Machine Learning for Complex Networks

Prof. Dr. Ingo Scholtes

Chair of Machine Learning for Complex Networks
Center for Artificial Intelligence and Data Science (CAIDAS)
Julius-Maximilians-Universität Würzburg
Würzburg, Germany

ingo.scholtes@uni-wuerzburg.de

Lecture 02
Graph-Theoretic and Algorithmic Foundations

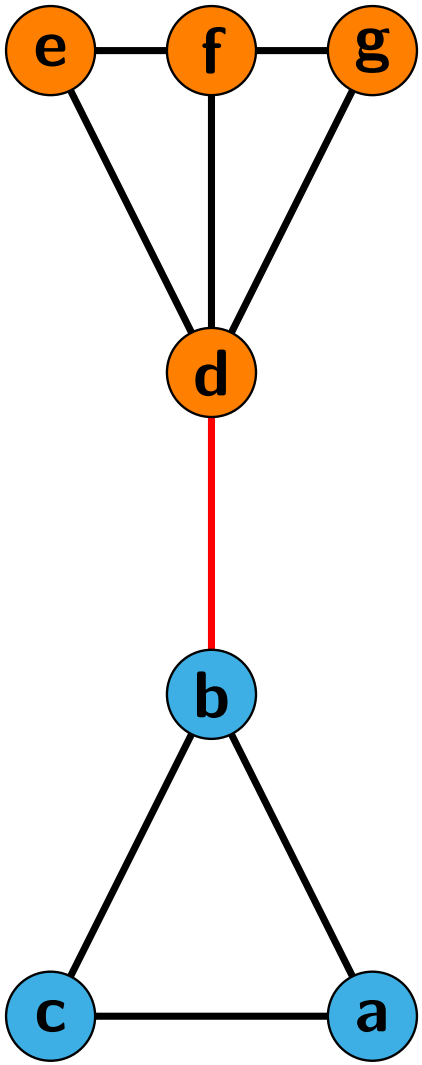
May 4, 2022

Notes:

- **Lecture L02: Graph-Theoretic and Algorithmic Foundations** 04.05.2022
- **Educational objective:** We show how we can mathematically represent graphs and networks. We introduce basic graph-theoretic concepts and show how we can use the graph Laplacian to detect communities in networks.
  - Graphs, networks, adjacency matrix
  - Paths and connected components
  - Communities, Minimal Cuts and Connectivity
  - Graph Laplacians and Spectral Clustering
- **Exercise sheet 01:** Connected Components and Cluster Detection due 11.05.2022

Motivation

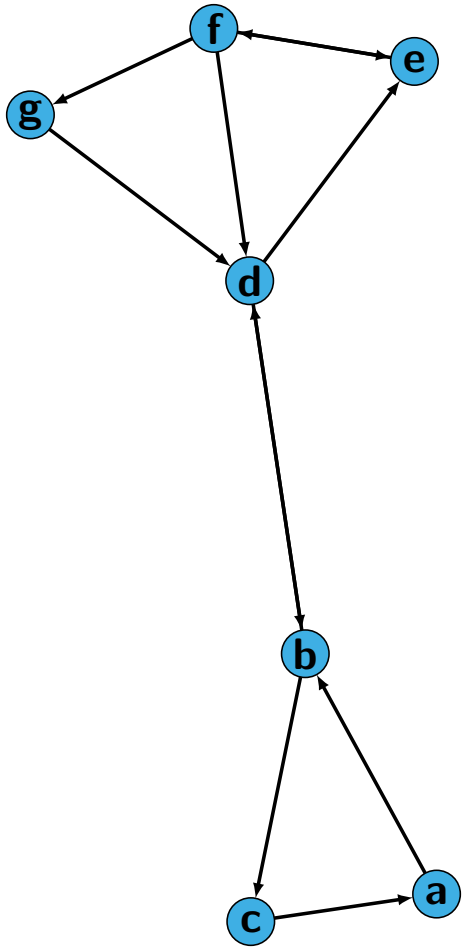
- to address **machine learning in graph-structured data** we need a **common mathematical language**
- we recap fundamental **graph-theoretic concepts** and **graph algorithms**
- we introduce the unsupervised learning task of **community detection in networks**
- we show how it can be addressed based on **graph Laplacians**



Notes:

- Before we can begin to address machine learning tasks in complex networks, we first need a common mathematical language for fundamental concepts in graphs and networks.
- In today's lecture we thus introduce fundamental definitions and concepts of graph theory, such as weighted, directed, and undirected networks, the adjacency matrix, paths, walks, cycles node degrees, or connected components.
- We then turn our attention to a first unsupervised machine learning task in graphs: the community detection problem, which seeks to detect groups of "well-connected" nodes in a network.
- We will briefly discuss different approaches to define and detect communities in networks. Taking a graph-theoretic perspective, we then introduce cuts and connectivity in networks and show how we can use the graph Laplacian to detect communities in networks.

What is a network?



graph or network

A graph or **network** is a tuple $G = (V, E)$ where

- V is a set of vertices or **nodes**
- $E \subseteq V \times V$ is a set of edges or **links**

$V \times V$ denotes the Cartesian product of the node set, i.e. the set of all possible links $(i, j) \in V \times V$.

- we say: link (i, j) points **from** node i **to** j
- if not defined otherwise $n := |V|$ $m := |E|$
- **multigraphs** can have multiple links between the same nodes, i.e. E is **multiset**

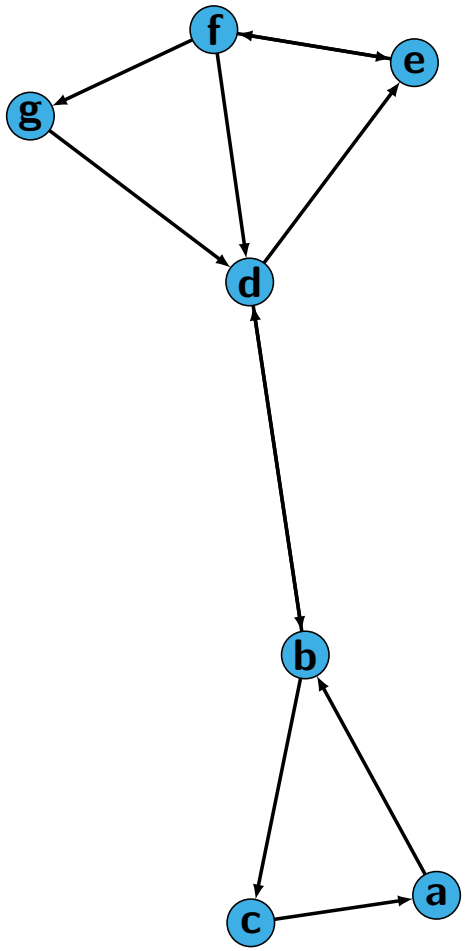
example network

$$V = \{a, b, c, d, e, f, g\}$$
$$E = \{(a, b), (b, c), (b, d), (c, a), (d, b), (d, e), (e, f), (f, d), (f, e), (f, g), (g, d)\}$$

Notes:

- In this course, we use the terms **vertex** and **node**, as well as **edge** and **link** or **graph** and **network** interchangeably. This is common in the interdisciplinary network science community.
- The tuples (v, w) in the set of links E are ordered, i.e. $(v, w) \neq (w, v)$ if $v \neq w$. This allows to distinguish links that have different **directionality**. We denote a link by a tuple (i, j), referring to a link that points from i to j. You sometimes find other conventions, where (i, j) refers to an edge pointing from j to i. For undirected networks → slide 4 this does not make a difference, but it is important to clarify the notation for directed networks. Hence, we consistently use a notation where links point from the left to the right element in a tuple.
- The nodes i and i that are the endpoints of an edge (i, j) are called **adjacent** (from Latin "adiacere" for "border upon" or "lie near"). A link (i, j) is said to be **incident** on nodes i and i (from Latin "incidere" for "to fall upon").
- If not defined otherwise, we often use n to refer to the number of nodes and m to refer to the number of links. The number of different, ordered tuples between sets with n nodes is n^2, i.e. a network with n nodes can have at most n^2 links.
- We can also define **multigraphs where E is a multiset of links**, i.e. elements in E can occur multiple times. Consequently the maximal number of links is unbounded. In this course we generally do not consider multigraphs, i.e. E is a set where elements can occur only once. For data where links are observed multiple times we can instead assign numeric edge attributes.

Adjacency matrix



- **adjacency matrix** $\mathbf{A} \in \{0,1\}^{n \times n}$ of network $G = (V, E)$ is a matrix with

$$A_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{else} \end{cases}$$

where A_{ij} refers to **row i and column j**

adjacency matrix of directed network

$$\mathbf{A} = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \end{array} \begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

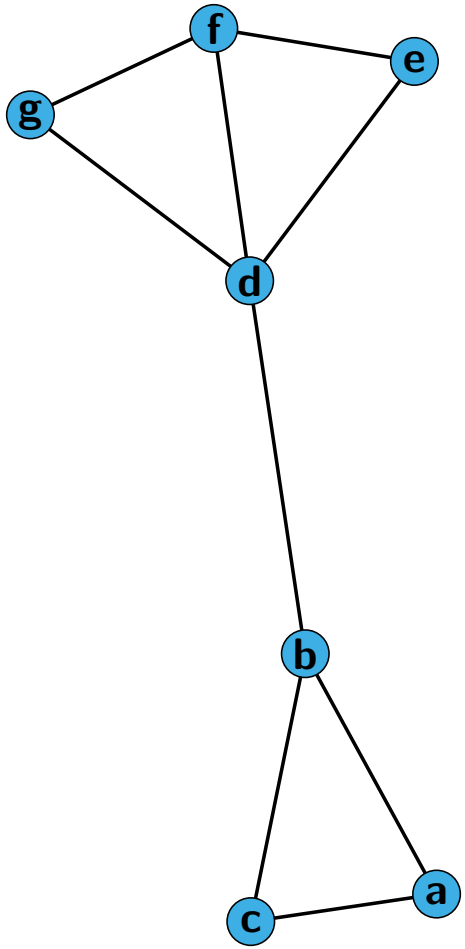
- for directed networks (no self-loops)

$$|E| = m = \sum_{i,j \in V} A_{ij}$$

Notes:

- Binary, square **adjacency matrices** $\mathbf{A} \in \{0,1\}^{n \times n}$ are a simple and widely used mathematical structure to mathematically represent networks. The existence of a link (i, j) from i to j (i.e. an "adjacency") is indicated by an entry $A_{ij} \in \{0,1\}$ in row i and column j, where 1 captures that the link is present while 0 indicates the absence of the link.
- In the example above, the adjacency matrix is not symmetric. This is due to the fact that links have a *direction*. For example, the link (a, b) exists, but the reverse link (b, a) does not exist. We call networks with this property *directed networks*. An example for a network that is naturally directed is a *citation* network. An article A that cites an article B does not imply that the opposite is true. In fact, except for rare cases where manuscripts were written (and published) at the same time, this cannot even happen.
- In practice, the adjacency matrices of many empirical networks are **sparse matrices**, i.e. there are many more 0 elements than 1 elements. This facilitates compressed representations, where only non-zero elements are actually stored.
- For the binary adjacency matrix of directed networks with no self-loops → slide 5 the sum of matrix elements corresponds to the number of links in the network. The **outgoing links** of node i are represented in **row** i of the matrix. The **incoming links** of node j are represented in **column** j of the matrix.

Undirected networks



- network is **undirected** iff

$$(i,j) \in E \Leftrightarrow (j,i) \in E$$

  and **directed** otherwise

- adjacency matrices of undirected networks are **symmetric**, i.e. $A_{ij} = A_{ji} \forall i,j \in V$

adjacency matrix of undirected network

$$\mathbf{A} = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \end{array} \begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

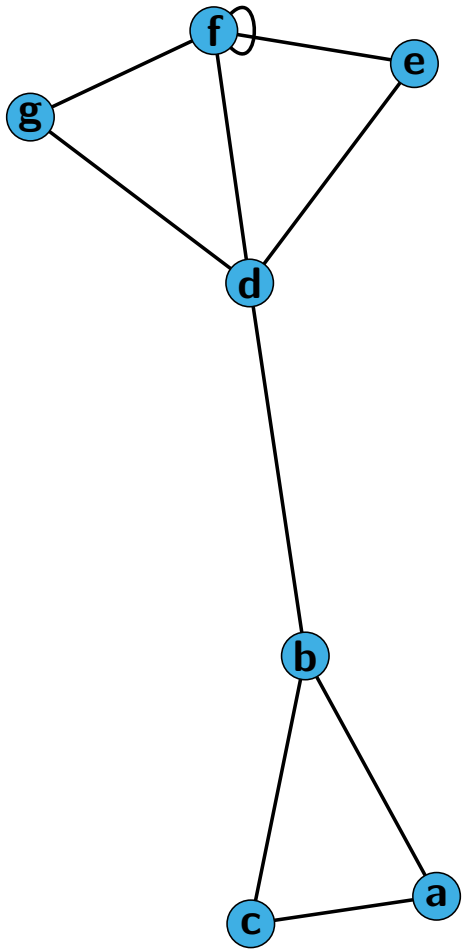
- for undirected networks (no self-loops)

$$|E| = m = \frac{1}{2} \sum_{i,j \in V} A_{ij}$$

Notes:

- We say that a network is **undirected** iff all links exist in both directions, i.e. $(i,j) \in E \Leftrightarrow (j,i) \in E$. For binary adjacency matrices of undirected networks we have $A_{ij} = A_{ji}$, i.e. the **adjacency matrix is symmetric**. We sometimes do not explicitly differentiate between an undirected network and a directed network in which each link exists in both directions (the adjacency matrix representation of both are identical). However, we do distinguish between directed and undirected networks regarding the question what a single undirected link is. In the example above, we say that we have an undirected network with nine undirected links, rather than counting 18 directed links. This has the implication that the number of undirected links is only half of the sum of adjacency matrix entries in an undirected network.
- In the **graphical representation of undirected networks** we use a single undirected link (with no arrow heads) instead of two directed links (x,y) and (y,x). Sometimes, we also use mixed representations for directed networks, where an undirected link is a simpler notation for two directed links between the same node pair in opposite directions.
- Many **collaboration networks** are naturally *undirected*. If two employees A and B work together on a project, A is linked to B and B is linked to A, i.e. collaborations are *symmetric*.
- Most **citation networks** are naturally *directed*. Except for exceptional cases, the fact that article A cites article B even means that the opposite link *cannot* exist (since the directionality of links typically implies that A was published before B).

