Statistical Network Analysis

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
Foundations of Graph Theory

October 27, 2021

Notes:

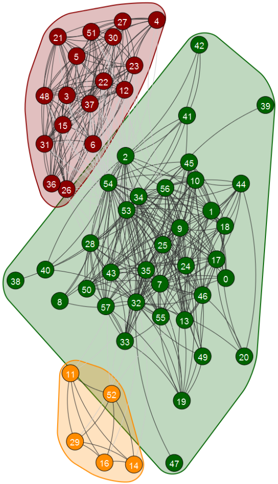
- **Lecture L02:** Foundations of Graph Theory 27.10.2021
- In this lecture, we will introduce graph-theoretic basics, explain how to identify community structures and show how to rank nodes by importance.
    - Basic definitions
    - Paths, diameter, and components
    - Adjacency matrices, components, and communities

Motivation

- large volume of **relational data** covering
  - technical infrastructures
  - information networks
  - biological systems
  - social organizations
- **network** abstraction of **dyadic relations**
  - focus on **topology**: who links to whom
  - unified mathematical model/language
  - toolbox of algorithms and metrics

exemplary network-analytic questions

- can all students influence each other?
- which students are connected by short paths?
- can we identify natural groups?
- which students have a central position?



natural group structure in student network

Notes:

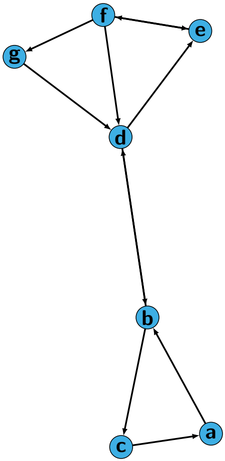
- The graph or network abstraction of **relational data** provides both a modelling framework as well as a common language for the study of complex systems. It enables the application of a toolbox of measures, algorithms, and statistics to quantitatively analyse systems from different disciplines. These methods allows us to answer questions about the topology of a graph or network, e.g.

    1. Are all nodes in the network connected by a path?
    2. Can we identify groups (or communities) of nodes that are more connected to themselves than to other nodes?
    3. Which nodes have the most central position in the network?

- For network models of real complex system (like, e.g., a group of students or employees, a network of documents, or users on social media) these abstract problems translate to practically relevant questions, e.g.

    1. Can information flow between all students? Can all students influence each other? Can an infectious disease spread to the whole population of students?
    2. Which groups of my employees work together closely? Which documents in this hyperlink graph address similar topics? Are there any "bottlenecks" in our communication network, i.e. are there nodes or links that are likely to be overloaded?
    3. Who are the most central employees in our company? How would our team be affected if a certain employee left? Whom should we target with an advertisement campaign?

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**

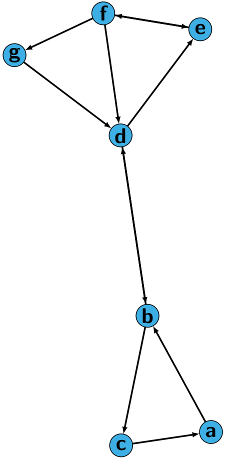
example network

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

Notes:

- Before we can study first network-analytic problems we formally introduce graph-theoretic foundations. In this course, we use the terms **vertex** and **node**, as well as **edge** and **link** 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 $\rightarrow$ 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. $\rightarrow$ slide 5 on weighted graphs

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

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

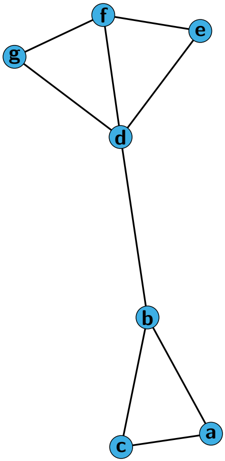
- for directed networks (no self-loops)

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

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 $\rightarrow$ slide 6 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

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

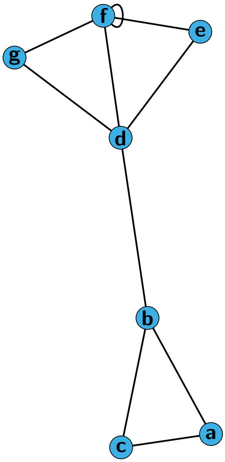
- for undirected networks (no self-loops)

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

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 $\mathbf{A}$
- two different representations of self-loops:
  1. $(i, i) \in E \implies A_{ii} = 1$
  2. $(i, i) \in E \implies A_{ii} = 2$

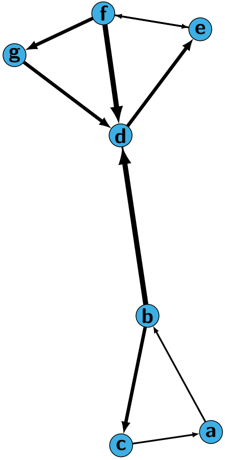
adjacency matrix of network with self-loop

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

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** $w : E \to \mathbb{R}$ 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

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

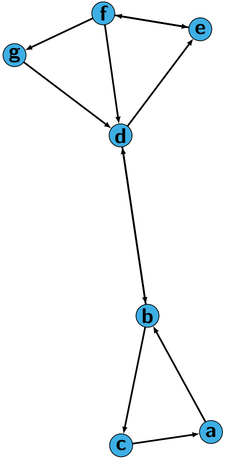
adjacency matrix of weighted network

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

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. $d_{out}(i) := |\{j \in V : (i, j) \in E\}|$

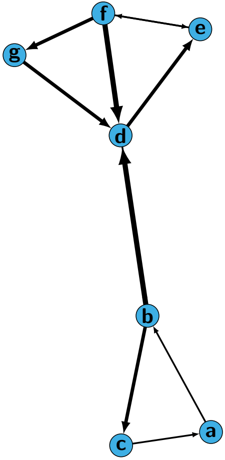
example network

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

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. $\rightarrow$ more in L07

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

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

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

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.

Representing networks

adjacency matrix

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

- requires n^2 **bits** of memory
- supports algebraic operations
- suitable for **small networks**

adjacency list

```
a,b
b,c
b,d
c,a
d,b
d,e
e,f
f,d
f,e
f,g
g,d
```

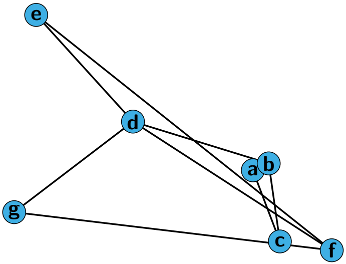
- requires $\mathcal{O}(m \log n)$ **bits** of memory
- preferable if $m \ll n^2$
- suitable for **large networks**

Notes:

- We end this introductory section with a reflection on data structures that are used to represent networks in a computer. A widely used approach is to store a list of tuples that represent adjacent nodes, i.e. pairs of nodes connected by links. We call this an **adjacency list representation**. For a network with n nodes and m links we need to store m pairs of nodes, where each node requires $\log n$ bits.
- A second approach is to store the **adjacency matrix** of a network (e.g. as an array of arrays). This has the advantage that we can use standard packages to perform algebraic operations (matrix multiplication, matrix powers, eigenvector calculations, etc.) that have a natural interpretation in networks. $\rightarrow$ more in L11 . For a network with n nodes a naive binary adjacency matrix representation requires us to store n^2 bits, independent of the number of links that actually exist. representation.
- For fully connected networks, adjacency list and adjacency matrix representations are equally efficient in terms of memory requirements. However, for networks where the majority of node pairs are not connected, an adjacency list representation is more efficient. In particular, if the average degree of nodes is a small constant we have $m \sim n$ and an adjacency list requires only $\mathcal{O}(n \log n)$ bits, while a naive adjacency matrix requires n^2 bits. Networks where the number of links is comparable to the number of nodes are called sparse networks. We can use **sparse matrix representations** to efficiently store matrices with many zero entries. This will be important when we apply spectral methods to large networks.
- As we shall see later, the question what is the most efficient representation of networks has interesting relations to information theory.

Visualizing networks

- to visualize networks, we need **geometric representation** of nodes and edges
- <u>but:</u> networks can capture arbitrary **non-Euclidean topologies**
- need to **map nodes to coordinates** in Euclidean space with 1 – 3 dimensions
- good **layouts of graph** 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

layouts used in graph drawing

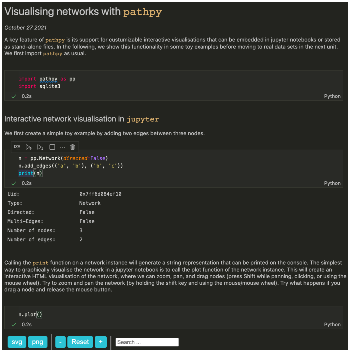
- Circular layout
- Force-directed layouts
- Spectral layout

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). Clearly, 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 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.

