf{A} - x\mathbf{I})$.
- Every $n \times n$ matrix has n eigenvalues.
- We denote the set of $\mathbf{A}$'s eigenvalues by

$$\lambda(\mathbf{A}) = \{x : det(\mathbf{A} - x\mathbf{I}) = 0\}$$

$$\lambda_{\max}(\mathbf{A}) = \max(\lambda(\mathbf{A})) \qquad \lambda_{\min}(\mathbf{A}) = \min(\lambda(\mathbf{A}))$$

- The **eigenvalue equation** expressed as the matrix multiplication

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}$$

  - Applying the matrix $\mathbf{A}$ to the eigenvector $\mathbf{v}$ only scales the eigenvector by the scalar value λ.
- Symmetry of a matrix $\mathbf{A}$ guarantees that all of its eigenvalues are real and that there is an orthonormal basis of eigenvectors.
- Let $\mathbf{A} \in \mathbb{R}^{n\times n}$ with eigenvalues λ and eigenvectors $\mathbf{v}$. Then $\mathbf{A}^k$ has eigenvalues λ^k and eigenvectors $\mathbf{v}$ for any positive integer k.

Schur Decomposition [GL13]

Theorem 1 (Symmetric Schur Decomposition, Theorem 8.1.1 [GL13], p.440)

If $\mathbf{A} \in \mathbb{R}^{n \times n}$ *is symmetric, then there exists a real orthogonal* $\mathbf{Q}$ *such that*

$$\mathbf{Q}^T \mathbf{A} \mathbf{Q} = \mathbf{\Lambda} = \mathit{diag}(\lambda_1, \ldots, \lambda_n).$$

Moreover, for $k = 1 : n$, $\mathbf{A}\mathbf{Q}(:, k) = \lambda_k \mathbf{Q}(:, k)$.

Theorem 2 (Schur Decomposition, Theorem 7.1.3 [GL13], p.351)

If $\mathbf{A} \in \mathbb{C}^{n \times n}$, *then there exists a unitary* $\mathbf{Q} \in \mathbb{C}^{n \times n}$ *such that*

$$\mathbf{Q}^H \mathbf{A} \mathbf{Q} = \mathbf{T} = \mathbf{\Lambda} + \mathbf{N}$$

where $\mathbf{\Lambda} = \mathit{diag}(\lambda_1, \ldots, \lambda_n)$ *and* $\mathbf{N} \in \mathbb{C}^{n \times n}$ *is strictly upper triangular.*

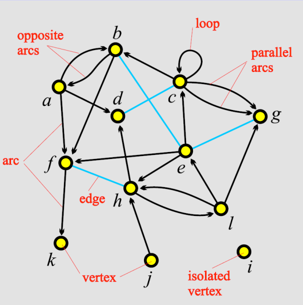
Outline

1 Algorithm and Linear Algebra Basics
- Algorithm Complexity
- Linear Algebra Reminder

2 Graph Terminology
- Graph/Network Definition
- Graph Algorithms

Graph [Weh13]

A **graph** is a set of vertices and a set of lines between pairs of vertices.



- **Actor** - **vertex**, **node**, point
- **Relation** - line, edge, arc, link, tie
  - **Edge** $=$ undirected line, $\{c, d\}$
    c and d are **end** vertices
  - **Arc** $=$ directed line, (a, d)
    a is the **initial** vertex, (source, start)
    d is the **terminal** vertex, (target, end)
  - Parallel (multiple) arcs/edges are only allowed in **multigraphs** with more than one relation (set of lines).
  - **Loop** (self-choice)

We focus on simple graphs!

A **simple** undirected graph has no loops and no parallel edges.
A simple directed graph has no parallel arcs.

Network [EK10, New10, Weh13, Erc15]

Network

A **network** consists of a graph and additional information on the vertices or the lines of the graph.

Formally, a network $\mathcal{N} = (\mathcal{V}, \mathcal{L}, \mathcal{P}, \mathcal{W})$ consists of:

- A graph $\mathcal{G} = (\mathcal{V}, \mathcal{L})$, where
  - $\mathcal{V}$ is the set of vertices,
  - $\mathcal{A}$ is the set of arcs,
  - $\mathcal{E}$ is the set of edges, and
  - $\mathcal{L} = \mathcal{E} \cup \mathcal{A}$ is the set of lines.
- $\mathcal{P}$ vertex value functions / properties: $p : \mathcal{V} \to A$
- $\mathcal{W}$ line value functions / weights: $w : \mathcal{L} \to B$

- **Long range dependencies** vs. multidimensional space
- **Specific topological properties** . . . non-trivial topology
- **Large/Huge volumes** of **sparse** data records

Subgraph [Die05, BM08, Wil98]

- A graph H is a **subgraph** of a graph G, if the following two inclusions are satisfied:
  - $V(H) \subseteq V(G)$
  - $E(H) \subseteq E(G) \cap \binom{V(H)}{2}$
- In other words, a subgraph is created so that:
  - Some vertices of the original graph are removed.
  - All edges incident to the removed vertices and possible some other edges are removed.

Graph Path [Die05, BM08, Wil98]

- A **path** is a non-empty graph $P = (V, E)$ of the form $V = \{v_0, v_1, \dots v_k\}$, $E = \{v_0 v_1, v_1 v_2, \dots v_{k-1} v_k\}$, where the v_i are all distinct.
- The vertices v_0 and v_k are **linked** by P and are called its **ends**, the vertices $v_1, \dots v_{k-1}$ are the **inner** vertices of P.
- A path P can often be identified by its natural sequence of its vertices, i.e. $P = v_0 v_1 \dots v_k$ and called a path **from** v_0 **to** v_k (or **between** v_0 *and* v_k).
- If $P = v_0 \dots v_{k-1}$ is a path and $k \geq 3$, then the graph $C := P + v_{k-1} v_0$ is called a **cycle**.

Graph Walk [Die05, BM08, Wil98]

- A **walk** in a graph G is a sequence $W := v_0 e_1 v_1 \dots v_{\ell-1} e_\ell v_\ell$, whose terms are alternately vertices and edges of G, such that v_{i-1} and v_i are the ends of e_i, $1 \leq i \leq \ell$.
- If $v_0 = x$ and $v_\ell = y$, we say that W **connects** x to y and refer to W as an xy-**walk**.
- The vertices x and y are called the **ends** of the walk, x being its **initial vertex** and y its **terminal vertex**, the vertices $v_1, \dots, v_{\ell-1}$ are its **internal vertices**.
- The integer ℓ (the number of edge terms) is the **length** of W.
- An x-**walk** is a walk with initial vertex x.
- If there is an xy-walk in a graph G, then is also an xy-path.
- The length of a shortest such xy-path is called the **distance** between x and y and denoted $d_G(x, y)$.
- The greatest distance between any two vertices in G is called the **diameter of** G, denoted by $diam(G) = \max_{u,v} d_G(u, v)$.

Graph Component [Die05, BM08, Wil98]

- A non-empty graph G is called **connected** if any two of its vertices are linked by a path in G, otherwise the graph is **disconnected**.
- If $U \subseteq V(G)$ and $G[U]$ is connected, we call U itself connected (in G).
- A maximal connected sugraph of G is called a **component** of G.

Graph Tree [Die05, BM08, Wil98]

- An **acyclic graph** is a graph that does not contain any cycle.
- An acyclic graph is also called a **forest**.
- A connected forest is called a **tree**.
- The vertices of degree 1 in a tree are its **leaves**.
- One vertex of a tree can be selected as special; such a vertex is then called the **root** of this tree.
- A tree T with a fixed root r is a **rooted tree**.
- A **spanning tree** of a graph G is a minimal connected spanning subgraph $T \subset G$

Tree Properties I

Theorem 3 (Theorem 1.5.1 [Die05], p.14)

The following assertions are equivalent for a graph T:

1. T *is a tree;*
2. *Any two vertices of* T *are linked by a unique path in* T*;*
3. T *is minimally connected, i.e.* T *is connected but* $T - e$ *is disconnected for every edge* $e \in T$*;*
4. T *is maximally acyclic, i.e.* T *contains no cycle but* $T + uv$ *does, for any two non-adjacent vertices* $u, v \in T$*.*

Corollary 1 (Corollary 1.5.3 [Die05], p.14)

A connected graph with N *vertices is a tree if and only if it has* $N - 1$ *edges.*

Outline

1. Algorithm and Linear Algebra Basics
   - Algorithm Complexity
   - Linear Algebra Reminder

2. Graph Terminology
   - Graph/Network Definition
   - Graph Algorithms

Tree Search [BM08]

- A systematic procedure, or **algorithm**, that generates a sequence of rooted trees in G, starting with the trivial tree consisting of a single root vertex r, and terminating either with a spanning tree of the graph or with a nonspanning tree whose associated edge cut is empty, is called **tree-search** and the resulting tree is referred to as a **search tree** [BM08].
- **Depth-first search** is a tree-search in which the vertex added to the tree T at each stage is one which is a neighbor of as recent an addition to T as possible.
- The resulting spanning tree is called a **depth-first search tree** or **DFS-tree**.

DFS-tree Search Edge Classification [BM08]

- There are two times associated with each vertex $v \in G$ during the construction of its DFS-tree T:
  - the **discovery time** $\tau_d(v)$ when v is incorporated into T and
  - the **finish time** $\tau_f(v)$ when all the neighbors of v are found to be already in T.
- In particular, $\tau_d(r) = 1$, $\tau_f(v) = \tau_d(v) + 1$ for every leaf v of T, and $\tau_f(r) = 2|V|$.
- Based on Proposition 1 and Theorem 4 any edge $e = uv$ in a graph G having a DFS-tree T with $\tau_d(u) < \tau_d(v) < \tau_f(v) < \tau_f(u)$ can be oriented as $\vec{e} = \overrightarrow{uv} = (u, v)$ and classified as:
  - **tree edge**, if $e \in T$, i.e. the vertex u is an ancestor of v in T,
  - **back edge**, if $e \notin T$.

Tree Search Times - Properties

Proposition 1 (Proposition 6.5 [BM08], p.141)

Let u and v be two vertices of G, with $\tau_d(u) < \tau_d(v)$.

a) *If u and v are adjacent in G, then $\tau_f(v) < \tau_f(u)$.*

b) *u is an ancestor of v in T if and only if $\tau_f(v) < \tau_f(u)$.*

Theorem 4 (Theorem 6.6 [BM08], p.142)

Let T be a DFS-tree of a graph G. Then every edge of G joins vertices which are related in T.

Lemma 1 (Lemma 22.11 [CLRS09], p.614)

A directed graph G is acyclic if and only if a depth-first search of G yields no back edges.

Tree Search Times - Properties

Proposition 2 (Proposition 1.5.6 [Die05], p.16)

Every connected graph contains a normal spanning tree, with any specified vertex as its root.

Breadth-first Search [CLRS09, Erc15]

Algorithm BFS

Input: $G(V, E)$, a source node s
Output: d_v, pred$[v]$, $\forall v \in V$
    ▷ distance and place of a vertex in BFS
Q ... a queue
for all $u \in V \setminus \{s\}$ **do**
    $d_u \leftarrow \infty$
    pred$[u] \leftarrow \perp$ ▷ undetermined value
end for
$d_s \leftarrow 0$
pred$[s] \leftarrow s$

BFS ... the main loop

$Q \leftarrow s$
while $Q \neq \emptyset$ **do**
    $u \leftarrow deque(Q)$
    for all $(u, v) \in E$ **do**
        if $d_v = \infty$ **then**
            $d_v \leftarrow d_u + 1$
            pred$[v] \leftarrow u$
            $enqueu(Q, v)$
        end if
    end for
end while

Theorem 5 (Theorem 3.1 [Erc15], p.35)

The time complexity of BFS algorithm is $\Theta(N + M)$ for a graph of order N and size M.

Depth-first Search [CLRS09, Erc15]

Algorithm DFS_Forest

```
Input: G(V, E), directed or undirected
Output: pred[v], firstVis[v], secVis[v], ∀v ∈ V
int time ← 0; visited[1 : n] ← 0
for all u ∈ V do
    visited[u] ← false
    pred[u] ← ⊥        ▷ undetermined value
end for
for all u ∈ V do
    if ¬visited[u] then
        DFS(u)
    end if
end for
```

DFS procedure

```
procedure DFS(u)
    visited[u] ← true
    time ← time + 1
    firstVis[u] ← time
    for all (u, v) ∈ E do
        if ¬visited[v] then
            pred[v] ← u
            DFS(v)
        end if
    end for
    time ← time + 1
    secVis[u] ← time
end procedure
```

Asymptotic complexity of the DFS algorithm

The time complexity is $\Theta(N + M)$ for a graph of order N and size M.

Dijkstra's Single Source Shortest Paths [CLRS09, Erc15]

Algorithm Dijkstra_SSSP

```
Input: G(V, E), directed or undirected,
Input: positive weights l_e on edges,
Input: a source node s
Output: d_v, pred[v], ∀v ∈ V
for all u ∈ V \ {s} do
    d_u ← ∞
    pred[u] ← ⊥                ▷ undetermined value
end for
d_s ← 0
pred[s] ← s
```

SSSP ...the main loop

```
S ← [V]                        ▷ insert all vertices
while S ≠ ∅ do
    u ← min(S)
    S ← S \ {u}
    for all (u, v) ∈ E do
        if d_v > d_u + l(u, v) then
            d_v ← d_u + l(u, v)
            pred[v] ← u
        end if
    end for
end while
```

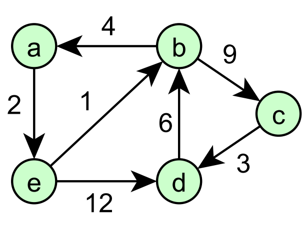
Theorem 6 (Theorem 5.1 [Erc15], p.84)

The time complexity of the Dijkstra's_SSSP is $O(N^2)$ for a graph of order N.

Floyd-Warshall All Pairs Shortest Paths [CLRS09, Erc15]

- The approach
  - Dynamic programming approach
  - Comparing all possible paths between each pair of nodes in G
  - Improving the shortest path between them at each step until the result is optimal.
- Distance matrix $D[N, N]$ between nodes u and v
- Matrix $P[N, N]$ with the first node on the current shortest path from u to v

Example 1



FW APSP Algorithm [CLRS09, Erc15]

Algorithm FW_APSP

Input: $G(V, E)$,
Input: weights w_e on edges,
no negative-weight cycles
Output: $D[N, N]$, $P[N, N]$
for all $\{u, v\} \in V$ **do**
    if $u = v$ **then**
        $D[u, v] \leftarrow 0$; $P[u, v] \leftarrow \perp$
    else if $(u, v) \in E$ **then**
        $D[u, v] \leftarrow w_{uv}$; $P[u, v] \leftarrow v$
    else
        $D[u, v] \leftarrow \infty$; $P[u, v] \leftarrow \perp$
    end if
end for

APSP . . . the main loop

$S \leftarrow \emptyset$
while $S \neq V$ **do**
    pick w from $V \setminus S$ ▷ Select a pivot
    for all $u \in V$ **do**
        for all $v \in V$ **do**
            if $D[u, w] + D[w, v] < D[u, v]$ **then**
                $D[u, v] \leftarrow D[u, w] + D[w, v]$
                $P[u, v] \leftarrow P[u, w]$
            end if
        end for
    end for
    $S \leftarrow S \cup \{w\}$
end while

Asymptotic complexity of the FW_APSP algorithm

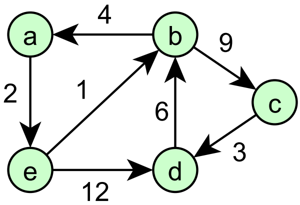
The time complexity is $\Theta(N^3)$ for a graph of order N.

FW APSP Algorithm Example [Erc15]

$$D = \begin{bmatrix} 0 & \infty & \infty & \infty & 2 \\ 4 & 0 & 9 & \infty & \infty \\ \infty & \infty & 0 & 3 & \infty \\ \infty & 6 & \infty & 0 & \infty \\ \infty & 1 & \infty & 12 & 0 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} 0 & 3 & \infty & 14 & 2 \\ 4 & 0 & 9 & 12 & 6 \\ \infty & 9 & 0 & 3 & \infty \\ 10 & 6 & 15 & 0 & \infty \\ 5 & 1 & 10 & 12 & 0 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} 0 & 3 & 12 & 14 & 2 \\ 4 & 0 & 9 & 12 & 6 \\ 13 & 9 & 0 & 3 & 10 \\ 10 & 6 & 15 & 0 & 12 \\ 5 & 1 & 10 & 12 & 0 \end{bmatrix}$$



Summary

- Graph Terminology Reminder
- Graph Path Algorithms Reminder

Competencies

- Define a complex network and its basic features.
- Define asymptotic bounds used for assessment of algorithm complexity.
- Describe DFS-tree search edge classification.
- Describe depth-first search algorithm.
- Describe breath-first search algorithm.
- Describe the Dijkstra's single source shortest paths.
- Describe the Floyd-Warshall all pairs shortest paths.

Appendix

References I

[BM08] J.A. Bondy and U.S.R. Murty. *Graph Theory*. Springer, 2008.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[Die05] Reinhard Diestel. *Graph Theory*. Springer, 2005.

[EK10] David Easley and Jon Kleinberg. *Networks, Crowds, and Markets. Reasoning About a Highly Connected World*. Cambridge University Press, July 2010.

[Erc15] Kayhan Erciyes. *Complex Networks, An Algorithmic Perspective*. CRC Press, 2015.

[GL13] Gene H. Golub and Charles F. Van Loan. *Marix computations*. The Johns Hopkins University Press, Baltimore, fourth edition, 2013.

[Lay12] David C. Lay. *Linear Algebra and Its Applications*. Addison-Wesley, fourth edition, 2012.

[New10] M. Newman. *Networks: an introduction*. Oxford University Press, Inc., 2010.

[Wat02] David S. Watkins. *Fundamentals of Matrix Computations*. Second edition, 2002.

[Weh13] Stefan Wehrli. Social network analysis, lecture notes, December 2013.

[Wil98] Robin J. Wilson. *Introduction to Graph Theory*. Longman, fourth edition, 1998.