Self-loops



- links (i, i) are called **self-loops**
- captured in the **diagonal entries** of adjacency matrix **A**
- two different representations of self-loops:
  1. $(i, i) \in E \implies A_{ii} = 1$
  2. $(i, i) \in E \implies A_{ii} = 2$

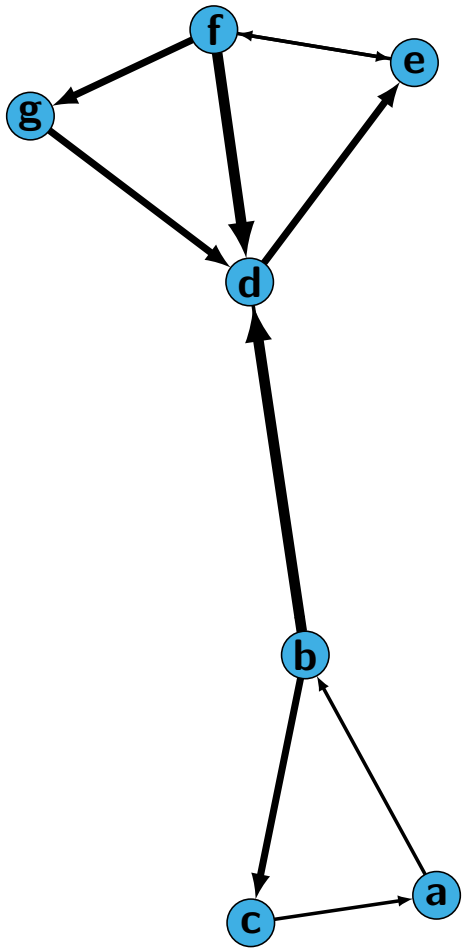
adjacency matrix of network with self-loop

$$\mathbf{A} = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \end{array} \begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & \mathbf{2} & 1 \\ 0 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

Notes:

- In the previous slide we mentioned so-called **self-loops**, which refers to links $(i, i) \in E$ from node i to the node i itself. Such self-loops are captured in the **diagonal of the adjacency matrix**.
- Different from links (i, j) for $i \neq j$ self-loops can only exist in one direction, which translates to the fact that even in directed networks there is only a single matrix entry for each self-loop. This is the reason why, in a network with self-loops, we cannot simply double the sum of matrix entries to calculate the number of edges. To account for this special characteristic, we sometimes define non-binary adjacency matrices of unweighted network, where a self-loop is represented by a 2 on the diagonal.
- Self-loops have special characteristics that can lead to complications in the definition of some network-analytic measures. We thus often exclude them when we analyse networks. Sometimes, we do however consider (or even explicitly add) them, e.g. in the modelling of dynamical processes. Here self-loops represent **node-internal feedback** or memory, i.e. they encode that the future state of a node is coupled to the previous state of that same node.

Weighted networks



- in **weighted** networks links have **numerical attributes** $w : E \rightarrow \mathbb{R}$ that capture strength, frequency, capacity, etc. of links
- weighted networks have a **real-valued adjacency matrix** $\mathbf{A} \in \mathbb{R}^{n \times n}$ with

$$A_{ij} = \begin{cases} w(i,j) \text{ if } (i,j) \in E \\ 0 \text{ else} \end{cases}$$

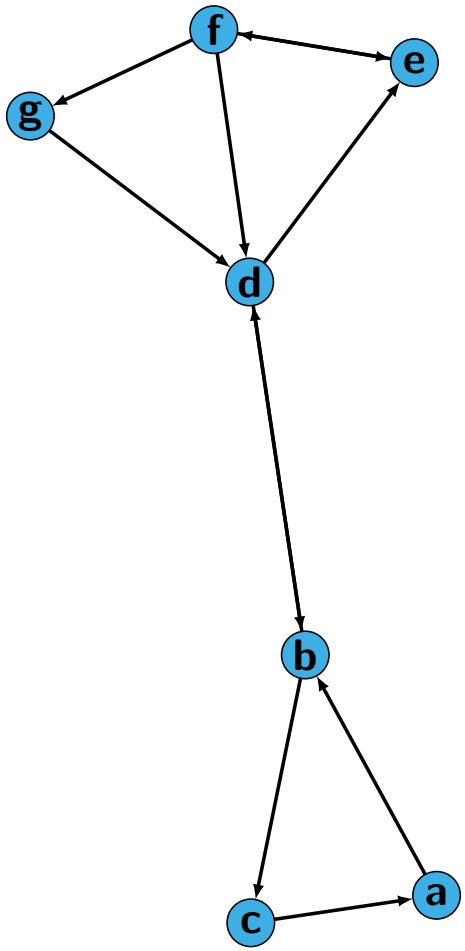
adjacency matrix of weighted network

$$\mathbf{A} = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \end{array} \begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 3 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 3 & 1 & 0 & 2 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

Notes:

- In many networks, we want to capture additional numerical properties of links, e.g. the strength, capacity, cost or frequency of an interaction or connection. For such settings weighted networks, each edge has an additional real- or integer-valued property, the so-called **link weight**. Examples for properties captured by link weights include
  - the frequency or duration of contacts between actors in social networks
  - the level of trust between actors in a social network
  - the number of co-authored papers in a co-authorship network
  - the bandwidth of a network connection in a communication network
  - the number of passengers travelling on a route between two airports
  - the average cost of flights between two airports
  - the geographical distance between two stations in a train network
  - the capacity of a transmission line in a power grid
  - the trade volume in a network of financial transactions between economic actors
- We can mathematically represent a weighted network by real-valued adjacency matrices, in which non-zero entries capture the weights of links.
- Note that also networks in which all links exist in both directions (which would qualify as an undirected network) can have asymmetric adjacency matrices if the weights of links in different directions differ.

Node degrees



undirected networks

- **degree** $d(i) = d_i$ of node i is defined as

$$d_i := |\{j \in V : (i,j) \in E\}|$$

directed networks

- **indegree** $d_{in}(i)$ is the number of incoming edges, i.e. $d_{in}(i) := |\{j \in V : (j,i) \in E\}|$
- **outdegree** $d_{out}(i)$ is the number of outgoing edges, i.e. $d_{out}(i) := |\{j \in V : (i,j) \in E\}|$

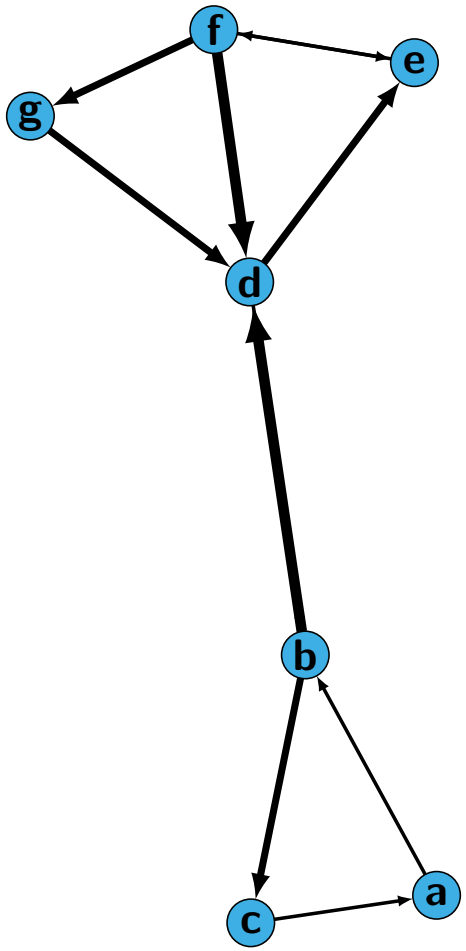
example network

- $d_{in}(f) = 1$
- $d_{out}(f) = 3$

Notes:

- The **degree of a node** i corresponds to the number of nodes to which it is directly connected. In directed networks we distinguish between **indegree and outdegree**. The indegree of i counts the number of predecessors, i.e. the number of nodes j for which a link (j, i) exists. The outdegree of i counts the number of sucessors, i.e. the number of nodes j for which a link (i, j) exists.
- For an undirected networks, we have $d_{in}(i) = d_{out}(i) = d_i$ and we simply call this the degree of a node.
- Sometimes, for directed networks a **total degree** $d_{total}(i)$ is defined as $d_{total}(i) = d_{in}(i) + d_{out}(i)$, i.e. the total degree counts both incoming and outgoing links.
- We can easily calculate degrees in directed and undirected networks by **summing the rows/columns of their adjacency matrix**. In directed networks, the outdegree of node i is the sum of entries in row i, i.e. $d_{out}(i) = \sum_j A_{ij}$ where index j runs over the columns. The indegree of node j is the sum of entries in column i, i.e. $d_{in}(j) = \sum_i A_{ij}$, where index i runs over the rows. In undirected networks both yields the same value since the adjacency matrix is symmetric, i.e. we can compute the degree in either way.
- The **degree sequence (or distribution)** of a network is an macroscopic feature of networks, which allows us to make surprisingly strong statements about the expected properties of a network. → more in Statistical Network Analysis

Weighted node degree



weighted degrees

for weighted networks, the **weighted in- or outdegree** of a node is the sum of incoming or outgoing link weights, i.e.

$$w_{in}(i) := \sum_{j \in V} w(j,i) = \sum_{j=1}^{n} A_{ji}$$

$$w_{out}(i) := \sum_{j \in V} w(i,j) = \sum_{j=1}^{n} A_{ij}$$

adjacency matrix of weighted network

$$\mathbf{A} = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \end{array} \begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 3 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 3 & 1 & 0 & 2 \\ 0 & 0 & 0 & 2 & 0 & 0 & 0 \end{bmatrix} \end{array}$$

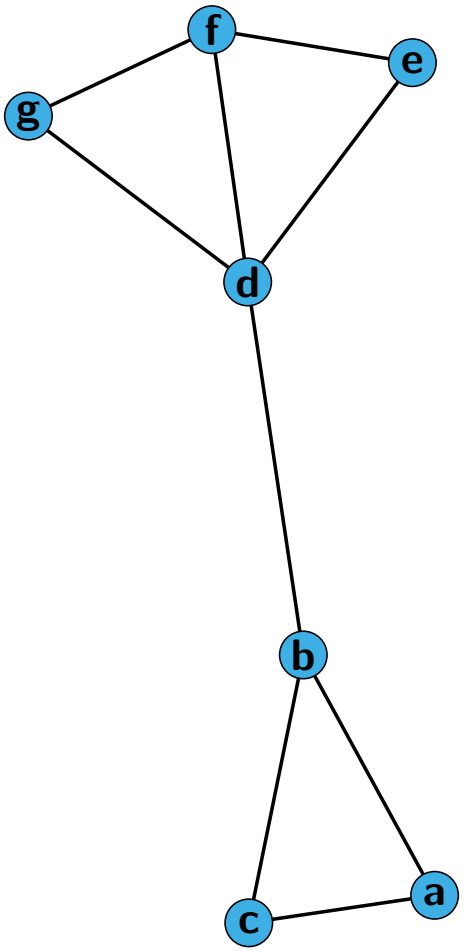
$$w_{in}(i) = \sum_{j \in V} A_{ji}, \; w_{out}(i) = \sum_{j \in V} A_{ij}$$

Notes:

- We can extend the definition of node degrees to weighted networks by summing the weights of incoming or outgoing links.
- For a binary adjacency matrix, the **weighted in-degree** of a node i is the sum of entries in column i of the adjacency matrix. Conversely, the **weighted out-degree** of node i is the sum of entries in row i of the adjacency matrix. In undirected networks with symmetric adjacency matrices, bot are the same and we call this the **weighted degree**.
- Sometimes, the **strength or total weighted degree** of a node i in a weighted and directed network is defined as the sum $w_{in}(i) + w_{out}(i)$ of the weighted in- and outdegree. However, for weighted directed networks it is more common to consider the weighted in- and out-degree separately.