Practice session

- we introduce the `python`-based **network analysis package** `pathpy`
- we show how to represent **directed, undirected, weighted, and multi-edge networks** with node/edge attributes
- we demonstrate how to **read and write network data** from/to files and databases
- we explain how to **visualize networks** with `pathpy`



practice session

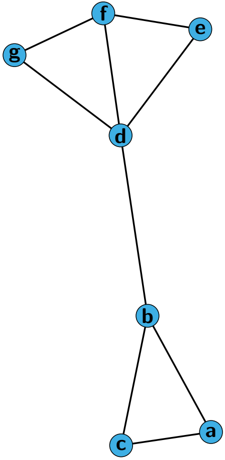
see notebooks 02-01 – 02–03 in `gitlab` repository at

→ `https://gitlab.informatik.uni-wuerzburg.de/ml4nets_notebooks/2021_wise_sna_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 further show how you can read/write network data from/to files or databases and how you can visualize networks based on layout algorithms.
- You can find the jupyter notebooks (and data) used in the practice sessions in an accompanying gitLab repository.

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 $i \neq j$

- **length of path, walk, or cycle** is defined as $\text{len}(p_0, \ldots, p_l) := l$

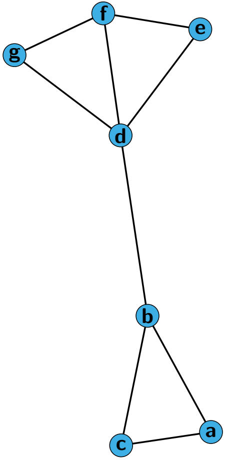
example

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

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 startpoint 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 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.

Topological distance



- **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

example

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

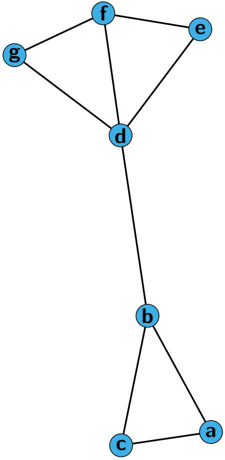
shortest path algorithms $\rightarrow$ practice session & exercise

- Dijkstra $\rightarrow$ single source, $O(n \log n + m)$
- Floyd-Warshall $\rightarrow$ all-pairs, $O(n^3)$
- Bellman-Ford $\rightarrow$ single source w/ negative cycles, $O(n \cdot m)$

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**.
- 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.
- 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.

Diameter and average shortest path length



- **diameter** $\mathrm{diam}(G)$ of network $G = (V, E)$ is length of the longest shortest path

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

- **average shortest path length** $\langle l \rangle$ is the average distance between nodes, i.e.

$$\langle l \rangle := \frac{1}{|V|^2} \sum_{(v,w) \in V \times V} \mathrm{dist}(v, w)$$

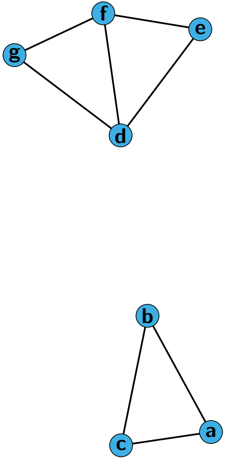
example

- $\mathrm{diam}(G) = 3$
- $\langle l \rangle = 1.59$

Notes:

- 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, rather than from a feature of any particular node or link. This is particularly true for large networks, where changing a single node or link is likely to not change the diameter.
- The diameter is defined as the length of the **longest shortest path**, i.e. it is the upper bound for the number of links that need to be traversed for one node to indirectly influence another node in the system. If we study the propagation of information starting from a node, the network diameter tells us how long it will take at maximum for the last node to see the information, if that information is passed along the shortest paths in the network.
- The upper bound for the shortest path length can be seen as a "worst-case estimate" that does not tell us anything about "typical shortest path lengths". We can instead characterize the distribution of path lengths by the **average shortest path length**. This provides a single statistical feature that captures one important aspect of the topology of a complex network. It tells us how many steps it will take – on average – for one node to be able to indirectly influence another node in the system. For some networks the diameter can be large, even if most shortest path lengths are actually very small. In these cases, the average shortest path length is a better characterization for the shortest path structures in a network.

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

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

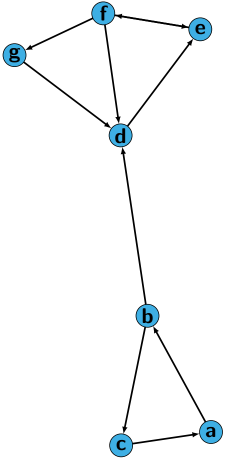
example

- 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. As we will see in the practice session, we can efficiently calculate the connected components of a graph using Tarjan's algorithm.
- 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 L07
- For finite-size empirical networks, we often use a reasonable threshold (like e.g. a fraction of 0.95 or 0.99), i.e. we accept that there is a small fraction of disconnected nodes. In these cases, we often disregard those nodes and study the largest connected component as a model of the system in question. In some network analysis packages "giant connected component" is used as a synonym for "largest connected component".

Connected components in directed graphs



example

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

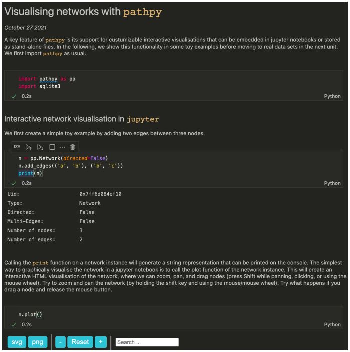
- we distinguish between **strongly and weakly connected** directed networks
- directed network is **weakly connected** iff corresponding undirected network is connected
- directed network $G = (V, E)$ is **strongly connected** iff $dist(v, w) < \infty \quad \forall v, w \in V$
- **strongly connected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

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 study **Dijkstra's algorithm for single-source shortest paths**
- we implement the **Floyd-Warshall algorithm for all-pairs shortest-paths**
- we use Tarjan's algorithm to **calculate maximally connected subgraphs**
- we apply those algorithms to empirical networks



practice session

see notebooks 02-04 and 02–05 in `gitlab` repository at

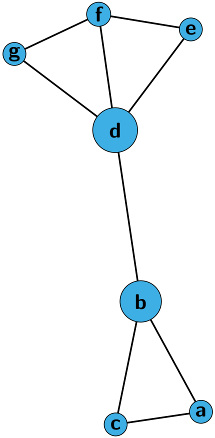
→ `https://gitlab.informatik.uni-wuerzburg.de/ml4nets_notebooks/2021_wise_sna_notebooks`

Notes:

- In the second practice session of today's lecture, we show how to implement two basic shortest path algorithms in `pathpy`, Dijstra's algorithm for the single-source-shortest-path problem as well as the Floyd-Warshall algorithm for the all-pairs-shortest-path problem.
- We then use Tarjan's algorithm to calculate connected components in directed and undirected networks.
- We finally use those algorithms to compute the diameter, component sizes, and average shortest path lengths of empirical networks.

In summary

- we introduced **fundamental definitions and concepts** of graph theory
- we showed how to **construct, read, write, and visualize networks with** `pathpy`
- we repeated foundational algorithms to compute **shortest paths and connected components**
- we highlighted relations between **matrix operations** and paths, components, and clusters in graphs



open questions

- how can we **detect cluster patterns** in large networks?
- how can we assess the **importance of nodes** in a network?

Notes:

- In today's lecture we have introduced foundational concepts of network analysis and graph theory, such as directed, undirected, and weighted networks, node degrees, walks and paths, the adjacency matrix and the meaning of its powers as well as connected components. These concepts are the basis for advanced methods that we will introduce in the following weeks.
- We have further introduced the notion of **community structures**. This is an example for an important network-analytic problem that we will address in the coming week.

Self-study questions

1. Give examples for complex systems that can naturally be represented by directed, undirected, or weighted networks.
2. Give an example for a walk, path, and cycle in a network.
3. Give an example for an acyclic network. How can we detect whether a network is acyclic?
4. Summarize the advantages and disadvantages of adjacency matrix and adjacency list representations.
5. How can the number of edge crossings be used to evaluate graph layouts?
6. Define the average shortest path length of a network.
7. Why do the entries of the k-th power of an adjacency matrix count walks of length k?
8. Explain how we can use adjacency matrix powers $\sum_{k=1}^{l} \mathbf{A}^k$ to compute the diameter of a network.
9. What is the computational complexity of calculating the k-th power of a network with n nodes.
10. Given an example for a weakly connected network where all nodes are in different strongly connected components.

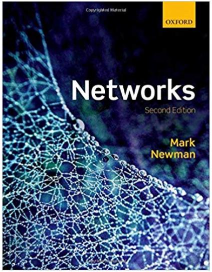
Notes:

References

reading list

- LR Ford: **Network flow theory**, Technical Report P-923, The Rand Corporation, 1956
- 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
- RE Tarjan, **Depth-first search and linear graph algorithms**, SIAM Journal on Computing, 1972
- TMJ Fruchterman, EM Reingold: **Graph Drawing by Force-Directed Placement**, Software – Practice & Experience, 1991
- TH Cormen, CE Leiserson, RL Rivest, C Stein: **Introduction to Algorithms**, Third Edition, 2009
- M Newman: **Networks**, Oxford University Press, 2010

  → Chapters 6 & 11
- D Easley, J Kleinberg: **Networks, Crowds, and Markets: Reasoning about a highly interconnected world**, Cambridge University Press, 2010 → Chapter 2
- V Latora, V Nicosia, G Russo: **Complex Networks: Principles, Methods, and Applications**, Cambridge University Press, 2017 → Chapters 1 & 9



Notes: