2-Review of Graph Theory

Complex Network Analysis Course
Reza Rezazadegan
Shiraz University, Department of Mathematics and Computer Science, Spring 2025
https://dreamintelligent.com/complex-network-analysis-2025/

Graphs

A graph G consists of a set V of vertices (or nodes) and a set E of edges or links. Each edge is of the form $\{u, v\}$ where u, v are two vertices of the graph. Such graphs are *undirected*, meaning that the relations between nodes are symmetric.

We denote the number of nodes and links in a network by N and L respectively.

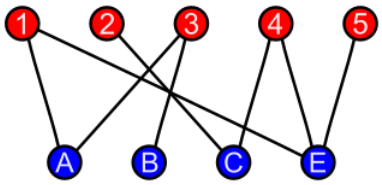
The *degree* $k(v)$ of a node v is the number of edges connected to it. Nodes with relatively high degree are called *hubs* of the network.

The *average degree* $E[k]$ of a network equal $2L/N$. (It is denoted by $< k >$ in the Network Science book.)

Most of the graphs we consider in this course are *simple graphs* meaning that there can be at most one edge between any two nodes. However in general there are *multigraphs* in which there can be more than one edge between two nodes.

Bipartite graphs

The nodes in a bipartite graph have two types, and there is no edge between the nodes of the same type.



For example the customer-product network for a recommender system is a bipartite graph.

The adjacency matrix

After choosing an enumeration of the vertices $V = \{v_1, v_2, v_3, \ldots, v_N\}$, the *adjacency matrix* A of the graph is defined by:
$A_{i,j} = 1$ if there is an edge between v_i, v_j and zero otherwise.

We have $k(v_i) = \sum_j A_{i,j}$.

For large networks with thousands or millions of edges, the adjacency matrix will be huge.

Real networks are sparse, meaning that the number of their links is proportional to N and is much smaller than the possible number of links i.e. $\binom{N}{2}$. The neural network of the worm C. elegans, has 297 neurons as nodes and 2,345 synapses as edges.

The adjacency matrix can be stored as a *sparse matrix*, however for large networks it is not efficient to use it computationally.

Directed Graphs

In directed networks, relations are asymmetric and thus edges are represented by ordered tuples (u, v) and depicted by an arrow $u \to v$.

Examples: citation networks, web,...

For directed graphs we have *in-degree* $k_{in}(v)$, i.e. the number of links coming into v, and *out-degree* $k_{out}(v)$, i.e. the number of links going out of v. Similarly, we have $N_{in}(v)$ i.e. the set of nodes which link to v (called its *predecessors*) and $N_{out}(v)$ i.e. the set of nodes that v links to (called its *successors*).
Of course the sum of the two equals degree of the node: $k(v) = k_{in}(v) + k_{out}(v)$.
We have $E[k_{in}] = E[k_{out}]$.

The adjacency matrix of a directed graph is not symmetric:
$A_{i,j} = 1$ if there is an edge from v_j towards v_i.

Image credit: Barabasi, Network Science

Every directed graph has an *underlying undirected graph* which is obtained by symmetrizing the relations.

Weighted Graphs

In the graphs considered so far, all links have the same "value" but in reality, in many networks, the edges have weights w_{ij} which quantify their role as a link between nodes v_i, v_j. For example:

- the total length of communication in a mobile communication network
- the amount of traffic in a communication or transportation network
- the number of clicks on a link between two webpages.

We can encode link weights in the adjacency matrix: $A_{i,j} = w_{i,j}$.

Adjacency lists

Adjacency lists are a way of encoding a graph in computer memory.

Adj(1)=[2,3]
Adj(2)=[1,3,4]
Adj(3)=[1,2]
Adj(4)=[2]

Note: changing the labeling (enumeration) of the nodes results in a different adjacency list. However, the output of the graph algorithms must be independent of the labeling.

Paths and connectivity

A path in an undirected network consists of a sequence of nodes

$$v_1, v_2, \ldots, v_n$$

such that there is an edge $v_i - v_{i+1}$ between v_i, v_{i+1} for each $i < n$. For directed networks, there must be a directed edge from v_i to v_{i+1}. The *length* of a path is the number of edges in it.

In a weighted graph the length of a path is defined to be the sum of the weights of the edges in it.
Whether weighted or not the length of the path $v_1, v_2, \ldots, v_n$ equals $\sum_i A_{v_i v_{i+1}}$.

A *cycle* is a path that starts from a node and goes back to it. A graph is called a *tree* if it has no cycles.

The *distance* between two nodes is the minimum length of all the paths connecting them. If there is no path connecting them, the distance is taken to be infinite. This way a network is a metric space and a topological space.

The *diameter* of a network is the maximum distance between its nodes.

We can define an equivalence relation on graph nodes based on whether there is a path connecting them. This relation partitions graph nodes into *connected components*. A graph is *connected* if it has only one connected component, or in other words, if there is a path between any pair of its nodes.

The adjacency matrix of a disconnected graph has block-diagonal form.

For directed graphs, distance is measured in the underlying undirected graph. However, even if the underlying undirected graph is connected, there may not be directed paths between pairs of nodes.

For weighted graphs, the length of a path is the sum of its weights. For example in the case of estimating time of arrival (e.g. in Google Maps), routes with more traffic have a

higher weight and can be thought of as being longer.

Relation to adjacency matrix: $A_{i,j}$ equals the number of paths of length 1 from v_i to v_j. Likewise

$$(A^2)_{i,j} = \sum_k A_{i,k}A_{k,j}$$

is the number of paths of length 2 between those two vertices. This way, $(A^l)_{i,j}$ is the number of paths of length l from v_i to v_j. The entries of the matrix

$$B = A^0 + A + A^2 + A^3 + \cdots$$

equal the number of all paths between pairs of nodes of the network. Note that the above sum equals $(I - A)^{-1}$ if all eigenvalues of A are < 1 and $(I - A)$ is invertible.

Since even in a small graph, paths can be arbitrarily long, we can introduce an *attenuation factor* $0 < \beta < 1$ and consider

$$\sum_{n=0}^{\infty}(\beta A)^n$$

instead. This way, longer paths contribute less and the number of paths between u to v can be thought of as a measure of neighborhood similarity of u and v. It is called the *Katz index* $K(u, v)$ of these two nodes. In a directed network, Katz index quantifies the *influence* of u on v. The sum $\sum_v K(u, v)$ is a measure of *centrality* of the node u. More on this in Chapter 5.

Even though the above computation is elegant, the adjacency matrix of a large network will be huge. BFS is a better option.

Breath-First Search (BFS)

BFS and DFS are algorithms for sweeping all the nodes in a graph.

BFS is a recursive algorithm and uses a queue data structure. Starting from a node u of the graph, BFS labels each node v by its distance $l(v)$ to u. Let $F(v)$ be the function (routine) that puts all the *unlabeled* neighbors of the node v in the queue and labels them with $1 + l(v)$ i.e. one plus the label of v.
BFS calls $F(u)$ then recursively takes a node v out of the queue and calls $F(v)$ until the queue becomes empty.

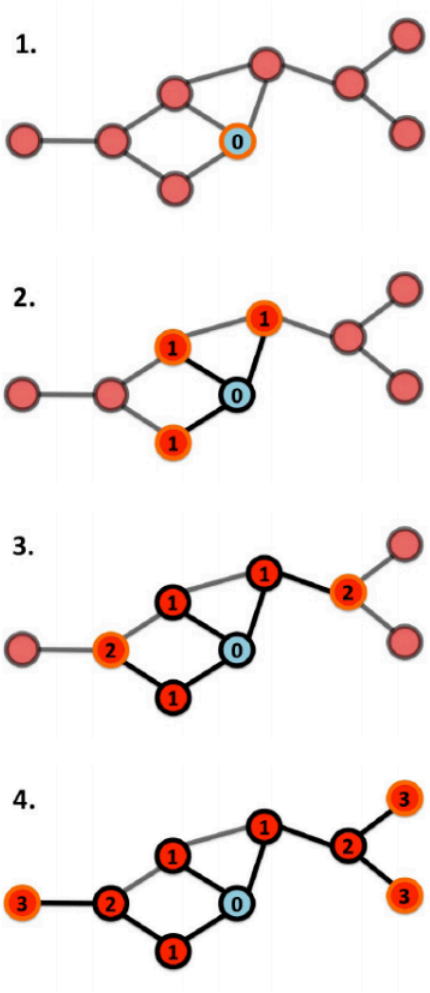


Image source: Barabasi, Network Science

```
def BFS(G, source):

    # distances from source to nodes
    dist = {v : Null for v in G}

    # Create a queue for BFS
    queue = []

    # Mark the source node as
    # visited and enqueue it
    queue.append(source)
    dist[source] = 0

    while queue:

        # Dequeue a vertex from the queue and print it
```

```
        s = queue.pop(0)
        print(s, end=" ")

        # Get all adjacent vertices of the dequeued vertex s.
        # If an adjacent has not been visited, its dist to source is
1+dist[s] and we enqueue it
        for v in G.nbhs[s]:
            if dist[v] == None:
                queue.append(v)
                dist[v] = dist[s]+1
```

Depth-First Search (DFS)

Dijkstra algorithm

BFS algorithm, as is, does not find the shortest paths between nodes for us. Note that shortest path finding can be generalized to weighted graphs.

Dijkstra algorithm is used for finding best paths in transportation and communication networks e.g. in internet routing.

```
def dijkstra(graph, source):
    # Initialize distances and visited set
    distances = {vertex: float('inf') for vertex in graph}
    distances[source] = 0
    visited = set()

    while len(visited) < len(graph):
        current_node = min((node for node in graph if node not in
visited), key=distances.get)
        visited.add(current_node)

        for (neighbor, weight) in graph[current_node].nbhs():
            new_distance = distances[current_node] + weight
            if new_distance < distances[neighbor]:
                distances[neighbor] = new_distance

    return distances
```

Induced subgraph

Let $G = (V, E)$ be a graph.
If $V' \subset V$ is a subset of the nodes of the graph, then the *subgraph induced by* V' is the graph which has V' as the vertex set, and its edges are the edges of G whose nodes are in V'. It is also called the restriction of G to V'.

Directed Acyclic Graphs (DAGs) and Topological Decomposition

A directed graph is called acyclic if it has no directed cycles. Citation networks are typically directed acyclic grapgs (DAGs).

DAGs admit a filtering on their nodes which is called *topological decomposition*. In this filtering:

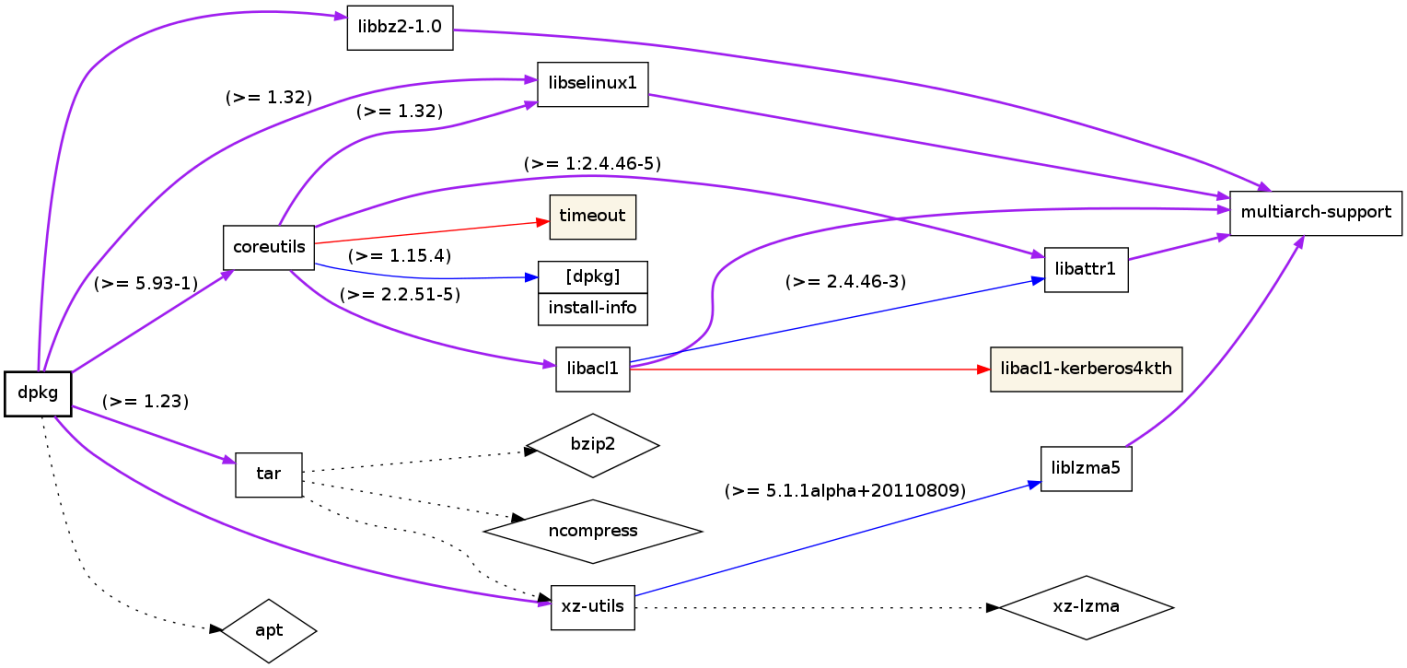
- Nodes with in-degree zero are considered level 0.
- Nodes that receive links only from nodes of level 0 are called level 1;
- Nodes that receive links *only* from nodes of level 0, 1 are considered level 2, and so on.

Note that a node receiving links from nodes of levels both n and $k < n$ is still considered to be of level n.

There is no link between nodes of each level. Why?

Applications:

- **Task scheduling**: Ensuring tasks in a project are executed in the correct order.
- **Dependency resolution**: Installing software packages with dependencies.



Dependency graph of the Linux package dpkg; produced using debtree

Exercise: Show that the adjacency matrix of a DAG is nilpotent i.e. $A^k = 0$ for some k.

Heterogeneous networks

Such as knowledge graphs, in which nodes and edges can have different types.

Spatial networks

Networks and graphs we considered so far are *abstract graphs* meaning that there nodes do not have specific locations. However there are many networks whose nodes have spatial locations.
Examples: Mobile antenna towers, transportation networks (such as road networks).

In this course we mainly focus on abstract networks, however see:

- Barthelemy, Spatial Networks, Physics Reports 2011