Visualizing networks

- we need **graphical representation**, i.e. Euclidean representation of graph
- but: networks can capture arbitrary **non-Euclidean topologies**
- need to **map nodes to positions in Euclidean space** $\mathbb{R}^d$ with $d \in \{1, 2, 3\}$
- we call $L : V \to \mathbb{R}^d$ **layout of graph**
- good graph layouts enable us to **follow paths and recognize patterns**



example: Fruchterman-Reingold algorithm

compute stable state of multi-body simulation with

- **repulsive** force between all pairs of nodes
- **attractive force** between all nodes connected by an edge

→ TMJ Fruchterman, EM Reingold, 1991

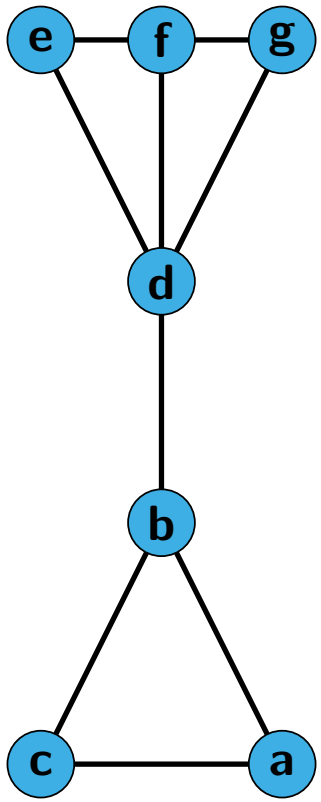
layouts used in graph drawing

- Circular layout
- Force-directed layouts
- Spectral layout → L08

Notes:

- The origin of the term *graph* is the Greek work *graphos* (to draw), i.e. an essential feature of a graph or network is that we can draw them (on paper or on a screen). We can always draw nodes as circles and connect pairs of nodes connected by an edge via a line. However, there are infinitely many different ways in which we can draw a graph or network. We are thus interested in principled methods to find a good – or even an in some way optimal – drawing of a network.
- In principle, to visualize a network, we need geometric representations of nodes and edges. However, graphs can capture arbitrary non-Euclidean topologies so mapping them to a one, two, or three-dimensional Euclidean space for the purpose of visualization is actually challenging.
- We are generally interested in mappings of nodes to coordinates such that the mapping retains as much "information" about the graph topology as possible (cf. node embedding techniques in machine learning, more in our future lecture *Machine Learning for Complex Networks*). In particular, a good graph layout should help us to easily follow paths along sequences of edges, and recognize clusters of nodes that are connected by many edges. For this, those nodes should be positioned close to each other. Hence we can say that for good network visualizations the notion of "similarity" captured in terms of edges between nodes should be reflected by the Euclidean distance between the geometric representations of nodes.
- There are many different algorithms that produce meaningful visual representations of networks, i.e. force-directed layouts like the Fruchterman-Reingold algorithm.

Walks, paths and cycles



- sequence $(p_0, p_1, \ldots, p_l)$ of nodes $p_i \in V$ is a **walk** from p_0 to p_l iff

$$(p_i, p_{i+1}) \in E \text{ for } i = 0, \ldots, l-1$$

- walk $(p_0, p_1, \ldots, p_l)$ is a **(simple) path** iff

$$p_i \neq p_j \text{ for } 0 \leq i, j \leq l \text{ and } i \neq j$$

- walk $(p_0, p_1, \ldots, p_l)$ is a **cycle** iff
  1. $p_0 = p_l$
  2. $p_i \neq p_j$ for $0 < i, j < l$ and $i \neq j$

example

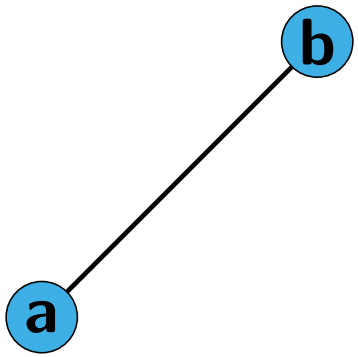
- (a, b, a, b, d) is a walk
- (a, b, c, a) is a cycle
- (a, b, d, g) is path of length three from a to g
- (a, b) and (a, c, b) are edge-independent

- **length of path, walk, or cycle** is defined as $\text{len}(p_0, \ldots, p_l) := l$
- two paths are **edge-independent** iff they do not have an edge in common

Notes:

- Arguably, the main (if not only) reason why we are interested in networks is because they allow us to understand how the elements of a complex system can directly and **indirectly** influence each other via sequences of links. A sequence of nodes where any two consecutive nodes are adjacent is called a **walk**. Walks can contain the same node multiple times like, e.g., as in the example below.
- A walk $(p_0, \ldots, p_l)$ where all nodes are different is called a **path** from node p_0 to node p_l. The terms "walk" and "path" are often used synonymously, in which case we call a path where all nodes are different **simple path**.
- A walk where only the start point p_0 and the endpoint p_l are identical is called a **cycle**. A network that contains at least one cycle is called a **cyclic network**. If a network contains no cycle we call it **acyclic network**. We call two paths that do not share a common edge **edge-independent** or (sometimes) **disjoint**.
- We define the **length of a walk, path or cycle** as the number of traversed links (i.e. the number of traversed nodes minus one). Hence, a single edge $(i, j) \in E$ (with $i \neq j$) defines a path of length one that connects node i to node j. In communication networks, this definition of path lengths has a natural interpretation as the number of *hops* a message takes from the origin to the destination.
- For any two nodes, there can be **many different paths** by which the same pair of nodes is indirectly connected. In the example network, there is only a single path of exactly length three from node a to node g, but there is another path of length four that first traverses node c. This shows that a path between two nodes does not necessarily need to follow the shortest sequence of links.

Powers of adjacency matrices



- consider a binary adjacency matrix of a network $G = (V, E)$ with $V = \{a, b\}$ and

$$\mathbf{A} =: \mathbf{A}^1 = \begin{bmatrix} \delta_{aa} & \delta_{ab} \\ \delta_{ba} & \delta_{bb} \end{bmatrix}$$

with $\delta_{ij} = 1$ if $(i, j) \in E$ and 0 otherwise

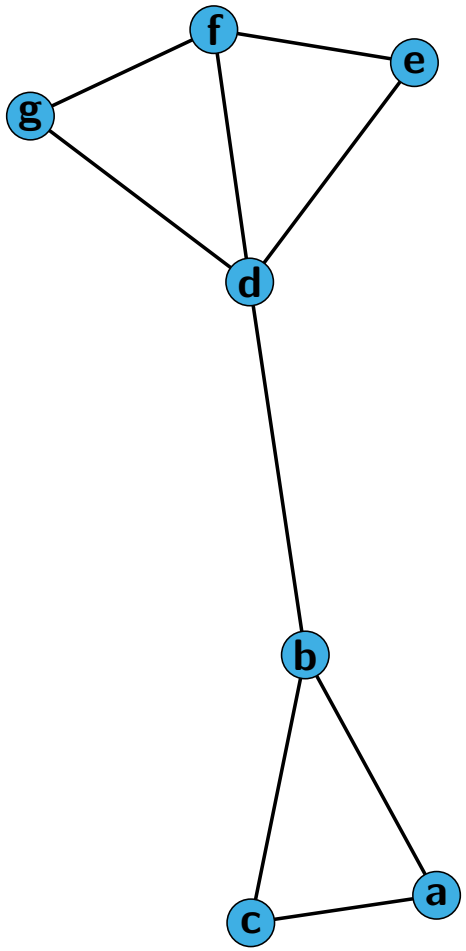
- let us **multiply the adjacency matrix** of G **with itself**

$$\mathbf{A}^2 = \begin{bmatrix} \delta_{aa} & \delta_{ab} \\ \delta_{ba} & \delta_{bb} \end{bmatrix} \cdot \begin{bmatrix} \delta_{aa} & \delta_{ab} \\ \delta_{ba} & \delta_{bb} \end{bmatrix} = \begin{bmatrix} \delta_{aa}\delta_{aa} + \delta_{ab}\delta_{ba} & \delta_{aa}\delta_{ab} + \delta_{ab}\delta_{bb} \\ \delta_{ba}\delta_{aa} + \delta_{bb}\delta_{ba} & \delta_{ba}\delta_{ab} + \delta_{bb}\delta_{bb} \end{bmatrix}$$

Notes:

- We introduce an important aspects in the representation of networks in terms of adjacency matrices. A key feature of this mathematical representation is that the multiplication of adjacency matrices naturally relates to the (transitive) notion of walks (or paths) in a network. To better understand this, let us consider an adjacency matrix of a maximally simple network with two nodes a and b. An example for such a network is shown above, but here we do not care about a specific topology. Let us assume that the entries δ_{ab} of the adjacency matrix capture whether an edge from a to b exists in the network, i.e. δ_{ab} is an indicator of the corresponding edge.
- Let us now multiply this adjacency matrix with itself. We apply the rules of matrix multiplication and study the entries of the resulting matrix $\mathbf{A}^2$. We find that those entries count the number of walks of exactly length two between all pairs of nodes, i.e. they are zero if no walk of length two exists and non-zero if one or two such walks exist.
- Example for the top left element: the sum captures the existence of walk $(a, a) \to (a, a)$ + the existence of walk $(a, b) \to (b, a)$
- example for the top right element: the sum captures the existence of walk $(a, a) \to (a, b)$ + the existence of walk $(a, b) \to (b, b)$

Powers of adjacency matrices



$$\mathbf{A}^3 = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \end{array} \begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\ \begin{bmatrix} 2 & 4 & 3 & 1 & 1 & 1 & 1 \\ 4 & 2 & 4 & 6 & 1 & 2 & 1 \\ 3 & 4 & 2 & 1 & 1 & 1 & 1 \\ 1 & 6 & 1 & 4 & 6 & 6 & 6 \\ 1 & 1 & 1 & 6 & 2 & 5 & 2 \\ 1 & 2 & 1 & 6 & 5 & 4 & 5 \\ 1 & 1 & 1 & 6 & 2 & 5 & 2 \end{bmatrix} \end{array}$$

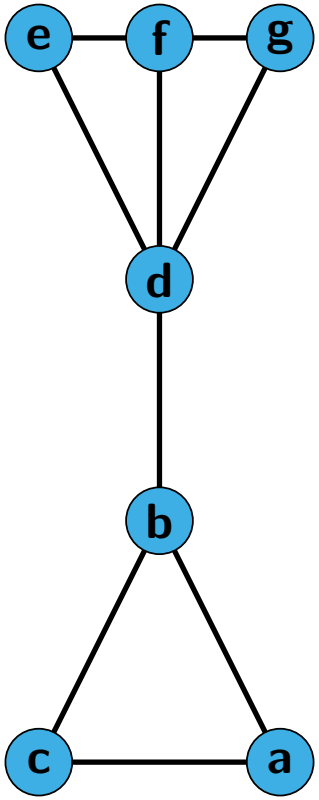
interpretation of $\mathbf{A}^k$

- for undirected networks we have $A^2_{ii} = d_i$
- entries A^k_{ij} of k-th power of adjacency matrix count **different walks of exactly length k** between node i and node j

Notes:

- We can test this in our example network from before: Here we have two different walks of length two which start in node f and end in node d. The first one is (f, g, d), the second one is (f, e, d). There are three different cycles of length two which start in node b and end in node b. The first one is the (b, a, b), the second one is (b, c, b), and the third one is (b, d, b).
- For any undirected network without self-loops, the diagonal entries of the squared adjacency matrix $\mathbf{A}^2$ contain the degrees of the corresponding nodes, i.e. $A^2_{ii} = d_i$ This is because
    1. in such a network each undirected link of i yields exactly one cycle of length two from i to i, and
    2. there cannot be other paths of length two that start in i and end in i
- By multiplying the adjacency matrix with itself once more, we now add one to the length of the walks that are counted. Hence, the entries of the matrix $\mathbf{A}^k$ count the walks of exactly length k. Consider the entry $A_{fg} = 5$ in the example of $\mathbf{A}^3$ above: the walks of lengths three between f and g are (f, e, d, g), (f, e, f, g), (f, d, f, g), (f, g, f, g), (f, g, d, g).
- From this, we see that the standard adjacency matrix $\mathbf{A} = \mathbf{A}^1$ is simply a special case that counts the number of walks/paths of length one (which are simply links).
- Looking at the entries of matrix $\mathbf{A}^3$, what else can we say about the topology of our example network?

Topological distance



example

- $\text{dist}(a, d) = 2$
- shortest path: (a, b, d)
- $\text{diam}(G) = 3$

- **distance** $\text{dist}(v, w)$ between nodes v and w is the minimum length of any path between v and w
- $\text{dist}(v, w) := \infty \Leftrightarrow \nexists$ path from v to w
- path $(p_0, \ldots, p_l)$ is **shortest path** iff $\text{len}(p_0, \ldots, p_l) = \text{dist}(p_0, p_l)$
- for weighted network $(p_0, \ldots, p_l)$ is **cheapest path** iff $\sum_{i=1}^{l} w(p_{i-1}, p_i)$ is minimal
- **diameter** $\text{diam}(G)$ of network $G = (V, E)$ is length of the longest shortest path

$$\text{diam}(G) := \max_{v,w \in V} \text{dist}(v, w)$$

Notes:

- Networks define a discrete topological space in which we can calculate a measure of **topological distance** between any pair of nodes. The topological distance between a node v and a node w is the minimal length of any path that connects them. We call the distance between the nodes the shortest path length and each path with that length that connects those nodes is called a **shortest path**. Note that this distance only fulfils the symmetry property of a metric if the network is undirected.
- In the example above, there is only a single path of length two from node a to node d and this path is the shortest path, so the distance between those two nodes is two. The shortest path is not necessarily unique, i.e. different shortest paths of the same length can exist for a given pair of nodes. We will see that the distribution of shortest path lengths is an important macroscopic characteristic of complex networks.
- In many systems (e.g. communication networks) *finding shortest paths* is a key problem, which must be solved in order to provide *routing* services. Indeed, the main task of routing algorithms on the Internet is to identify optimal communication paths between computer networks. As a first approximation, *best* can be thought of as *shortest* but we can also include link costs represented as link weights in a weighted network. Here, we can extend the definition of shortest path to **cheapest path**, i.e. paths where the sum of edge weights from v to w is minimal.
- The **diameter** is a simple but important **systemic or collective property** of complex networks. Why do we call this a systemic or collective feature? Because it results from the global topology of the network. This is particularly true for large networks, where changing a single node or link is likely to not change the diameter.

Finding shortest paths

Dijstra's algorithm → EW Dijkstra, 1959

- **single-source shortest paths** for graphs with **positive edge weights**
- repeatedly relax path length for neighbors of first node in **priority queue**
- worst-case time complexity $\mathcal{O}(m + n \cdot \log n)$

Bellman-Ford algorithm → R Bellman, 1958

- **single-source shortest paths** for graphs with **real edge weights** (no negative cycles)
- repeatedly relax path length of node v for all edges (v, w)
- worst-case complexity $\mathcal{O}(n \cdot m)$

Floyd-Warshall algorithm → RW Floyd, 1962

- **all-pairs shortest paths** for graphs with **real edge weights** (no negative cycles)
- test triangle inequality for all triples of nodes
- worst-case complexity $\mathcal{O}(n^3)$

Richard E. Bellman
1920 – 1984

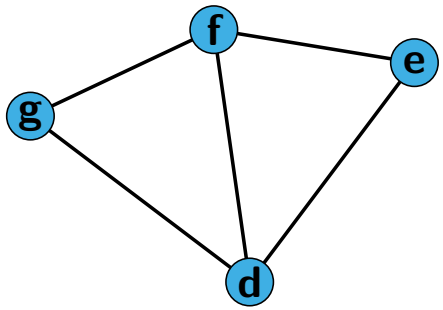
John von Neumann Theory Prize 1976

image credit: Wikipedia, Fair use

Notes:

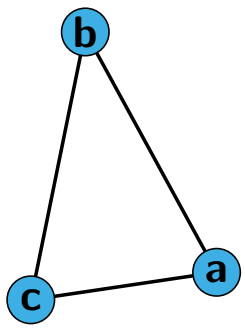
- A number of algorithms have been proposed to compute (i) shortest paths from one node to all other nodes (single-source problem), (ii) between all node pairs (all-pairs problem), or (iii) cheapest paths in networks with positive/negative weights. I assume that you studied those algorithms in depth in your BSc courses and we will thus not discuss them in detail here.
- In the practice session, we provide implementations of the three key algorithms mentioned above and we demonstrate how we can use them to calculate and reconstruct shortest and cheapest paths using `pathpy` and how to calculate diameter and average shortest path length.

Connected components



- undirected network $G = (V, E)$ is **connected** if $\text{dist}(v, w) < \infty$ for all $v, w \in V$
- **connected components** of $G = (V, E)$ are maximally connected subgraphs $G' = (V', E')$ with $V' \subseteq V$ and $E' \subseteq E$
- size of connected component $G' = (V', E')$ is $|V'|$
- largest connected component G' is called **giant connected component** iff

$$\frac{|V'|}{|V|} \approx 1$$



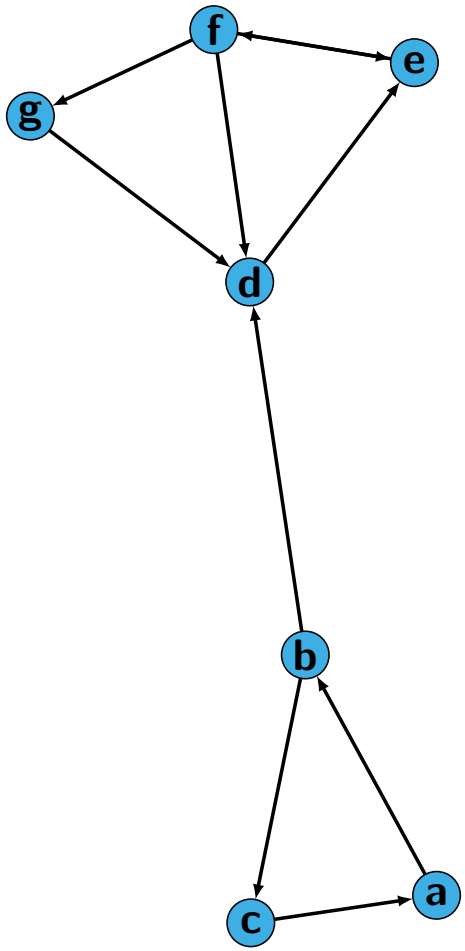
example

- connected component $\{a, b, c\}$
- largest connected component $\{d, e, f, g\}$

Notes:

- One of the most important characteristics of a network is whether all nodes can actually influence each other, i.e. whether all nodes are connected via a path. If this is the case (i.e. when all distances are finite) we say that the **network is connected**.
- For networks which are not connected, we are often interested in the **connected components**, i.e. a partition into the largest subsets of nodes for which all pairs of nodes are connected by a path.
- A **largest connected component** of a network is any connected component that contains a maximum number of nodes. We say that the largest connected component is a **giant connected component** if it contains almost all of the nodes (i.e. it is much larger than the second-largest component).
- The exact definition of a giant connected component depends on the context: For theoretical studies of random graph models with a variable number of nodes n we often call the largest connected component G' a giant connected component if $\frac{|V'|}{|V|} \to 1$ for $n \to \infty$. → see Statistical Network Analysis

Connected components in directed graphs



- we distinguish between **strongly and weakly connected** directed networks
- directed network is **weakly connected** iff corresponding undirected network is connected
- directed network $G = (V, E)$ is **strongly connected** iff
  $dist(v, w) < \infty \quad \forall v, w \in V$
- **strongly connected components** of $G = (V, E)$ are maximal strongly connected subgraphs $G' = (V', E')$ with $V' \subseteq V$ and $E' \subseteq E$

example

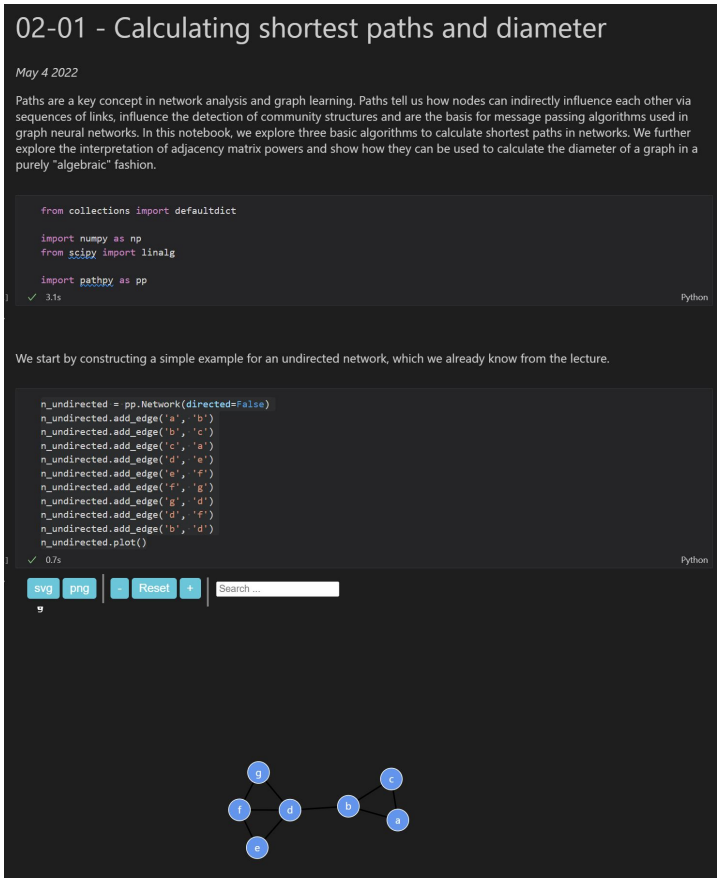
- weakly but not strongly connected
- strongly connected components $\{a, b, c\}, \{d, e, f, g\}$

Notes:

- The definitions of connectedness and connected components in undirected networks can be extended to directed networks in a natural way. In an undirected network, we can traverse any path $(p_0, \ldots, p_l)$ in both directions, which implies that for any pair of nodes we have $dist(v, w) = dist(w, v)$. A trivial consequence is that $dist(v, w) < \infty \Leftrightarrow dist(w, v) < \infty$.
- For directed networks a path $p_0, \ldots, p_l$ may not be a path if we reverse the order of nodes. Hence, we generally have $dist(v, w) \neq dist(w, v)$ and we can have the $dist(v, w) < \infty$ while $dist(w, v) = \infty$, i.e. v is connected to w via path, but w is not connected to v via a path. For directed networks, we thus distinguish between **strongly and weakly connected networks**.
- A directed network is called **weakly connected** if the undirected network that we obtain by replacing every directed link $(v, w) \in E$ with a corresponding undirected link is connected. In a connected network, for each pair of nodes v, w we have that there is either a path from v to w or a path from w to v (or both).
- A directed network is called **strongly connected** if all pairs of nodes are connected by paths, i.e. for every pair $v, w \in V$ we have $dist(v, w) < \infty$. This is the directed equivalent of a connected undirected network, where all nodes are connected to each other via a path. Every strongly connected network is necessarily weakly connected.
- Similar as in undirected networks, we can define the **strongly connected components** of a directed network as the maximal strongly connected subgraphs.

Practice session

- we show how to construct **directed, undirected and weighted networks** with `pathpy`
- we explain how to **visualize networks** with `pathpy`
- we implement three key **algorithms to calculate shortest paths**
- we use Tarjan's algorithm to **calculate maximally connected subgraphs**



practice session

see notebooks 02-01 – 02-02 in `gitlab` repository at

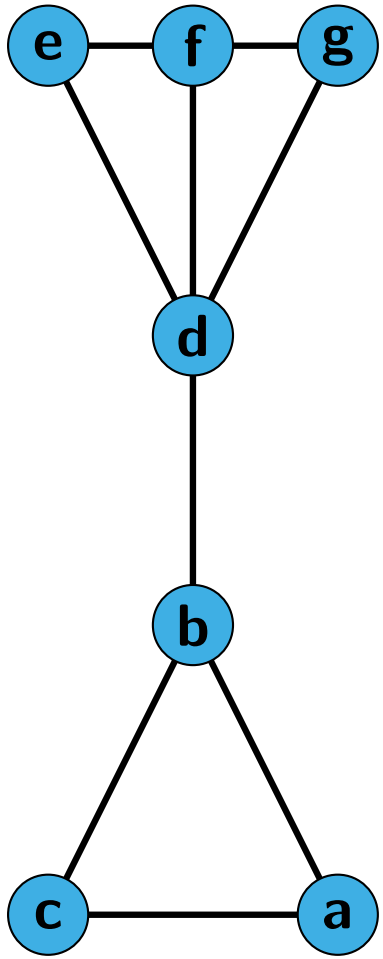
→ `https://gitlab.informatik.uni-wuerzburg.de/ml4nets_notebooks/2022_sose_ml4nets_notebooks`

Notes:

- Now that we have covered some foundations of graph theory, we move to the first practice session of our course. In the practice session, we study practical demonstrations of the theoretical concept introduced in the lecture.
- In this first session, we will show you how directed, undirected, and weighted networks are represented in the network analysis package `pathpy`.
- We implement and explore three basic shortest path algorithms in `pathpy`, and use Tarjan's algorithm to calculate connected components in directed and undirected networks.
- We further show how we can use adjacency matrix powers to calculate the number of walks between nodes and discuss algebraic approachs to calculate diameter and connected components.
- We further show how you can visualize such networks in a jupyter notebook.
- You can find the jupyter notebooks (and data) used in the practice sessions in an accompanying gitLab repository.

From components to communities ...

- connected component = subset $C \subseteq V$ of nodes **exclusively connected** to other nodes in C
- yields (trivial) definition of **clusters** in networks
- we need a "soft" definition that **allow (few) links between clusters**



communities in graphs

a subset $C \subset V$ of nodes is called **community** if nodes in C are "more strongly connected" to other nodes in C than to nodes not in C.

- generalizes definition of **clusters in Euclidean space** to graph-shaped data

Notes:

- We could see the connected components of a graph as a trivial type of a **cluster**, where the nodes within the component are connected to each other via paths, while nodes in different components are not connected via a path. This would correspond to a similarity-based definition of node clusters, where nodes are considered “similar” iff a path exists between them.
- In the exercise, you will see that we can indeed use connected components to detect clusters in Euclidean data. However, the definition of clusters based on components is very restrictive. To be detected as separated clusters, there cannot be a single link between components. This calls for a relaxation of the cluster definition in graphs that has been studied extensively in graph learning.
- Intuitively, we can generalize density- or similarity-based definitions of clusters to graphs, i.e. we consider a set of nodes a cluster if the nodes within the cluster are – in some way – “more strongly connected” to each other than to nodes in other clusters.
- This “soft” definition of clusters or communities in graphs is the basis for important problems in social network analysis, recommender systems, links prediction, etc.

The many facets of community detection ...

- there is **no single notion of communities** in complex networks
- existing methods can be broadly categorized into **four different approaches**

different approaches to community detection

- cluster/modularity perspective

  → Statistical Network Analysis (L02), L08/L10
- stochastic approaches → L03 – L05
- dynamical processes/random walks → L06
- cut-based methods → today

- we start with **graph-theoretic perspective** that is based on **minimization of cut size**

The many facets of community detection in complex networks

Michael T. Schaub,[1,2,3,*] Jean-Charles Delvenne,[2,4] Martin Rosvall,[5,†] and Renaud Lambiotte[3,‡]

[1] *Institute for Data, Systems, and Society, Massachusetts Institute of Technology, Cambridge, MA 02139, USA*
[2] *ICTEAM, Université catholique de Louvain, B-1348 Louvain-la-Neuve, Belgium*
[3] *naXys and Department of Mathematics, University of Namur, B-5000 Namur, Belgium*
[4] *CORE, Université catholique de Louvain, B-1348 Louvain-la-Neuve, Belgium*[§]
[5] *Integrated Science Lab, Department of Physics, Umeå University, SE-901 87 Umeå, Sweden*

Community detection, the decomposition of a graph into essential building blocks, has been a core research topic in network science over the past years. Since a precise notion of what constitutes a community has remained evasive, community detection algorithms have often been compared on benchmark graphs with a particular form of assortative community structure and classified based on the mathematical techniques they employ. However, this comparison can be misleading because apparent similarities in their mathematical machinery can disguise different goals and reasons for why we want to employ community detection in the first place. Here we provide a focused review of these different motivations that underpin community detection. This problem-driven classification is useful in applied network science, where it is important to select an appropriate algorithm for the given purpose. Moreover, highlighting the different facets of community detection also delineates the many lines of research and points out open directions and avenues for future research.

Keywords: community detection, graph partitioning, Modularity; block models

I. INTRODUCTION

Sparked by the work of Newman and Girvan [1, 2] on Modularity in Complex Systems, the area of community detection has become one of the main pillars of network science research. The promise that we can gain a deeper understanding of a system by discerning important structural patterns within a network has spurred a huge number of studies in this area. However, as has become abundantly clear by now, this problem has no canonical solution. In fact, even a general definition of what constitutes a community is still lacking. The reasons for this are not only grounded in the computational difficulties of tackling community detection. Furthermore, various research areas view community detection from different perspectives, which the lack of a consistent terminology illustrates: 'network clustering', 'graph partitioning', 'community', 'block' or 'module detection' all carry slightly different connotations. This jargon barrier creates confusion as soon as readers and authors have different preconceptions and intuitive notions are not made explicit.

We argue that community detection should not be considered as a well-defined problem, but rather as an umbrella term with many facets. These facets emerge from different goals and motivations of what it is about the network that we want to understand or achieve, which lead to different perspectives on how to formulate the problem of community detection. Therefore, it is critical to be aware of these underlying motivations when selecting and comparing community detection methods. Thus, rather than an in-depth discussion of the technical details of different algorithmic implementations [3–10], here we focus on the conceptual differences between different perspectives on community detection.

By providing a problem-driven classification, however, we do *not* argue that the different perspectives are unrelated. In fact, in some situations, different mathematical problem formulations can lead to similar algorithms and methods, and the different perspectives can offer valuable insights. For example, for undirected networks, optimizing the objective function Modularity [1], initially proposed from a clustering perspective, can be interpreted both as optimizing a particular stochastic block model [11] and a particular diffusion process on the networks [12], In other situations, however, such relations are not apparent.

Neither do we argue that there is a particular perspective that is *a priori* better suited for any given network. In fact, no method can consistently perform best on all kinds of networks [13]. Community detection is an unsupervised learning task and we cannot know what are the quantities of interest for the analysis. Instead, to understand how useful a particular method is, we must take into account the context of why the researcher is interested in the communities [14].

In the following, we unfold different aims underpinning community detection and discuss how the resulting problem perspectives relate to various applications. We focus on four broad perspectives that have served as motivation for community detection in the literature: (*i*) community detection as minimization of some form of constraint violation; (*ii*) community detection framed as a discretised analogue of data clustering, in which densely knit groups of nodes are to be found; (*iii*) community detection aiming to identify structurally equivalent nodes in a network, leading to notions such as stochastic block models; and (*iv*) community detection looking for simplified descrip-

* mschaub@mit.edu; corresponding author
† martin.rosvall@umu.se
‡ renaud.lambiotte@unamur.be
§ jean-charles.delvenne@uclouvain.be

→ M Schaub, JC Delvenne, M Rosvall, R Lambiotte, 2017

Notes:

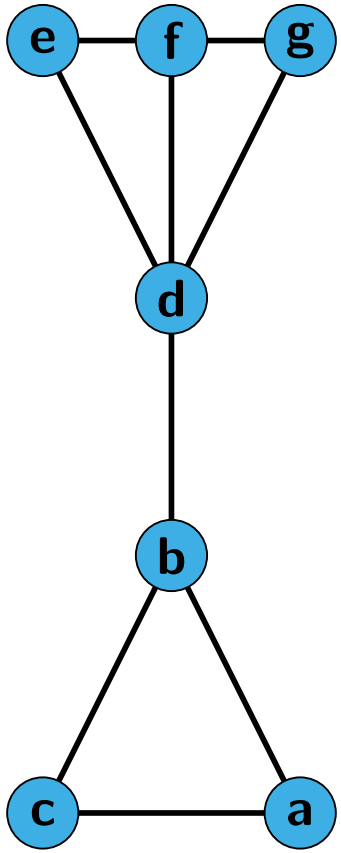
- How can we formalize the notion of communities as groups of nodes that are “more strongly connected” to each other than to other nodes?
- There is no single answer to this question. Indeed, many different definitions or notions of communities in complex networks have been explored in the network science and graph learning communities. This has given rise to different types of methods that are, at times, unfairly compared to each other. A discussion of these different types of methods, and their advantages or limitations, can be found in the perspective article mentioned abovce.
- According to the categorization introduced in this article, a first type of methods are those building on a cluster- or modularity perspective. The modularity maximization approach discussed in our course *Statistical Network* falls into this category. We will not repeat this here, but if you are interested you can check → Statistical Network Analysis: L03 .
- A second type of methods are based on a cut-based definition of communities, which we will introduce in the remainder of this lecture.
- A third and fourth category of methods use stochastic approaches or models of dynamical processes, like random walks. We will cover those in subsequent lectures.

Vertex cuts and vertex connectivity

- consider undirected network $G = (V, E)$
- **vertex cut** is subset $C \subseteq V$ such that $G = (V - C, E)$ is disconnected
- number of nodes $|C|$ in vertex cut C is called **vertex cut size**
- **vertex connectivity** C_{vertex} of a graph G is minimal vertex cut size, i.e.

$$C_{\text{vertex}} := \min\{|C| : C \subseteq V \text{ vertex cut for } G\}$$

- for graph with n nodes vertex connectivity $C_{\text{vertex}} \in \{0, 1, \ldots, n-1\}$
- **vertex connectivity generalizes binary definition of connectedness**



undirected graph with two minimal vertex cuts $C_1 = \{b\}$ and $C_2 = \{d\}$

$$C_{\text{vertex}} = 1$$

Notes:

- We note that – so far – we have considered a **binary notion of connectivity** in undirected graphs, i.e. a graph is either connected (if at least one path exists that connects each pair of nodes) or it is not connected (if we can find at least one pair of nodes not connected by a path). One approach to community detection is to generalize this notion of connectivity to a numerical value, which allows us to formalize how "well-connected" a network is.
- For this, consider an undirected network. A vertex cut is a subset of nodes that cuts the network, i.e. we are interested in a set of nodes that – if we were to remove this set of nodes and all incident edges – leaves the remaining network disconnected. We call the number of nodes in such a vertex cut the size of this cut.
- We now define the **vertex connectivity of a network** as the size of the smallest vertex cut, i.e. the minimal number of nodes that we must remove to separate remaining nodes into at least two connected components.
- We note that the vertex cut size of a disconnected network is zero, while the vertex cut size of a connected network is at least one. Connected networks can have vertex connectivities larger than one. For a network in which each node is connected to all other nodes (including itself), we can, for instance, remove $n - 1$ nodes and the remaining network (consisting of a single node with a self-loop) is still connected.
- Intuitively, the presence of community structures in a network is related to the fact that we can find a small number of nodes whose removal will split the network into multiple disconnected components. In the example above, removing either node b or d will lead to two connected components.

Menger's theorem

- vertex connectivity of a graph is related to number of edge-independent paths

Menger's theorem

in a finite graph, the **minimum size of a vertex cut** is equal to the **maximum number of edge-independent paths** between any pair of vertices.

- small vertex connectivity indicates **"bottleneck" in topology**, i.e. nodes connected by small number of edge-independent paths
- formalizes definition of communities as **groups of "well-connected" nodes**

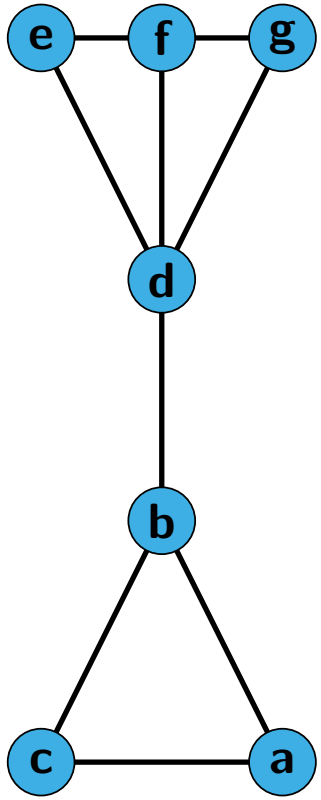
Karl Menger
1902 – 1985

image credit: Shimer College, Public domain

Notes:

- Using Menger's theorem, which relates the minimal size of vertex cuts with the number of disjoint paths in a graph, we can take a different perspective that links community structures in networks to so-called "bottlenecks" in the topology, i.e. a small set of edges through which paths between communities must pass.
- Menger's theorem states that the minimal size of a vertex cut (and thus the vertex connectivity of a network) is equal to the maximum number of edge-independent paths between any pair of vertices. In other words, if we can find a pair of vertices for which all paths must pass through a specific edge, the vertex connectivity of the network is one. In this case, we can remove any of the adjacent nodes to cut the network into two disconnected components.
- We thus find that a small vertex connectivity indicates a "bottleneck" in the topology of a network, i.e. the presence of nodes that are connected only by a small number of edge-independent paths. This formalizes the definition of communities as groups of well-connected nodes and is an important concept in the analysis of robustness in networks.
- Note that Menger's theorem is the basis for the **max-flow-min-cut theorem**, which relates size of a minimal cut to the maximum flow in a graph.

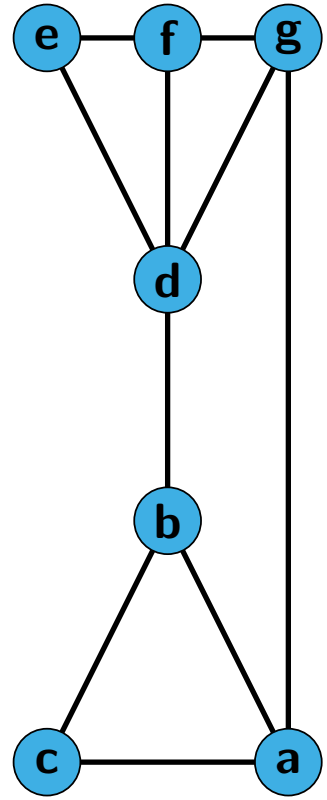
Example: vertex connectivity and paths



$C_{\text{vertex}} = 1$

nodes a and g connected by **single edge-independent path**

$\{(a, b, d, g)\}$



$C_{\text{vertex}} = 2$

nodes a and g connected by **two edge-independent paths**

$\{(a, b, d, g), (a, g)\}$

Notes:

- In the two example networks above, we explore the concept of **vertex connectivity** and its relation to **edge-independent paths** as stated by Menger's theorem.
- In the **left example**, a removal of either node b or node c will leave the remaining network disconnected. We thus have a vertex connectivity of one, i.e. in a sense this is a network that is not "well-connected". The vertex connectivity of one is equal to the number of edge-independent paths between, for example, nodes a and g. Here we can only find a single path (a, b, d, g). Any other path from a to g must necessarily pass through edge (b, d), which means it is not edge-independent.
- In the **right example**, we have added another edge to the network that connects nodes in the two "communities". This edge increases the vertex connectivity to two. Now removing either node b or d is not sufficient to disconnect the network. We must, for instance, remove the two nodes b and a or d and g. We now find that the nodes a and g are connected by two edge-independent paths that either traverse the edge (b, d) or (a, g).

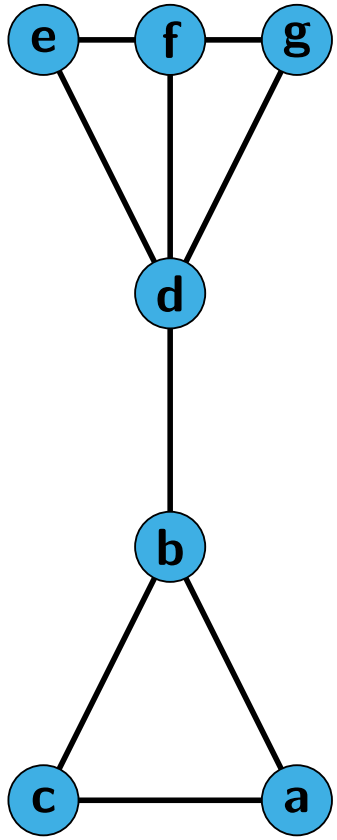
Edge cuts and normalized cut size

- **edge cut** is a partition of nodes into two disjoint subsets $C_1, C_2 \subset V$ with $C_1 \cup C_2 = V$ and $C_1 \cap C_2 = \emptyset$
- we call number of edges crossing C_1 and C_2 **size of edge cut**, i.e.

$$s(C_1, C_2) = |\{(v, w) \in E : v \in C_1, w \in C_2\}|$$

- **normalized cut size** is given as

$$N_s(C_1, C_2) = s(C_1, C_2) \left(\frac{1}{\sum_{v \in C_1} d_v} + \frac{1}{\sum_{v \in C_2} d_v} \right)$$



network with minimal edge cut

$C_1 = \{a, b, c\}, C_2 = \{d, e, f, g\}$

$N_s(C_1, C_2) = 1 \cdot \left(\frac{1}{7} + \frac{1}{11}\right) \approx 0.234$

- computation of edge cuts with minimal normalized size is **NP-hard**

→ Shi and Malik, 2000

Notes:

- Apart from vertex cuts, we can also consider how the connectivity of a network is changed if we **remove links from the network**. For this, consider a partition of the nodes in a network into two disjoint sets. We now assume that we remove those edges that cross the two sets. We call this an **edge cut**, which can be thought of a process to generate two connected components, each of them consisting of the nodes in one of the sets as well as the induced edges, i.e. we only keep edges between pairs of nodes that are in the same set.
- We call the number of edges that cross those sets the **size of an edge cut**.
- We can define a **normalized edge cut size**, by multiplying the size of the edge cut with the sum of the reciprocal node degree sums in the two components. This yields a "cost" of the cut in terms of the number of cut edges relative to the total number of edges in the two resulting partitions.
- To find communities, we are interested in edge cuts with minimal normalized cut size. Unfortunately, finding such cuts is an NP-hard problem. However, we can use heuristic methods to find edge cuts that approximate the minimal cut size.

The Laplacian Matrix

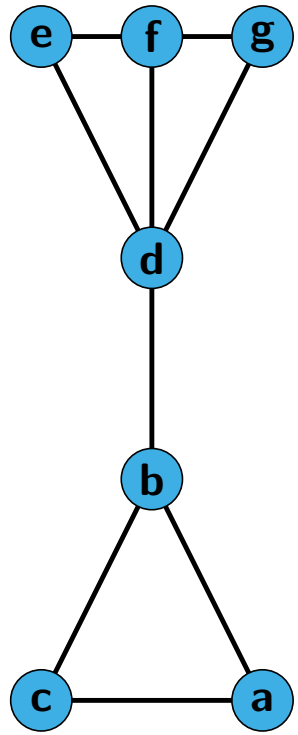
- for **undirected network** consider **Laplacian matrix**

$$\mathcal{L} := \mathbf{D} - \mathbf{A}$$

where $\mathbf{D}$ is degree-diagonal matrix, i.e. $D_{ii} := d_i$

- rows and columns of graph Laplacian sum to zero
- graph Laplacian is **discrete generalization of Laplacian operator in Euclidean space** to arbitrary topologies

→ Statistical Network Analysis, L12



$$\mathcal{L} = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \end{array} \begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\ \begin{bmatrix} 2 & -1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & -1 & 0 & 0 & 0 \\ -1 & -1 & 2 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 4 & -1 & -1 & -1 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & -1 & 3 & -1 \\ 0 & 0 & 0 & -1 & 0 & -1 & 2 \end{bmatrix} \end{array}$$

Notes:

- We can heuristically find edge cuts with minimal normalized cut size based on the so-called **Laplacian matrix** or **graph Laplacian**, which is defined as the matrix

$$\mathcal{L} = \mathbf{A} - \mathbf{D}$$

- We note that $\mathbf{D}$ denotes a diagonal matrix, where the diagonal entries capture the degrees of nodes. This implies that the rows and columns of the Laplacian matrix sum to zero.
- The Laplacian matrix captures the topology of a network. As formally shown in our course *Statistical Network Analysis* it can be viewed as a **discrete generalisation of the Laplacian operator in Euclidean space to arbitrary (discrete) interaction topologies**.
- The Laplacian matrix is just another matrix representation of an (undirected and unweighted) network. Generalizations for weighted and directed cases exist but are unfortunately more complicated to work with. → F Chung, 2005
- The Laplacian matrix is also sometimes called “Kirchhoff matrix”. This is due to the fact that the problem of diffusion in networks is intimately related to electrical circuits, which motivated Gustav Kirchhoff to Kirchhoff’s matrix tree theorem in which the Laplacian matrix appears naturally.

Algebraic Connectivity

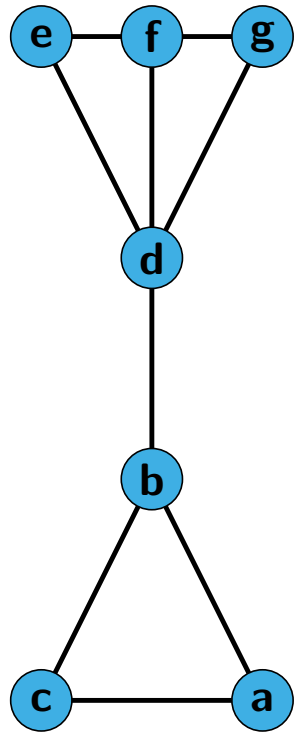
- consider eigenvalues $\lambda_1 \leq \lambda_2 \leq \ldots \leq \lambda_n$ of Laplacian $\mathcal{L}$ in **ascending order**
- $(1, \ldots, 1) \cdot \mathcal{L} = \vec{0}$ so $\vec{1}$ is eigenvector for **smallest eigenvalue** $\lambda_1 = 0$

algebraic connectivity

we call second-smallest eigenvalue λ_2 of the graph Laplacian **algebraic connectivity**

- algebraic connectivity is **lower bound for vertex connectivity**

$$\lambda_2 \leq C_{\text{vertex}}$$



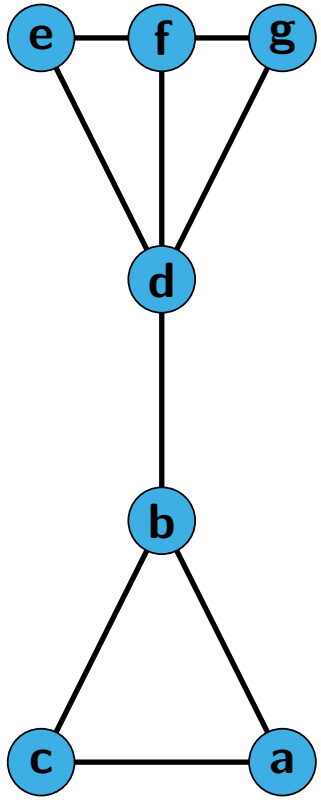
$$\mathcal{L} = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \end{array} \begin{array}{c} \begin{array}{ccccccc} a & b & c & d & e & f & g \end{array} \\ \begin{bmatrix} 2 & -1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 3 & -1 & -1 & 0 & 0 & 0 \\ -1 & -1 & 2 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 4 & -1 & -1 & -1 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 \\ 0 & 0 & 0 & -1 & -1 & 3 & -1 \\ 0 & 0 & 0 & -1 & 0 & -1 & 2 \end{bmatrix} \end{array}$$

$$\lambda_2 = 0.398$$

Notes:

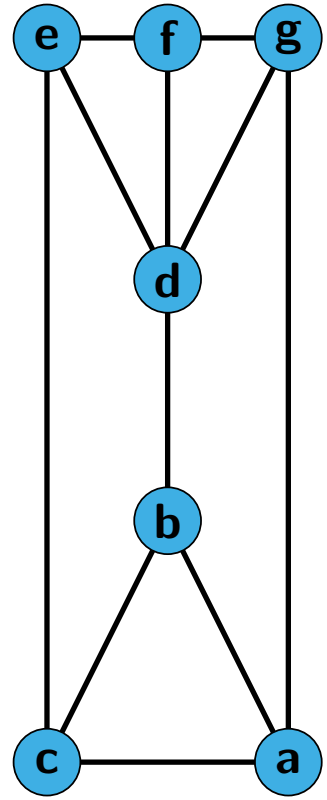
- We have defined **vertex connectivity** as a numerical generalization of binary connectivity in a network, i.e. it captures not only if but also "how" connected a network is. A related numerical generalization of connectivity can be defined based on the eigenvalues of the Laplacian matrix. For this, consider the sequence of eigenvalues in ascending order. We first note that, due to the fact that the rows and sums of the Laplacian matrix sum to one, the vector that contains only ones is an eigenvector of the Laplacian matrix with an associated eigenvalue of zero.
- The first non-trivial eigenvalue $\lambda_2 \geq 0$ in the sequence of ascending Laplacian eigenvalues is called **algebraic connectivity**. In the exercise, you will see that it generalized the concept of connectivity in a network, i.e. $\lambda_2 > 0$ iff the network is connected. It can be shown that the algebraic connectivity is bounded above by the vertex connectivity of a network.

Example: algebraic connectivity



$C_{\text{vertex}} = 1$

$\lambda_2 \approx 0.398 < 1$



$C_{\text{vertex}} = 3$

$\lambda_2 \approx 1.596 < 3$

algebraic connectivity can be used to assess **how "well-connected"** a network is, i.e. **whether topology contains "bottlenecks"**

Notes:

- Let us consider the relationship between vertex connectivity and algebraic connectivity in two simple examples.
- We have already considered the example on the left, which has a vertex connectivity of one. We find that the algebraic connectivity is 0.398, a rather small value smaller than one, which indicates that this network is connected but not very well-connected. In other words, the small value of algebraic connectivity tells us that there is a bottleneck in the topology.
- If we add two edges to mitigate this bottleneck, we increase both the algebraic connectivity and the vertex connectivity. We now have a vertex connectivity of three amd an algebraic connectivity of 1.596. This shows that we have improved the connectivity in the network compared to the example on the left.

Fiedler vector

- consider sequence of **eigenvectors**

$$\vec{v_1}, \vec{v_2}, \ldots, \vec{v_n}$$

corresponding to eigenvalues $\lambda_1 \leq \ldots \leq \lambda_n$ of $\mathcal{L}$ in ascending order

- due to $(1, \ldots, 1) \cdot \mathcal{L} = \vec{0}$ we have eigenvector $\vec{v_1} = (1, \ldots, 1)$ corresponding to **smallest eigenvalue** $\lambda_1 = 0$

Fiedler vector

- eigenvector $\vec{v_2}$ corresponding to λ_2 is called **Fiedler vector** → M Fiedler, 1973
- entries can be used to **find edge cuts with minimal normalized size**

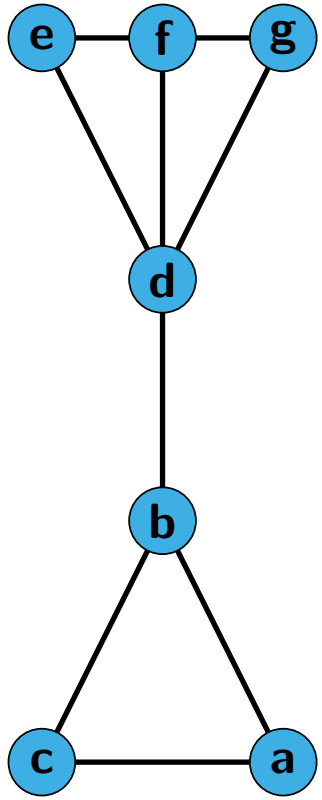
Miroslav Fiedler
1926 – 2015

image credit: `http://www.cs.cas.cz/fiedler/`

Notes:

- While the second-smallest eigenvalue of the Laplacian matrix is related to the vertex connectivity of a graph, we can use the associated eigenvector to detect minimal edge cuts. Again, we first observe that the first eigenvector of the graph Laplacian, which corresponds to the smallest eigenvalue zero, is a trivial vector that only consists of one entries. We call the eigenvector corresponding to the second-smallest eigenvalue the **Fiedler vector**.
- In our course *Statistical Network Analysis*, we have argued that the spectrum of eigenvalues and eigenvalues captures the influence of the topology of a network on the evolution of dynamical processes like, e.g. a heat diffusion process → L12, Statistical Network Analysis
- The eigenvalue sequence determines the speed of the diffusion process, while the eigenvectors give independent “modes” of the dynamics that depend on the graph topology.

Example: Fiedler vector



$C_{\text{vertex}} = 1, \lambda_2 \approx 0.398$

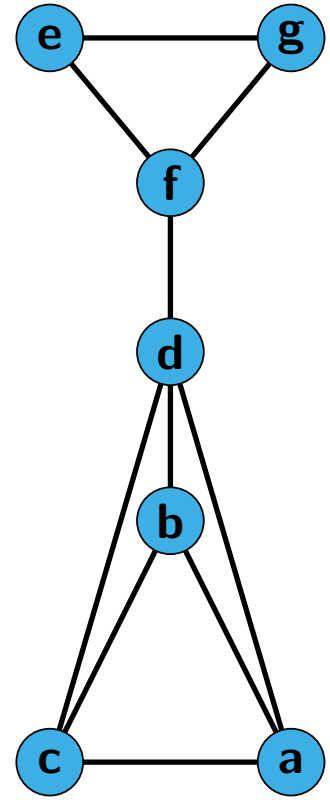
minimal edge cut

$\{C_1 = \{a, b, c\}, C_2 = \{d, e, f, g\}\}$

$N_s(C_1, C_2) \approx 0.234$

$\vec{v_2} =$

$(-0.49, -0.3, -0.49, 0.21, 0.36, 0.36, 0.36)$



$C_{\text{vertex}} = 1, \lambda_2 \approx 0.398$

minimal edge cut

$\{C_1 = \{a, b, c, d\}, C_2 = \{e, f, g\}\}$

$s(C_1, C_2) = 0.22$

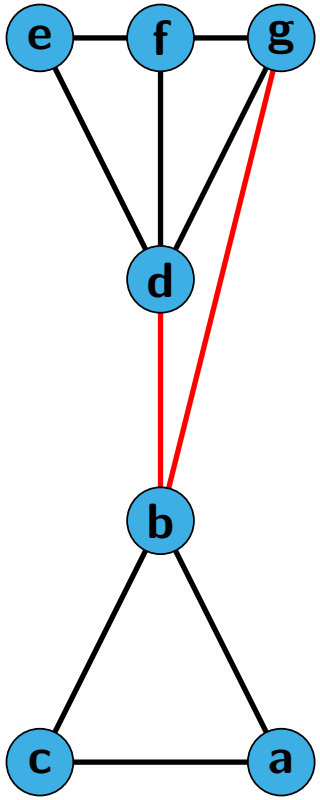
$\vec{v_2} =$

$(-0.36, -0.36, -0.36, -0.21, 0.49, 0.3, 0.49)$

Notes:

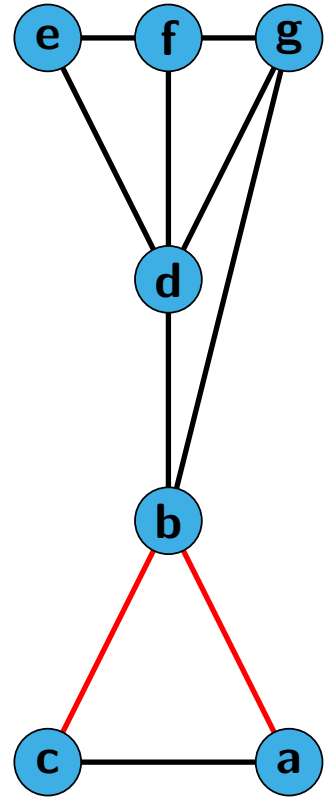
- Let us consider the relationship between the entries of the Fiedler vector and minimal edge cuts in a few example networks.
- In the left example, we have an edge cut with size one. We find that the sign of the entries in the Fiedler vector map the nodes to the two partitions of the cut.
- In the right example, we have changed the links such that we now have a minimal edge cut of size one with a different node partition. We find that the entries of the Fiedler vector change accordingly, i.e. we can again use the sign to map nodes to the two partitions. In other words: we can use the Fiedler vector to find a minimal edge cut.

Example: Fiedler vector



edge cut $\{C_1 = \{a, b, c\}, C_2 = \{d, e, f, g\}\}$

$N_s(C_1, C_2) \approx 0.417$



edge cut $\{C_1 = \{a, c\}, C_2 = \{b, d, e, f, g\}\}$

$N_s(C_1, C_2) \approx 0.625$

$$C_{\text{vertex}} = 2, \lambda_2 \approx 0.64$$

$$\vec{v_2} = (0.52, 0.19, 0.52, -0.24, -0.45, -0.37, -0.18)$$

Notes:

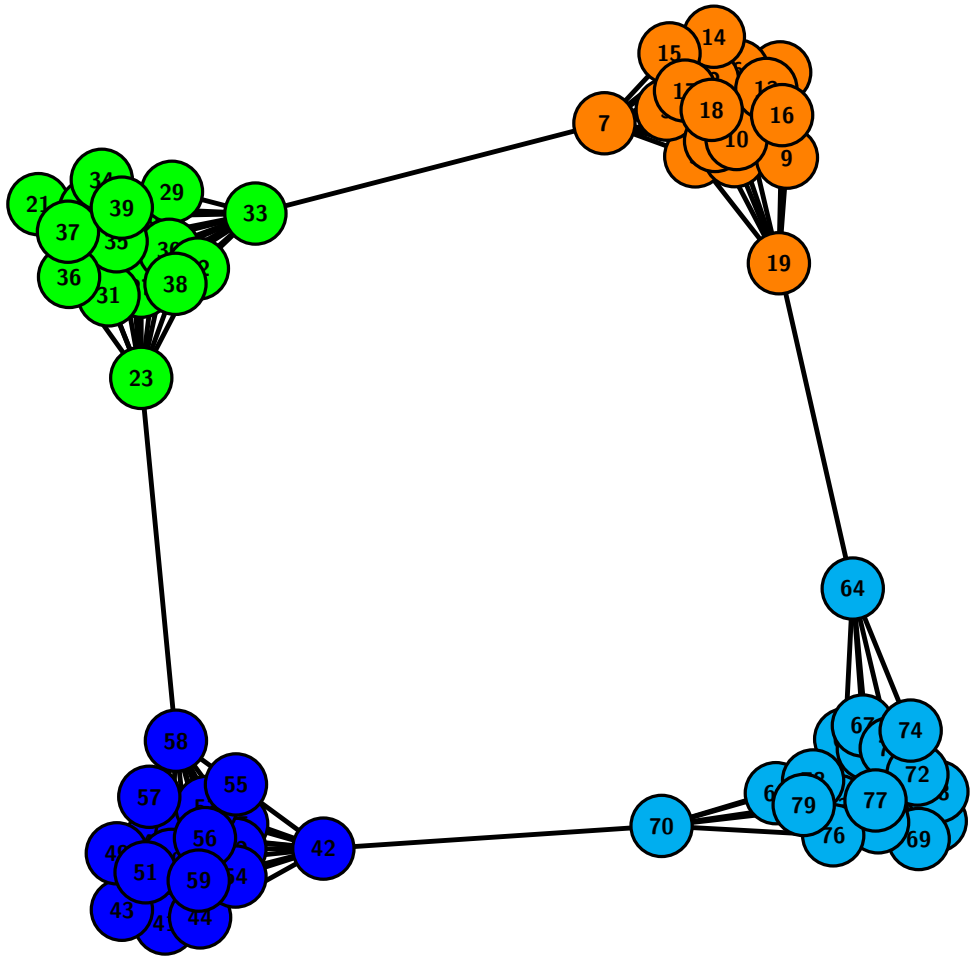
- But what if several cuts with the same size exist? Consider the example network above, where we can find two edge cuts with size two, i.e. in both cases we need to remove two edges to split the network into multiple connected components. From a clustering perspective, we intuitively prefer the cut in the left example, because it generates two communities that are each well-connected, while the edge cut on the right separates a pair of nodes that happens to be in the periphery of the network. In general, if we look for edge cuts with minimal size, we often find solutions that isolate peripheral nodes, rather than splitting the network into multiple components that are each well-connected.
- We see that this issue is fixed by the minimization of edge cuts with **normalized size**, because the normalized size considers the connectivity of nodes within the two partitions. Even though the two edge cuts above have the same size of two, the normalized cut size of the edge cut on the left is smaller, because the resulting components are better connected than in the edge cut on the right.
- We find that the Fiedler vector maps nodes to partitions that correspond to the edge cut with minimal normalized size in the left example, rather than to the edge cut with size two in the right example. In the exercise, you will consider an analytical argument that explains this behavior. It may seem remarkable that using the Fiedler vector we can efficiently (i.e. in polynomial time) the problem of finding minimal normalized edge cuts, which is NP hard. However, we technically solve a relaxed problem which, rather than providing an integer indicator function that maps nodes to partitions,

Spectral clustering

- we can use Fiedler vector to **bisect vertices**
- bisection is based on **edge cut with minimal normalized size**

spectral clustering algorithm

- Q: what if a network contains **more than two clusters**?
- A: we can **recursively apply bisection** to vertex sets in edge cut
- Q: when do we **stop recursion**?
- A: we can use **threshold for algebraic connectivity**

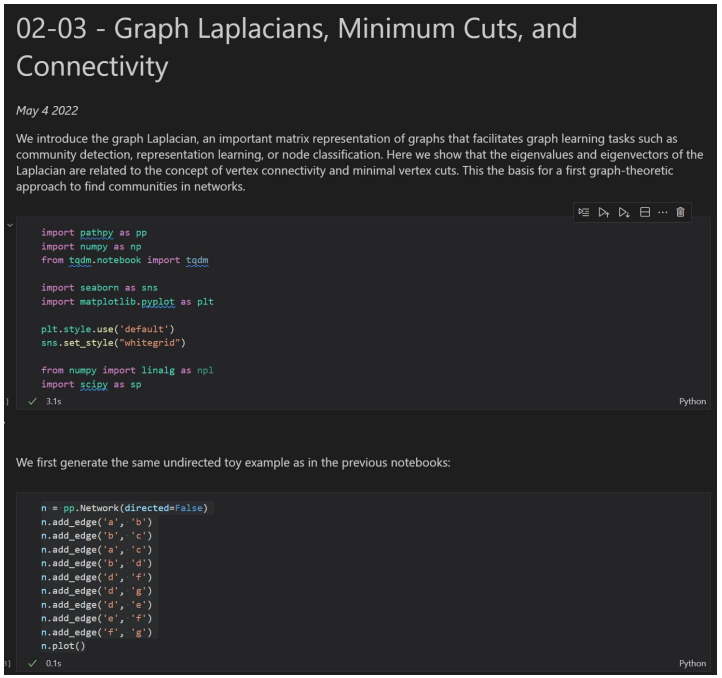


Notes:

- These remarkable properties of the Fiedler vector and the algebraic connectivity suggest an efficient algorithm to detect communities based on the Laplacian matrix. We simply use the *sign* of entries in the Fiedler vector to **bisect the network** into two partitions. We call this a “spectral bisection” of networks. This approach can be applied recursively to detect more than two communities. We will explore this in the practice session.
- The recursive application of this spectral bisection raises the question st which point we should stop to bisect the network. We have seen that the algebraic connectivity provides us with a value that indicates whether there exists a bottleneck or small vertex cut in the topology. We can thus stop the recursion as soon as the algebraic connectivity of the partitions exceeds a given threshold, which acts as a **resolution parameter** for our community detection algorithm.

Practice session

- we explore **minimum cuts and vertex connectivity**
- we compute **graph Laplacians** and **algebraic connectivity**
- we study **minimum edge cuts** and compute the **Fiedler vector**
- we use the Laplacian matrix for **spectral community detection**



02-03 - Graph Laplacians, Minimum Cuts, and Connectivity

May 4 2022

We introduce the graph Laplacian, an important matrix representation of graphs that facilitates graph learning tasks such as community detection, representation learning, or node classification. Here we show that the eigenvalues and eigenvectors of the Laplacian are related to the concept of vertex connectivity and minimal vertex cuts. This the basis for a first graph-theoretic approach to find communities in networks.

```
import pathpy as pp
import numpy as np
from tqdm.notebook import tqdm

import seaborn as sns
import matplotlib.pyplot as plt

plt.style.use('default')
sns.set_style("whitegrid")

from numpy import linalg as npl
import scipy as sp
```

✓ 3.1s Python

We first generate the same undirected toy example as in the previous notebooks:

```
n = pp.Network(directed=False)
n.add_edge('a', 'b')
n.add_edge('b', 'c')
n.add_edge('a', 'c')
n.add_edge('b', 'd')
n.add_edge('d', 'f')
n.add_edge('d', 'g')
n.add_edge('d', 'e')
n.add_edge('e', 'f')
n.add_edge('f', 'g')
n.plot()
```

✓ 0.1s Python

practice session

see notebooks 02-03 – 02-04 in `gitlab` repository at

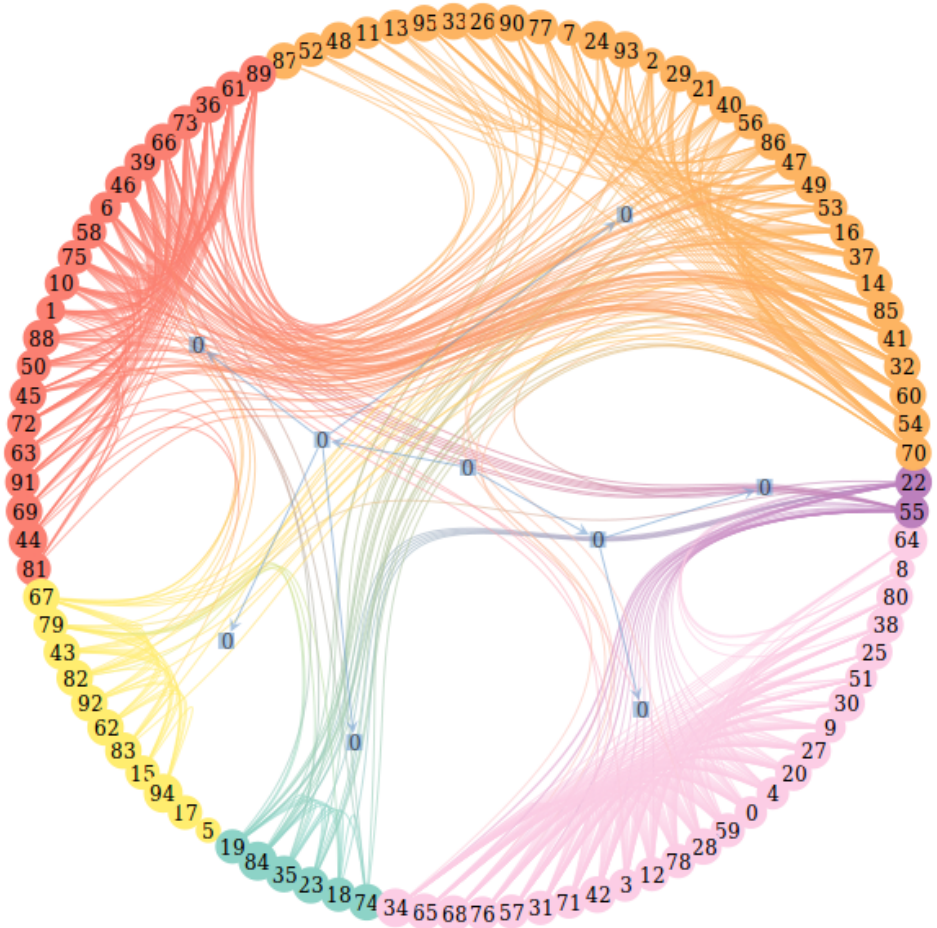
→ `https://gitlab.informatik.uni-wuerzburg.de/ml4nets_notebooks/2022_sose_ml4nets_notebooks`

Notes:

- In the second practice session, we explore small cuts and spectral clustering.

In summary …

- we revisited fundamental **graph-theoretic concepts**
- we explored algorithms to compute **shortest paths and connected components**
- we highlighted **different notions of communities in networks**
- we introduced the **graph Laplacian** and its application in **spectral clustering**

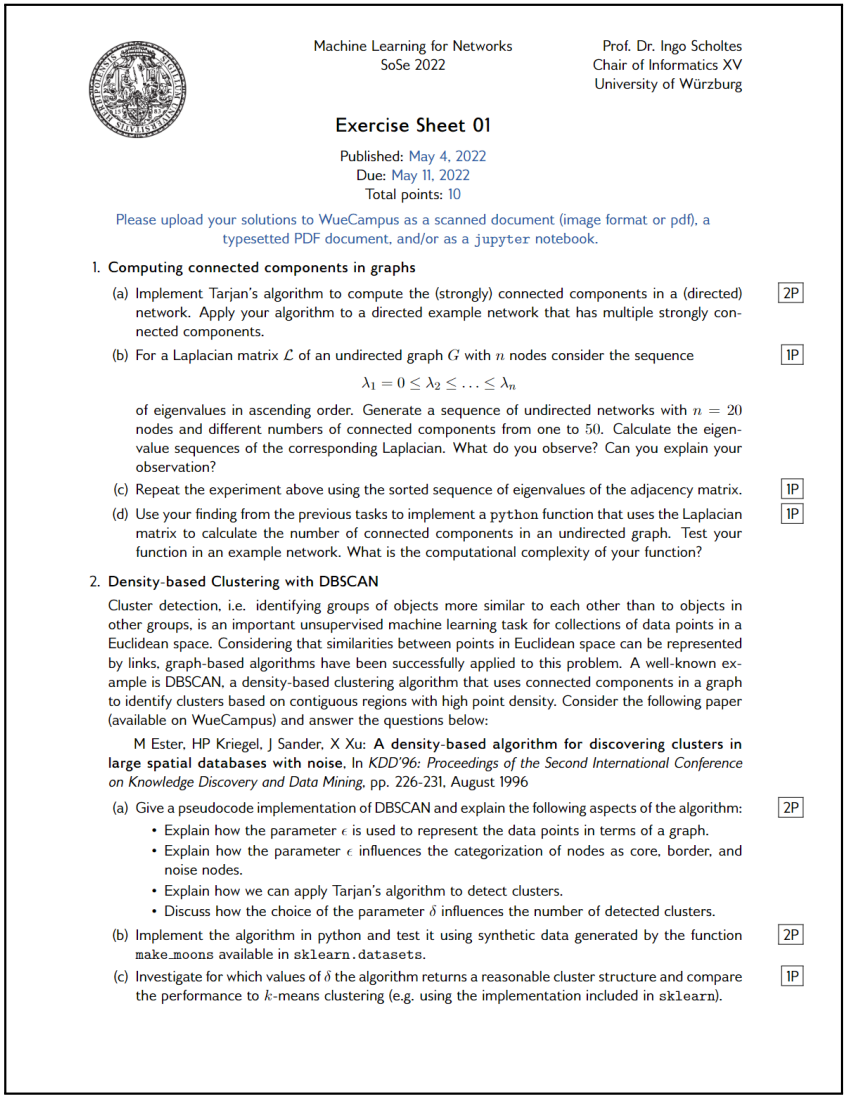


communities detected in social network of students at MIT

Notes:

Exercise sheet 01

- **first exercise sheet** will be released today
  - implement and test **Tarjan's algorithm** to compute connected components in a graph
  - explore how we can use graph Laplacians to detect connected components
  - explore graph-based density-based clustering with **DBSCAN** → M Ester at al., 1996
- solutions are due **May 11th** (via WueCampus)
- solutions are presented in exercise sessions held in week 4
- present your solution to earn bonus points



Machine Learning for Networks
SoSe 2022

Prof. Dr. Ingo Scholtes
Chair of Informatics XV
University of Würzburg

Exercise Sheet 01

Published: May 4, 2022
Due: May 11, 2022
Total points: 10

Please upload your solutions to WueCampus as a scanned document (image format or pdf), a typesetted PDF document, and/or as a `jupyter` notebook.

1. **Computing connected components in graphs**

(a) Implement Tarjan's algorithm to compute the (strongly) connected components in a (directed) network. Apply your algorithm to a directed example network that has multiple strongly connected components. [2P]

(b) For a Laplacian matrix $\mathcal{L}$ of an undirected graph G with n nodes consider the sequence [1P]

$$\lambda_1 = 0 \leq \lambda_2 \leq \ldots \leq \lambda_n$$

of eigenvalues in ascending order. Generate a sequence of undirected networks with $n = 20$ nodes and different numbers of connected components from one to 50. Calculate the eigenvalue sequences of the corresponding Laplacian. What do you observe? Can you explain your observation?

(c) Repeat the experiment above using the sorted sequence of eigenvalues of the adjacency matrix. [1P]

(d) Use your finding from the previous tasks to implement a `python` function that uses the Laplacian matrix to calculate the number of connected components in an undirected graph. Test your function in an example network. What is the computational complexity of your function? [1P]

2. **Density-based Clustering with DBSCAN**

Cluster detection, i.e. identifying groups of objects more similar to each other than to objects in other groups, is an important unsupervised machine learning task for collections of data points in a Euclidean space. Considering that similarities between points in Euclidean space can be represented by links, graph-based algorithms have been successfully applied to this problem. A well-known example is DBSCAN, a density-based clustering algorithm that uses connected components in a graph to identify clusters based on contiguous regions with high point density. Consider the following paper (available on WueCampus) and answer the questions below:

M Ester, HP Kriegel, J Sander, X Xu: **A density-based algorithm for discovering clusters in large spatial databases with noise**, In *KDD'96: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pp. 226-231, August 1996

(a) Give a pseudocode implementation of DBSCAN and explain the following aspects of the algorithm: [2P]
- Explain how the parameter ϵ is used to represent the data points in terms of a graph.
- Explain how the parameter ϵ influences the categorization of nodes as core, border, and noise nodes.
- Explain how we can apply Tarjan's algorithm to detect clusters.
- Discuss how the choice of the parameter δ influences the number of detected clusters.

(b) Implement the algorithm in python and test it using synthetic data generated by the function `make_moons` available in `sklearn.datasets`. [2P]

(c) Investigate for which values of δ the algorithm returns a reasonable cluster structure and compare the performance to k-means clustering (e.g. using the implementation included in `sklearn`). [1P]

Notes:

Self-study questions

1. Give an example for a walk, path, and cycle in a network.
2. Why do the entries of the k-th power of an adjacency matrix count walks of length k?
3. Explain how we can use adjacency matrix powers $\sum_{k=1}^{l} \mathbf{A}^k$ to compute the diameter of a network.
4. Investigate the DBSCAN algorithm. How are connected components used to detect clusters in Euclidean data?
5. How is the model selection problem in clustering reflected in the parameters of the DBSCAN algorithm.
6. How is the definition of a community related to edge cuts?
7. Give an example for a network where the sizes of both the minimal vertex and edge cut are $k = 5$.
8. Give an example for a network with two edge cuts with size two and different normalized cut sizes.
9. What is the multiplicity of zero in the eigenvalue sequence of the Laplacian for a graph with three connected components?
10. What is the computational complexity of calculating the eigenvalues and eigenvectors of a Laplacian matrix?

Notes:

References

reading list

- R Bellman: **On a routing problem**, In Quarterly of Applied Mathematics, No. 16, 1958
- EW Dijkstra: **A note on two problems in connexion with graphs**, In Numerische Mathematik, 1959
- RW Floyd: **Algorithm 97: Shortest Path**, Communications of the ACM, 1962
- RE Tarjan: **Depth-first search and linear graph algorithms**, SIAM Journal on Computing, 1972
- M Fiedler: **Algebraic Connectivity of Graphs**, Czechoslovak Mathematical Journal 23, 1973
- TMJ Fruchterman, EM Reingold: **Graph Drawing by Force-Directed Placement**, Software – Practice & Experience, 1991
- J Shi and J Malik: **Normalized Cuts and Image Segmentation**, IEEE Transactions on Patterns Analysis and Machine Intelligence, 2000
- M Schaub, JC Delvenne, M Rosvall, R Lambiotte: **The many facets of community detection in complex networks**, Applied Network Science, 2017
- M Ester, HP Kriegel, J Sander, X Xu: **A density-based algorithm for discovering clusters in large spatial databases with noise**, Proceedings of KDD'96, 1996
- M Belkin, P Niyogi: **Laplacian Eigenmaps for Dimensionality Reduction and Data Representation**, Neural Computation, 2003

→ section 4

888 IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 22, NO. 8, AUGUST 2000

Normalized Cuts and Image Segmentation

Jianbo Shi and Jitendra Malik, *Member, IEEE*

Abstract—We propose a novel approach for solving the perceptual grouping problem in vision. Rather than focusing on local features and their consistencies in the image data, our approach aims at extracting the global impression of an image. We treat image segmentation as a graph partitioning problem and propose a novel global criterion, the *normalized cut*, for segmenting the graph. The *normalized cut* criterion measures both the total dissimilarity between the different groups as well as the total similarity within the groups. We show that an efficient computational technique based on a generalized eigenvalue problem can be used to optimize this criterion. We have applied this approach to segmenting static images, as well as motion sequences, and found the results to be very encouraging.

Index Terms—Grouping, image segmentation, graph partitioning.

1 Introduction

Nearly 75 years ago, Wertheimer [24] pointed out the importance of perceptual grouping and organization in vision and listed several key factors, such as similarity, proximity, and good continuation, which lead to visual grouping. However, even to this day, many of the computational issues of perceptual grouping have remained unresolved. In this paper, we present a general framework for this problem, focusing specifically on the case of image segmentation.

Since there are many possible partitions of the domain I of an image into subsets, how do we pick the "right" one? There are two aspects to be considered here. The first is that there may not be a single correct answer. A Bayesian view is appropriate—there are several possible interpretations in the context of prior world knowledge. The difficulty, of course, is in specifying the prior world knowledge. Some of it is low level, such as coherence of brightness, color, texture, or motion, but equally important is mid- or high-level knowledge about symmetries of objects or object models. The second aspect is that the partitioning is inherently hierarchical. Therefore, it is more appropriate to think of returning a tree structure corresponding to a hierarchical partition instead of a single "flat" partition.

This suggests that image segmentation based on low-level cues cannot and should not aim to produce a complete final "correct" segmentation. The objective should instead be to *use the low-level coherence of brightness, color, texture, or motion attributes to sequentially come up with hierarchical partitions*. Mid- and high-level knowledge can be used to either confirm these groups or select some for further attention. This attention could result in further repartitioning or grouping. The key point is that image partitioning is to be done from the big picture downward, rather like a painter first marking out the major areas and then filling in the details.

Prior literature on the related problems of clustering, grouping and image segmentation is huge. The clustering community [12] has offered us agglomerative and divisive algorithms; in image segmentation, we have region-based merge and split algorithms. The hierarchical divisive approach that we advocate produces a tree, the *dendrogram*. While most of these ideas go back to the 1970s (and earlier), the 1980s brought in the use of Markov Random Fields [10] and variational formulations [17], [2], [14]. The MRF and variational formulations also exposed two basic questions:

1. What is the criterion that one wants to optimize?
2. Is there an efficient algorithm for carrying out the optimization?

Many an attractive criterion has been doomed by the inability to find an effective algorithm to find its minimum—greedy or gradient descent type approaches fail to find global optima for these high-dimensional, nonlinear problems.

Our approach is most related to the graph theoretic formulation of grouping. The set of points in an arbitrary feature space are represented as a weighted undirected graph $G = (V, E)$, where the nodes of the graph are the points in the feature space, and an edge is formed between every pair of nodes. The weight on each edge, $w(i, j)$, is a function of the similarity between nodes i and j.

In grouping, we seek to partition the set of vertices into disjoint sets $\mathbf{V}_1, \mathbf{V}_2, \ldots, \mathbf{V}_m$, where by some measure the similarity among the vertices in a set $\mathbf{V}_i$ is high and, across different sets $\mathbf{V}_i$, $\mathbf{V}_j$ is low.

To partition a graph, we need to also ask the following questions:

1. What is the precise criterion for a good partition?
2. How can such a partition be computed efficiently?

In the image segmentation and data clustering community, there has been much previous work using variations of the minimal spanning tree or limited neighborhood set approaches. Although those use efficient computational

- J. Shi is with the Robotics Institute, Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, PA 15213. E-mail: jshi@cs.cmu.edu
- J. Malik is with the Electrical Engineering and Computer Science Division, University of California at Berkeley, Berkeley, CA 94720. E-mail: malik@cs.berkeley.edu.

Manuscript received 4 Feb. 1998; accepted 16 Nov. 1999.
Recommended for acceptance by M. Shah.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number 107618.

0162-8828/00/$10.00 © 2000 IEEE

Notes: