Module 6
Network Topology

Network Topology (1)

Network topology refers to the way different computers, devices, or nodes connect to each other in a communication network. It describes their physical arrangement and explains the logical flow of information throughout the network. Network topology plays a crucial role in determining the performance of neural networks:

- **Depth and Width**: The depth and width of a neural network refer to the number of layers and the number of neurons in each layer, respectively. Deeper networks with more layers can learn hierarchical representations of data, capturing increasingly complex features. However, very deep networks can suffer from vanishing gradients or overfitting if not properly regularized.
- **Layer Types**: Different types of layers, such as recurrent layers and dense layers, serve different purposes and are suitable for different types of data.

Network Topology (2)

- **Connectivity**: The connectivity pattern between neurons in a neural network can affect its performance. Sparse connectivity, where neurons are only connected to a subset of neurons in the next layer, can reduce computational complexity but may require careful design to ensure effective information flow.
- **Skip Connections**: Skip connections, or residual connections, allow information to bypass specific layers and be passed directly to subsequent layers. This can help alleviate the vanishing gradient problem in deep networks and facilitate the training of deeper architectures.
- **Topology Optimization**: optimized topology can significantly improve the performance of a network. This includes techniques such as architecture search algorithms, which automatically search for the best network architecture for a given task, and network pruning, which removes redundant connections or neurons to reduce computational complexity without significantly affecting performance.

Network Topology in Reservoir Computing (1)

The network topology, or the structure of connections between neurons within the reservoir, plays a crucial role in determining the performance of reservoir computing systems.

- **Information Processing Capacity**: The topology of the reservoir determines its information processing capacity. A well-connected reservoir with dense interconnections between neurons can potentially capture complex temporal dynamics and exhibit high computational power. On the other hand, overly sparse or poorly connected reservoirs might struggle to capture the input-output mappings effectively.
- **Echo State Property**: The echo state property is a fundamental requirement for reservoirs in RC. It implies that the reservoir's dynamics should exhibit fading memory; past inputs should influence the network's current state for only a finite duration.

Network Topology in Reservoir Computing (2)

- **Dynamic Range and Nonlinearity**: The dynamic range and nonlinearity of the reservoir's response are influenced by its topology. A diverse range of connections and feedback loops can lead to a rich set of dynamical behaviors, allowing the reservoir to perform complex computations and nonlinear transformations on input data.
- **Robustness to Perturbations**: The reservoir’s resilience to noise and perturbations can be affected by its topology. Certain topologies might be more robust to noise or parameter variations, while others might be more sensitive. Robustness is crucial for real-world applications where the system must operate reliably in varying conditions.
- **Training Efficiency**: The effectiveness of the learning algorithm during training depends on the reservoir’s topology. Some topologies might facilitate faster convergence during training, while others might require more extensive tuning or regularization to achieve desirable performance.

Network Topology in Reservoir Computing (2)

- **Resource Efficiency**: The computational and memory resources required to implement the reservoir are influenced by its topology. Complex topologies might require more neurons and connections, leading to higher resource consumption. Optimizing the topology can help balance performance with resource constraints.

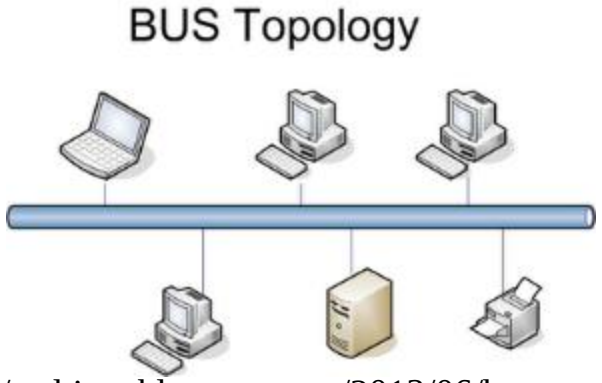
Fundamental Network Topologies (1)

- A physical topology explains how computers, devices, or nodes connect with each other in a network based on their location. It involves assessing the physical layout of network cables and workstations. A logical topology, however, explains how data flows from one device to another based on network protocols. It assesses the way devices communicate with each other internally.
- Therefore, network topology defines the virtual shape, layout, and structure of a network from both a physical and logical viewpoint.
- There are seven primary network topology types.

1. Point-to-point network topology involves connecting two nodes or devices using a common link. Some of the advantages of point-to-point topology are high bandwidth and speed, low latency, and easy maintenance

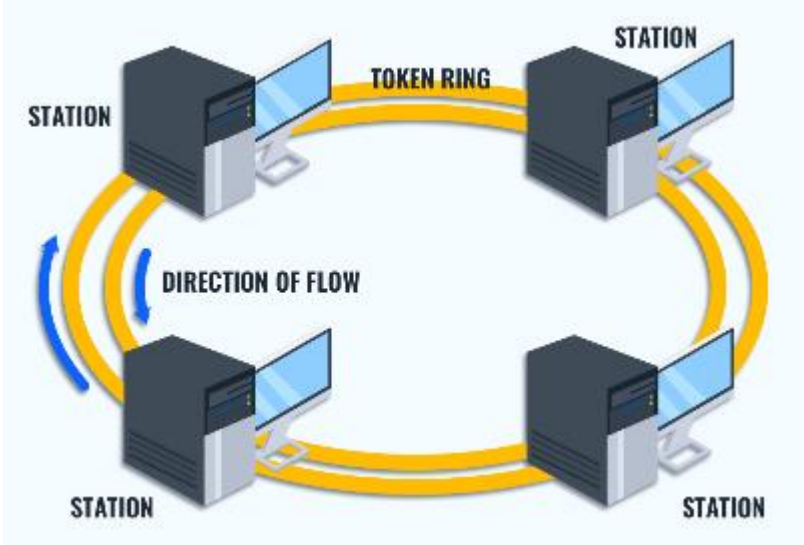
Fundamental Network Topologies (2)

2. Bus network topology is where all of the devices or nodes are connected to a common line (or bus) → easy installation and implementation, minimal hardware wire, and cost efficiency.



https://cedtinet.blogspot.com/2013/06/bus-topology.html

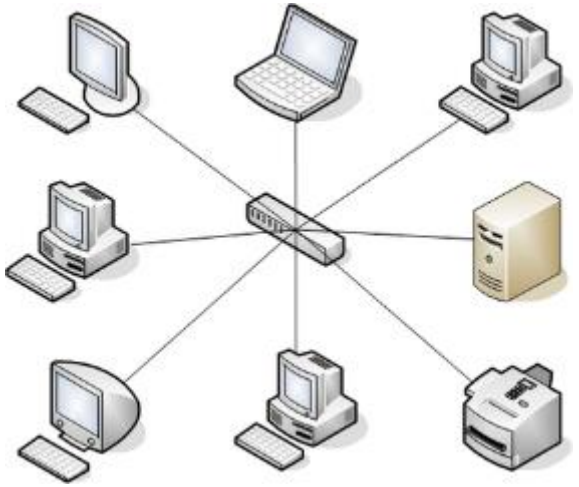
3. Ring network topology consists of two primary point-to-point links that connect one device to two more devices on each side. This creates a ring of devices through which data can flow until it reaches its target device → minimal hardware wire, minimal data collision.



https://www.cbtnuggets.com/blog/technology/networking/what-is-dual-ring-topology

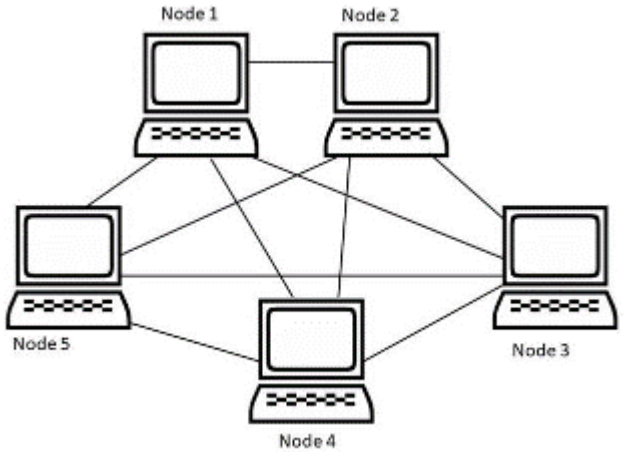
Fundamental Network Topologies (3)

4. Star network topology is where each device or node connects to a central network hub. Devices use this central hub to communicate with each other indirectly → through centralized control, simple scalability and reconfiguration, and cost efficiency.



https://en.wikipedia.org/wiki/Star_network

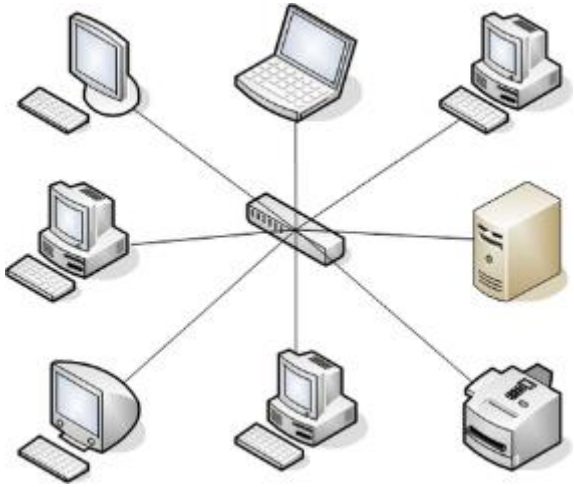
5. Mesh network topology involves creating a dedicated point-to-point link between each device in a network → fast communication, more privacy and better security, and decreased congestion on channels.



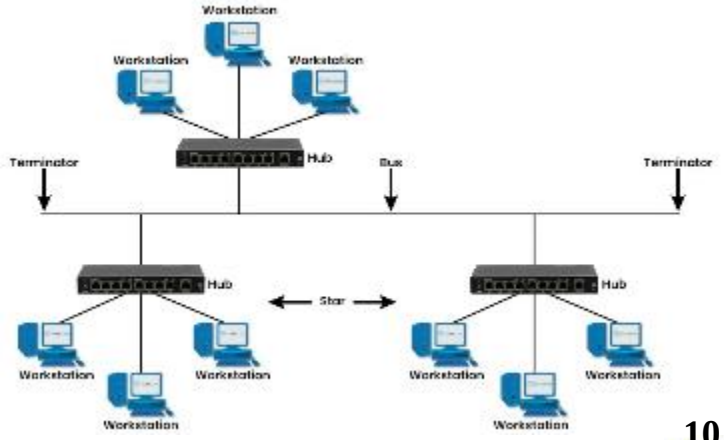
https://www.edrawsoft.com/article/mesh-topology.html

Fundamental Network Topologies (4)

6. Star network topology is where each device or node connects to a central network hub. Devices use this central hub to communicate with each other indirectly → through centralized control, simple scalability and reconfiguration, and cost efficiency.
7. Tree network topology connects star networks by integrating bus networks to create a parent-child hierarchy → , Extended distance network coverage, limited data loss, and increased number of direct and indirect nodes.



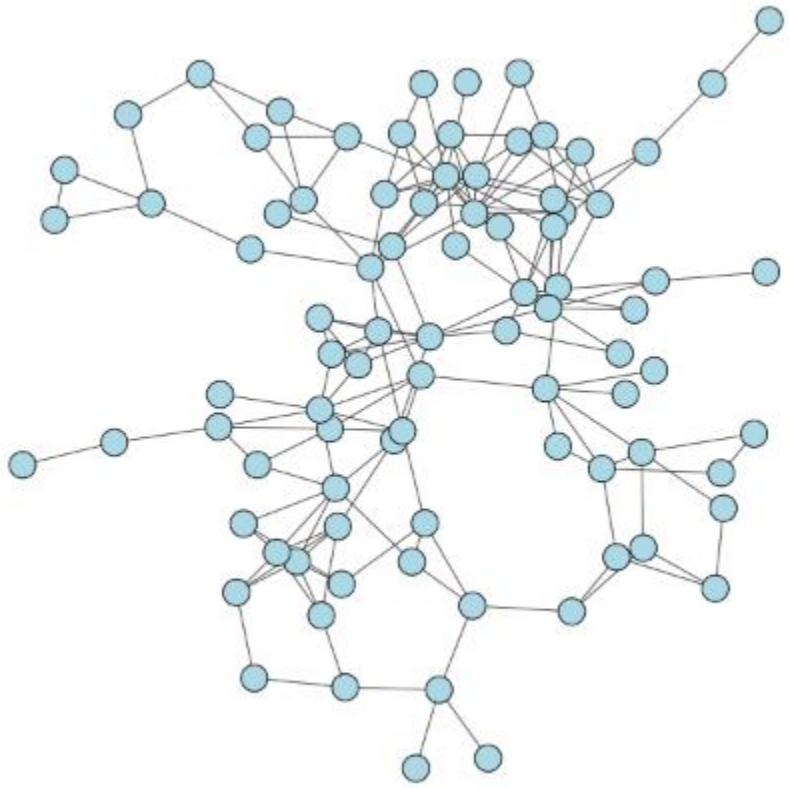
https://en.wikipedia.org/wiki/Star_network



https://www.pynetlabs.com/what-is-tree-topology-in-computer-network/

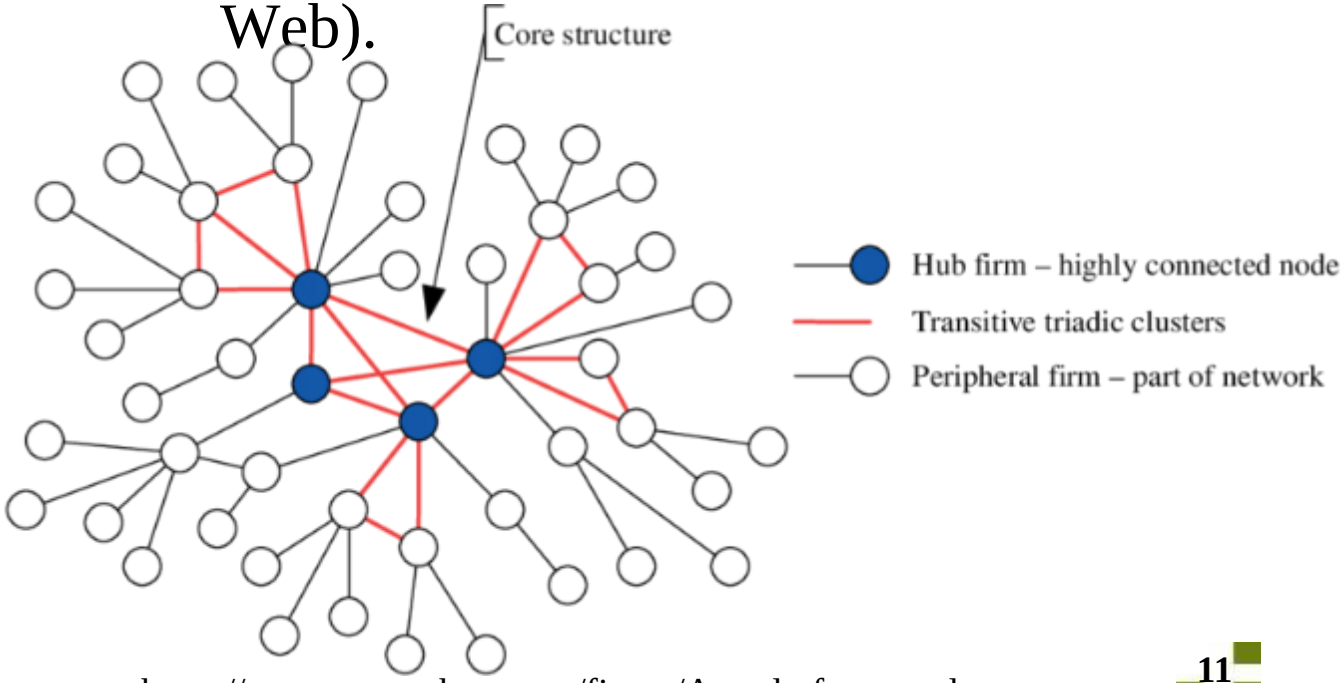
Social and Biological Network Topologies (1)

- Random Network



https://dsg.ac.upc.edu/esterl/KDet/

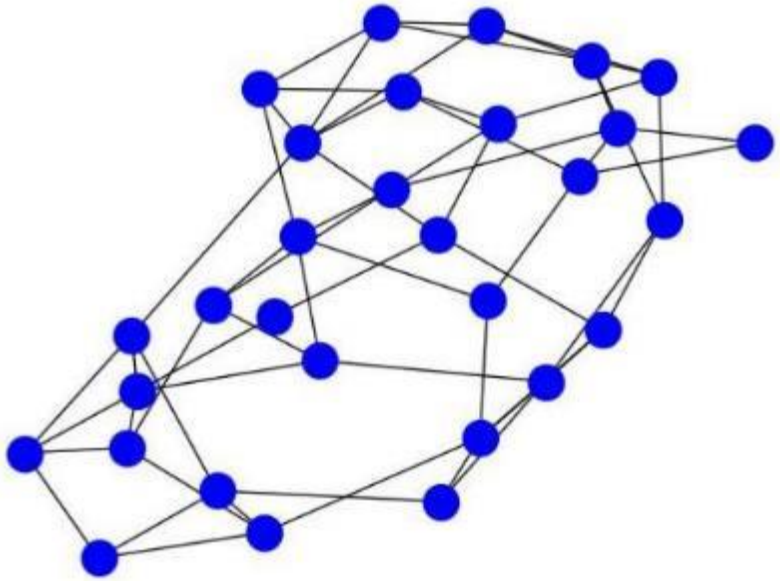
- A scale-free network follows a power law distribution where most nodes have few connections while a few nodes have many connections (the World Wide Web).



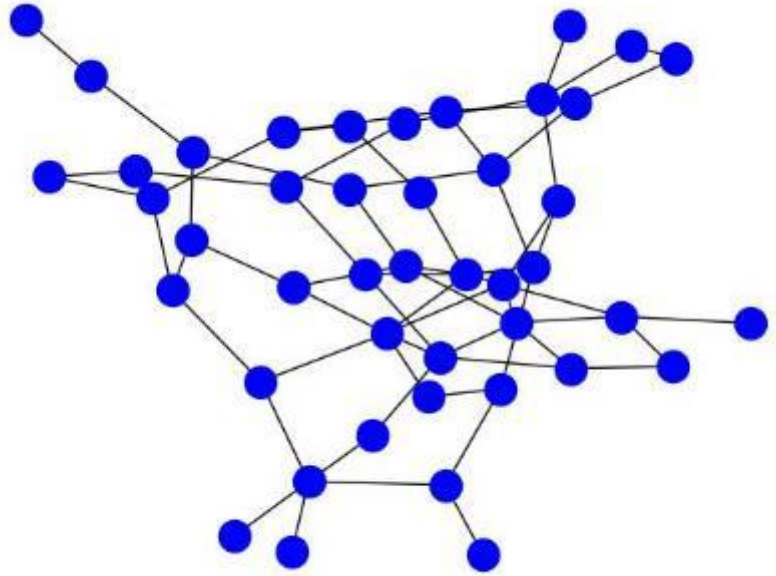
https://www.researchgate.net/figure/A-scale-free-topology-demonstrating-a-core-periphery-structure_fig3_263268796

Social and Biological Network Topologies (2)

A small-world network is a type of graph in which most nodes are not neighbors of one another, but most nodes can be reached from every other by a small number of hops.

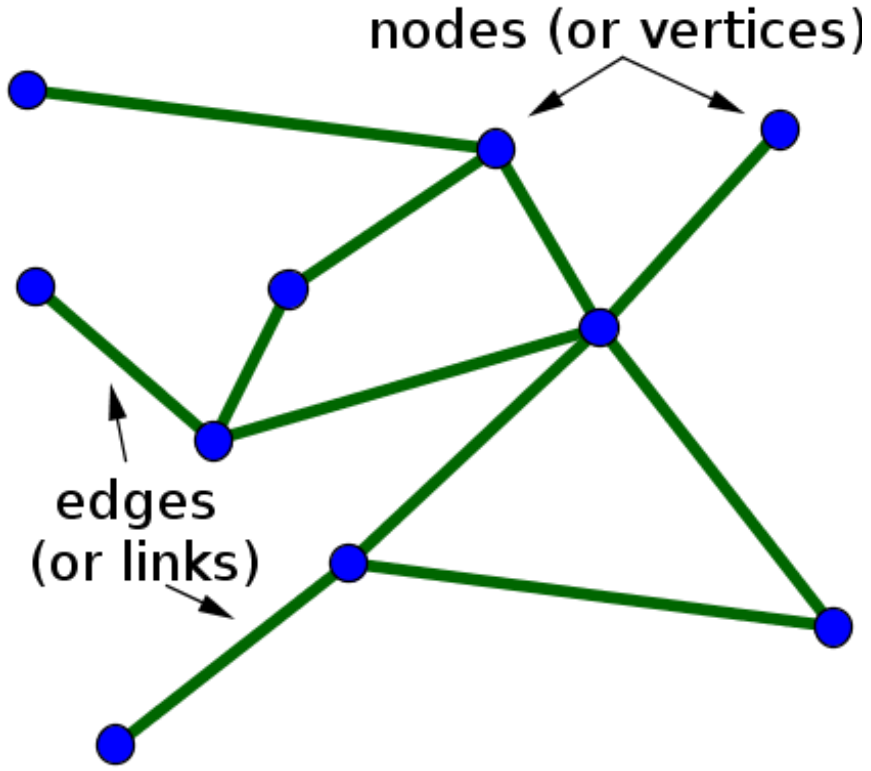


A small-world power-law network is similar to a small-world network where distance nodes have a lower chance (decay power law) of forming a connection than nearby nodes.



Network Graph (1)

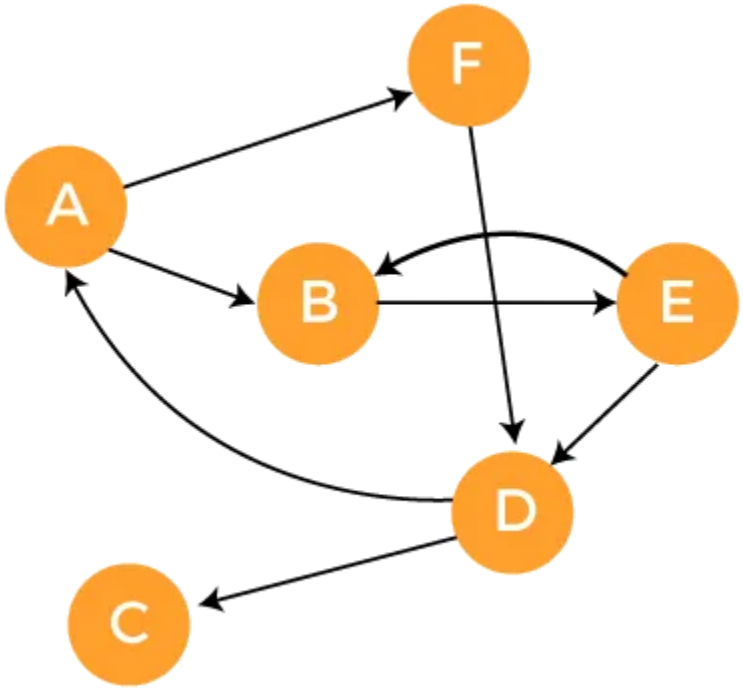
- Graphs are mathematical representations of relationships between objects or entities, points connected by lines. There are different types of graphs, each with its unique characteristics and applications.
- A graph $G(V, E)$ consists of a number of vertices V (nodes) and a set of edges E (links), where each edge connects a pair of vertices.



https://mathinsight.org/image/small_undirected_network_labeled

Directed Graphs

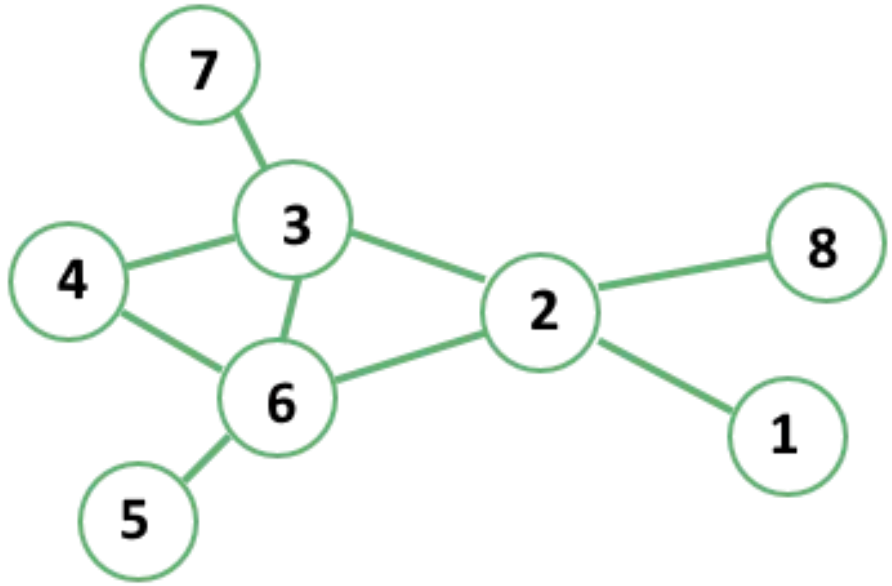
- Directed Graphs (Digraphs): Edges have a direction indicating a one-way relationship between vertices.
- Directed graphs are useful for modeling systems with a clear direction or flow of information, such as electrical circuits, computer networks, and workflow systems.
- For example, a social media platform could use a directed graph to model the flow of information between users, with edges representing messages, posts, or comments.



https://medium.com/@elfrmkr98/graphs-101-understanding-graph-data-structures-4d8252f26af

Undirected Graphs

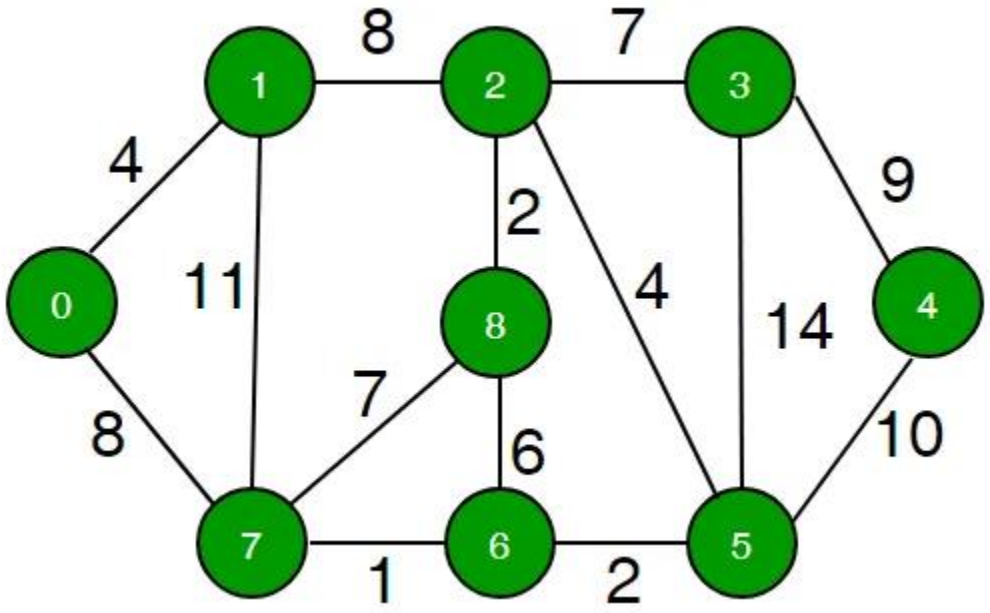
- Undirected Graphs: Edges have no direction, indicating a two-way relationship between vertices.
- Undirected graphs are useful for modeling systems where the relationship between nodes is symmetric, such as social networks, transportation networks, or biological networks.
- For example, an undirected graph could be used to model a road network, where edges represent the roads and nodes represent the intersections.



https://medium.com/@elfrmkr98/graphs-101-understanding-graph-data-structures-4d8252f26af

Weighted Graphs

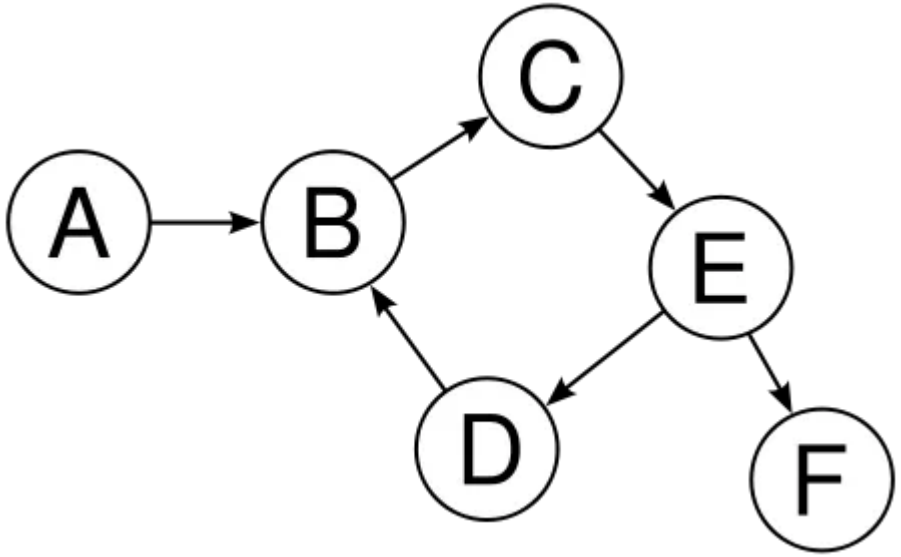
- Weighted Graphs: Edges have associated weights or values, representing the cost or distance between vertices.
- Weighted graphs are useful for modeling systems where the edges have different costs, such as transportation networks, supply chain networks, or social networks.
- For example, a weighted graph could be used to model a transportation network, where edges represent roads or rail lines, and the weights represent the travel time or distance between them.



https://medium.com/@elfrmkr98/graphs-101-understanding-graph-data-structures-4d8252f26af

Cyclic Graphs

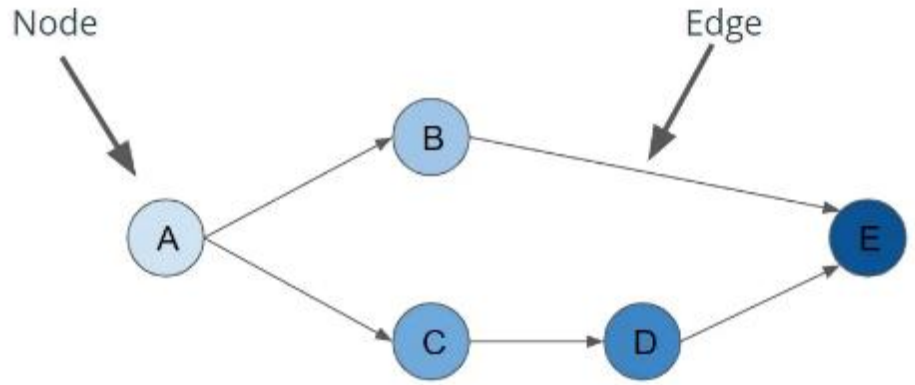
- Cyclic graph contains at least one cycle, which is a path that starts and ends at the same vertex. In other words, there is a sequence of edges that leads you back to where you started.
- Cyclic graphs are commonly used to represent systems with feedback loops, such as control systems or social networks. They can also be used to model processes that repeat themselves over time, such as the seasons of the year or the phases of the moon.



https://medium.com/@elfrmkr98/graphs-101-understanding-graph-data-structures-4d8252f26af

Acyclic Graphs

- Acyclic Graphs, also known as directed acyclic graphs (DAGs), are used in many areas of computer science and mathematics. They are particularly useful in representing dependencies between tasks or events, such as in project management or scheduling problems.
- They are also used in machine learning and artificial intelligence, for example, in Bayesian networks, which represent probabilistic relationships between variables.



https://medium.com/@elfrmkr98/graphs-101-understanding-graph-data-structures-4d8252f26af

Python Networkx

NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.

- Data structures for graphs, digraphs, and multigraphs
- Many standard graph algorithms
- Network structure and analysis measures
- Generators for classic graphs, random graphs, and synthetic networks
- Nodes can be "anything" (e.g., text, images, XML records)
- Edges can hold arbitrary data (e.g., weights, time series)
- Open source 3-clause BSD license
- Well tested with over 90% code coverage
- Additional benefits from Python include fast prototyping, easy-to-teach, and multi-platform

Aric A. Hagberg, Daniel A. Schult and Pieter J. Swart, “Exploring network structure, dynamics, and function using NetworkX”, in Proceedings of the 7th Python in Science Conference (SciPy2008), Gäel Varoquaux, Travis Vaught, and Jarrod Millman (Eds), (Pasadena, CA USA), pp. 11–15, Aug 2008

Erdős-Rényi Random Graph (1)

- The Erdős – Rényi model is one of two closely related models for generating random graphs. These models are named after Hungarian mathematicians Paul Erdős and Alfréd Rényi, who introduced one of the models in 1959.
- There are two closely related variants of the Erdős–Rényi random graph model:
- In the $G(n, M)$ model, a graph is chosen uniformly randomly from the collection of all graphs: n nodes and M edges.
- In the $G(n, p)$ model, a graph is constructed by randomly connecting n nodes with each probability p edge, independently from every other edge.
- The probability of a graph from the two models $G(n, M)$ and $G(n, p)$ is

$$\mathbb{P}(G) = p^M (1 - b)^{\binom{n}{2} - M}$$

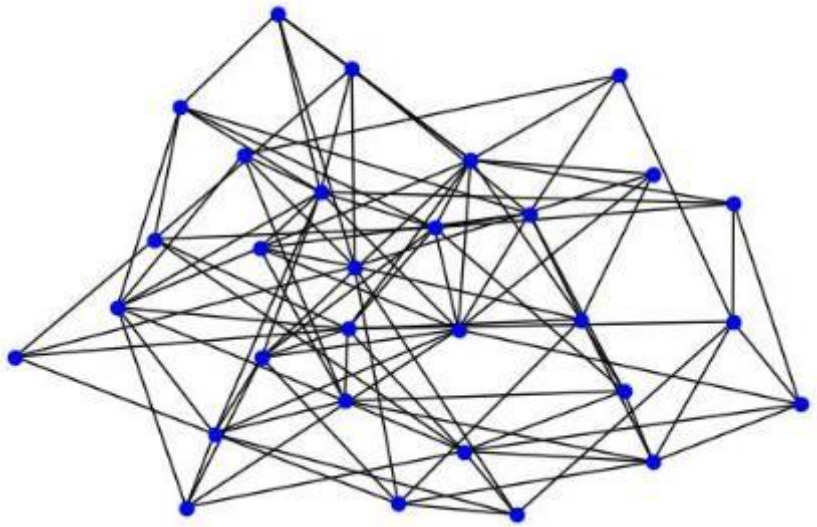
where $0 \leq p \leq 1$ and $0 \leq M \leq \binom{n}{2}$

https://en.wikipedia.org/wiki/Erd%C5%91s%E2%80%93R%C3%A9nyi_model

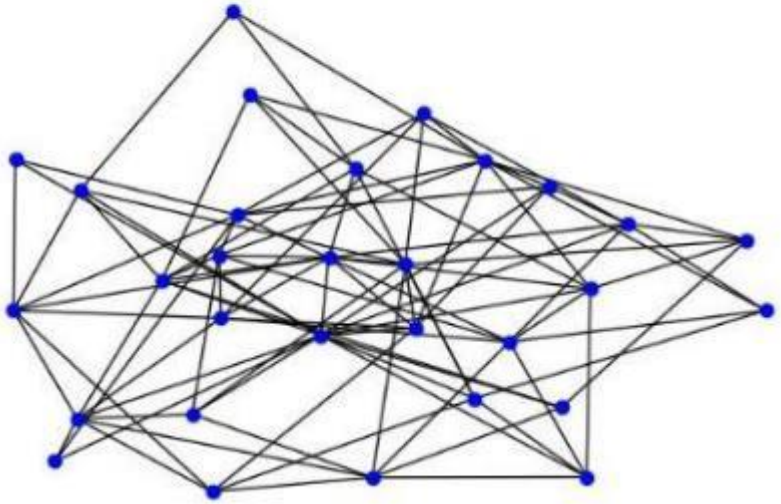
Erdős-Rényi Random Graph (2)

- Erdős-Rényi Graph with $n = 30$ and $p = 0.22$

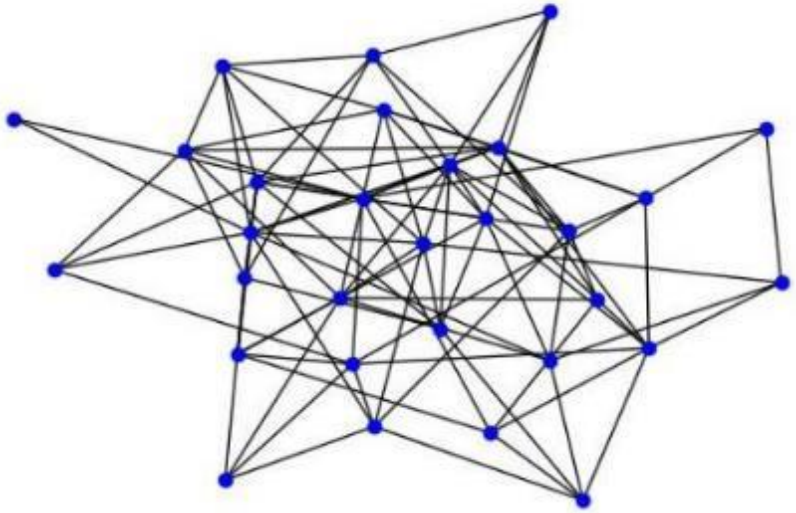
Erdos_Renyi Graph
(Nodes = 30, Edges = 104, p = 0.22)



Erdos_Renyi Graph
(Nodes = 30, Edges = 96, p = 0.22)



Erdos_Renyi Graph
(Nodes = 30, Edges = 100, p = 0.22)



Python Codes for Erdős-Rényi Random Graph (1)

```
# ********************************************************
# import modules
# ********************************************************
import networkx as nx
from os.path import join
import matplotlib.pyplot as plt
from matplotlib import rcParams
rcParams.update({"figure.autolayout": True})
# ********************************************************
# set the font family
# ********************************************************
FontSize = 16
font = {"family": "Times New Roman", "size": FontSize}
plt.rc("font", **font)  # pass in the font dict as kwargs
# ********************************************************
# create a graph
# ********************************************************
GraphName   = "Erdos_Renyi"
Nodes       = 30
p           = 0.22
Graph = nx.erdos_renyi_graph(Nodes, p)
```

```
# ********************************************************
# set graph name and title
# ********************************************************
plt.figure(GraphName)
plt.title(GraphName + " Graph")

# ********************************************************
# display the graph
# ********************************************************
nx.draw_networkx(Graph, with_labels=False, alpha=1.0, node_color="b",
font_color="w")

# ********************************************************
# set the parameters for axes
# ********************************************************
plt.tick_params(
    axis="both",       # changes apply to the x_axis
    which="both",      # both major and minor ticks are affected
    bottom="off",      # ticks along the bottom edge are off
    left="off",        # ticks along the bottom edge are off
    top="off",         # ticks along the top edge are off
    labelbottom="off",
    labelleft="off") # labels along the bottom edge are off
```

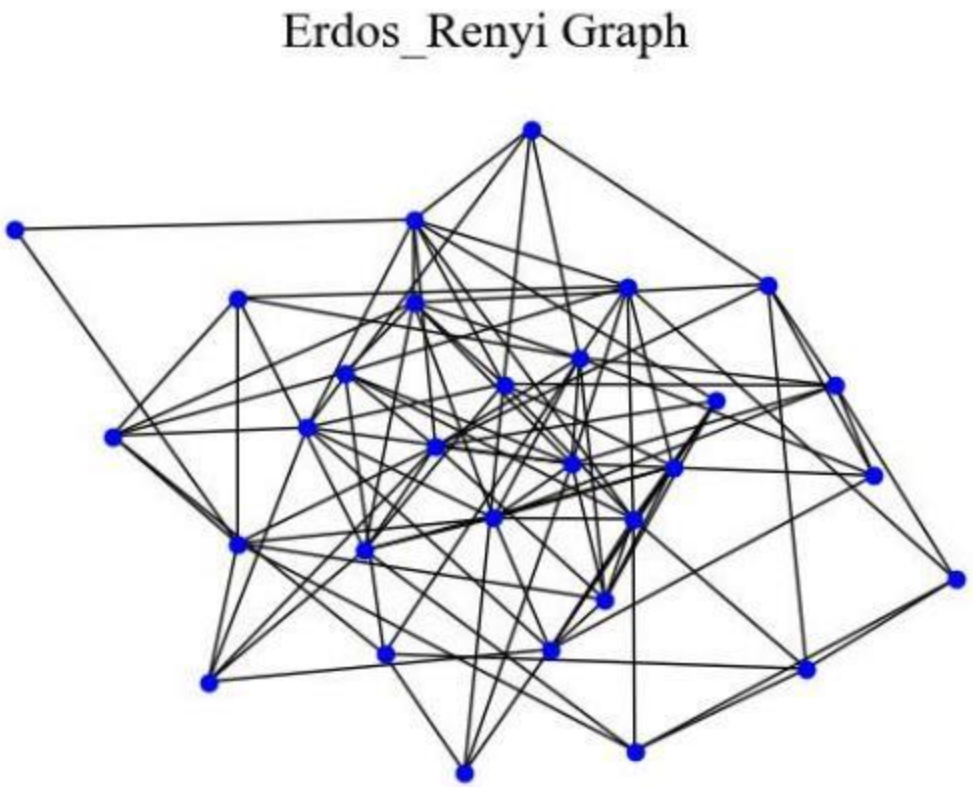
Python Codes for Erdős-Rényi Random Graph (2)

```
plt.axis("off")
plt.axis("tight")

# ********************************************************
# set the file name
# ********************************************************
FileName = (GraphName + ".jpg")
print("...Saving figure to file = <%s> ..." % FileName)

# ********************************************************
# save the figure
# ********************************************************
# plt.savefig(FileName, bbox_inches='tight')

plt.show()
```



Python Codes for Small-world Graph (1)

```
# ************************************************************
# import modules
import networkx as nx
from os.path import join
import matplotlib.pyplot as plt
from matplotlib import rcParams
rcParams.update({"figure.autolayout": True})
# ************************************************************
# set the font family
# ************************************************************
FontSize = 16
font = {"family": "Times New Roman", "size": FontSize}
plt.rc("font", **font)  # pass in the font dict as kwargs
# ************************************************************
# create a graph
# ************************************************************
GraphName   = "Newman_Watts_Strogatz"
Nodes       = 30
K           = 4
p           = 0.22
Graph = nx.newman_watts_strogatz_graph(Nodes, k, p)
```

```
# ************************************************************
# set graph name and title
# ************************************************************
plt.figure(GraphName)
plt.title(GraphName + " Graph")

# ************************************************************
# display the graph
# ************************************************************
nx.draw_networkx(Graph, with_labels=False, alpha=1.0, node_color="b",
font_color="w")

# ************************************************************
# set the parameters for axes
# ************************************************************
plt.tick_params(
    axis="both",       # changes apply to the x_axis
    which="both",      # both major and minor ticks are affected
    bottom="off",      # ticks along the bottom edge are off
    left="off",        # ticks along the bottom edge are off
    top="off",         # ticks along the top edge are off
    labelbottom="off",
    labelleft="off") # labels along the bottom edge are off
```

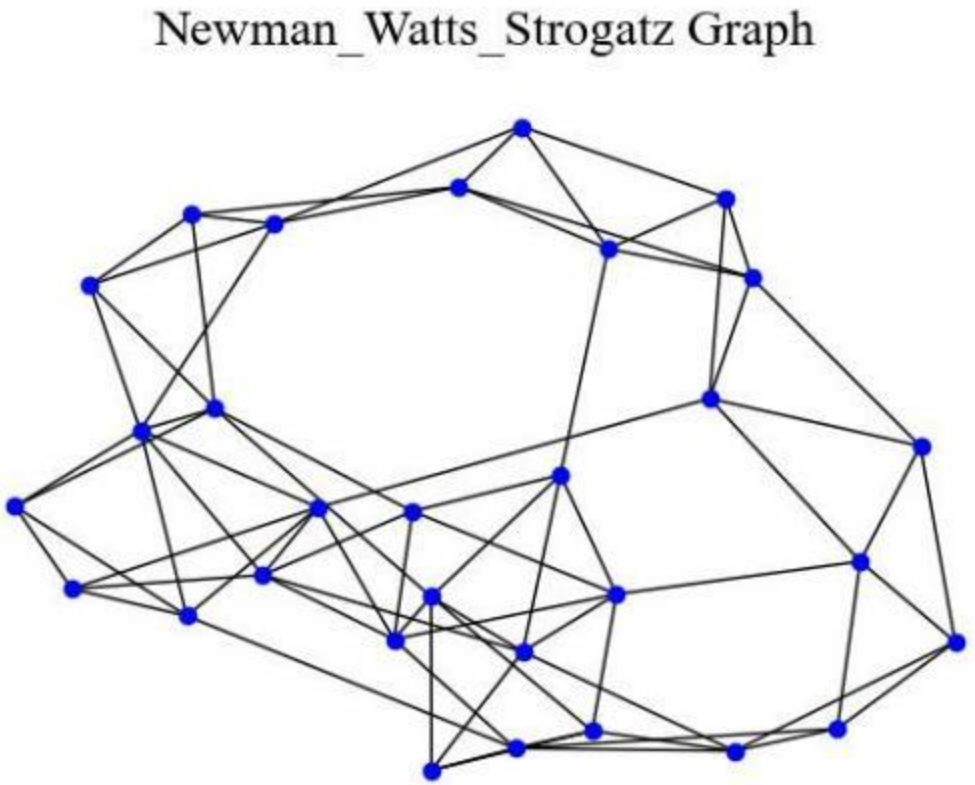
Python Codes for Small-world Graph (2)

```
plt.axis("off")
plt.axis("tight")

# ********************************************************
# set the file name
# ********************************************************
FileName = (GraphName + ".jpg")
print("...Saving figure to file = <%s> ..." % FileName)

# ********************************************************
# save the figure
# ********************************************************
# plt.savefig(FileName, bbox_inches='tight')

plt.show()
```



Small-world Power-law Networks (1)

- Both the mall-world (SW) and small-world power-law (SWPL) networks describe the fundamental characteristics of a small-world phenomenon.
- The main difference is the power-law distribution that governs the formation of connections during a rewiring process.
- The formation of SW networks is unrealistic since any connection, local or global, is associated with a cost (wire connection) to the system.
- In the SWPL model, the decay power law, $q(l) = l^{-\alpha}$, controls the connections between nodes, in which nearby nodes have a higher probability of forming links than distant nodes since short links require fewer resources than long ones.
- SWPL networks tend to have more local connections than global ones.
- The SWPL structure is a common characteristic found in physical, biological, and neural networks

Small-world Power-law Networks (2)

- The characteristics of SWPL networks depend on two hyperparameters: the locality factor α $(\alpha < D + 1)$ and the randomness factor β $(0 \leq \beta \leq 1)$.
- Different topologies will emerge from a regular $(\beta = 0)$ to a completely random$(\beta = 1.0)$ network.
- The locality α depicts the network feature whether it is highly local (α is large) or global $(\alpha = 0)$.
- D is the dimension of regular network.

- Additional parameters, removed links γ and added links δ, provides flexibility.