# CS4423-W02-1

January 23, 2025

Table of Contents

# CS4423 : Week 02 - Lecture 1 - Networks # More on Graphs, and `networkx`

Niall Madden, School of Mathematical and Statistical Sciences
University of Galway

(These notes are adapted from Angela Carnevale's work)

This notebook is at https://www.niallmadden.ie/2425-CS4423/W02/CS4423-W02-1.ipynb You can read the HTML version at https://www.niallmadden.ie/2425-CS4423/W02/CS4423-W02-1.html

This version of this notebook was written by Niall Madden, adapted from notebooks by Angela Carnevale.

## 0.1 News:

### 0.1.1 Labs

Labs start next week, and an (reintroduction) to Python. This will run: * Tuesday at 4 in AC215 (slight chance this might get moved to Tuesday at 3), and * Wednesday at 10am in CA116a.

These rooms are not labs: BYoD! (Bring Your Own Device)

### 0.1.2 Website

I now plan to post all notes to https://www.niallmadden.ie/2425-CS4423/ as well as to Canvas.

## 0.2 networkx

Last week we learned a little about the networkx package. We'll return to that now, while also revisiting some key ideas about graphs.

As ever, we'll start with importing the networkx module, as well as numpy: more about that later. And we'll define the opts option dictionary.

```
[ ]: import networkx as nx
     import numpy as np
     opts = { "with_labels": True, "node_color": 'y' } # show labels; yellow noodes
```

To create a graph with nodes 1, 2, 3, 4, 5, and edges between all even and odd labelled nodes:

```
[ ]: K32 = nx.Graph() # makes THE empty graph
     K32.add_edges_from([(1, 2), (1,4), (2,3), (2,5), (3,4), (4,5)])
```

We'll later learn this is the graph $K_{3,2}$. Now draw it:

```
[ ]: nx.draw(K32, **opts)
```

We can also be lazy, and just give 2-letter strings for the edges: this implicitly defines the nodes too.

```
[ ]: K33 = nx.Graph(["A1", "A2", "A3", "B1", "B2", "B3", "C1", "C2", "C3"])
     nx.draw(K33, **opts)
```

### 0.2.1 Check basic properties:

```
[ ]: print(f"K33 has {K33.number_of_nodes()} nodes and {K33.number_of_edges()}␣
       ↪edges")
```

```
[ ]: print(f"This is the same as saying K33 order {K33.order()} and size {K33.
       ↪size()}")
```

To list the nodes and edges (as lists)

```
[ ]: list(K33.nodes)
```

```
[ ]: list(K33.edges)
```

A **loop** over a graph G will effectively loop over G's nodes. As an example, (recall?) that the **degree** of a node is the number of edges incident to it (or, if you prefer, the number of neighbours).

```
[ ]: for node in K33:
         print(f"node {node} has neighbours {list(K33.neighbors(node))}")
```

### 0.2.2 Adding and removing nodes and edges

We say that * `G.add_node('v')` adds a node to $G$ called 'v' * `G.add_nodes_from([2, 3, 5])` adds all the nodes from a list * `G.add_nodes_from(H])` adds all the nodes from Graph $H$ to Graph $G$ * `G.add_edge('x','y')` add edge from Node $x$ to Node $y$, adding one or both nodes, if needed. * `G.add_edges_from([(1,5), (2,5), (3,5)])` add edges from a list * `G.add_edges_from(H.edges)` add edges from another graph * `G.remove_edge('x','y')` remove edge from $x$ to $y$, but keep the nodes * `G.remove_node('x')` remove node $x$ and any edge it was incident to.

## 0.3 Neighbours and degree

(As we've seen) * The **neighbours** of a node are those that it shares an edge with; * If nodes $a$ and $b$ are neighbours, we say they are **adjacent**. * the **degree of a node** is the number of edges incident to it (or, if you prefer, the number of neighbours).

Let's look at an example:

[ ]:
```
Edges1 = [('Aoife', 'Brian'), ('Aoife', 'Ciara'), ('Aoife', 'Daire'), ('Aoife',␣
 ↪'Ella'),
          ('Aoife', 'Finn'), ('Brian', 'Ciara'), ('Brian', 'Finn'), ('Ciara',␣
 ↪'Daire'),
          ('Daire', 'Ella'), ('Ella', 'Finn')  ]
G1 = nx.Graph(Edges1)
```

[ ]:
```
nx.draw(G1, **opts)
```

This is example, which is known as a *wheel graph* is chosen because it exhibits a famous concept in Network Science: *The Friendship Paradox*: your friends (probably) have more friends, on average, than you do!

Explanation:

[ ]:
```
#pos = nx.nx_agraph.graphviz_layout(G1)
#nx.draw(G1, pos=pos,**opts)
```

## 0.4 Important Graphs

In this section we'll discuss some important examples of graphs, which we'll return to later as key examples of networks. These include * Complete Graphs * Bipartite and complete bipartite graphs * Path graphs

### 0.4.1 Complete Graphs

The **complete graph** on a vertex set $X$ is the graph with edge set all of $\binom{X}{2}$. That is: every node is a neighbour of every other node. It is denoted $K_n$ where $n = |X|$. E.g., if $X = \{0, 1, 2, 3\}$, then $K_4$ ("the complete graph on 4 nodes") has edges $E = \{01, 02, 03, 12, 13, 23\}$.

[ ]:
```
nodes = range(4)
list(nodes)
```

```
[ ]: E4 = [(x, y) for x in nodes for y in nodes if x < y]
     print(E4)
```

```
[ ]: K4 = nx.Graph(E4)
     nx.draw(K4, **opts)
```

While it is somewhat straightforward to find all 2-element subsets of a given set $X$ with a short `python` program, it is probably more convenient (and possibly efficient) to use a function from the `itertools` package for this purpose.

```
[ ]: from itertools import combinations
     nodes5 = range(5)
     combinations(nodes5, 2)
```

```
[ ]: print(list(combinations(nodes5, 2)))
```

```
[ ]: K5 = nx.Graph(combinations(nodes5, 2))
```

```
[ ]: nx.draw(K5, **opts)
```

`networkx` has a built-in function to create complete graphs: `complete_graph` [doc]

```
[ ]: nx.draw(nx.complete_graph("NETWORKS"), **opts)
```

```
[ ]: nx.draw(nx.complete_graph(22), **opts)
```

### 0.4.2 Bipartite Graphs

A graph is **bipartite** if we can divide the node set, $X$, into two subsets $X_1$ and $X_2$ such that * $X_1 \cap X_2 = \emptyset$ (the sets have no edge in common) * $X_1 \cup X_2 = X$ * For any edge $(u_1, u_2)$ we have $u_1 \in X_1$ and $u_2 \in X_2$. That is we only ever have edges between nodes from different sets.

Such graphs are very common in Network Science, where nodes in the network represent two different types of entities. For example, we might have a graph where nodes represent students and modules, with edges between students and modules they are enrolled in, often called an **affiliation network**.

```
[ ]: Edges2 = [('Aoife', 'CS4423'), ('Aoife', 'CS319'), ('Aoife', 'MA432'),
               ('Brian', 'CS4423'), ('Brian', 'CS319'),
               ('Ciara', 'CS319'),  ('Ciara', 'MA432'),
               ('Daire', 'MA432')  ]
     G2 = nx.Graph(Edges2)
     nx.draw(G2, **opts)
```

Somehow that previous graph did not catch the essence of the network: there are two different types of node. We could make that clearer, by colouring the nodes. Here we'll do it manually (later, automatically).

```
[ ]: print(G2.nodes)
     color_list= ['c','y','y','y','c', 'c','c'] # y=yellow; c=cyan
```

```
nx.draw(G2, node_color=color_list, with_labels=True)
```

### 0.4.3 Complete Bipartite Graphs

A **complete bipartite graph** is a particular bipartite graph where there is an edge between every node in $X_1$ and every node in $X_2$. Such graphs are denoted $K_{m,n}$ where $|X_1| = m$ and $|X_2| = n$. (We met $K_{2,2}$ and $K_{3,3}$ earlier).

As usual, there is a built-in generator: `complete_bipartite_graph` [doc]

[ ]:
```
K33 = nx.complete_bipartite_graph(3,3)
nx.draw(K33,**opts)
```

### 0.4.4 Path Graphs

The **Path Graph** with $n$ nodes, denoted $P_n$, is one where two nodes have degree 1, and the other $n-2$ have degree 2:

[ ]:
```
P4 = nx.Graph(["ab", "bc", "cd", "de"])
nx.draw(P4)
```

The built-in `nerworkx`generator is called `path_graph` [doc]

[ ]:
```
P10 = nx.path_graph(10)
nx.draw(P10)
```

## 0.5 Exercises

1. For what values of $n$ is $K_n$ bipartite?
2. For what values of $m$ and $n$ is $K_{m,n}$ bipartite?
3. For what values of $n$ is $P_n$ bipartite?
4. (Based on Q2(a) from the 2023/2024 CS4423 Exam paper) Let $G$ be the graph on the set of nodes $\{1, 2, 3, 4, 5, 6\}$ with edges $1-2$, $1-3$, $2-$4, $3-4$, $3-6$, $4-5$, $4-6$. Draw the graph $G$. Is $G$ bipartite? Justify your answer. (Note saying $a-b$ is an edge in $G$ is the same as saying $(a, b)$ is an element of its edge set).
5. (Based on Q1(b) of the 2019/2019 CS4423 paper) At a party with $n = 5$ people, some people know each other already while others don't. Each of the 5 guests is asked how many friends they have at this party. Two report that they have one friend each. Two other guests have two friends each, and the fifth guest has three friends at the party. Understanding friendship as a symmetric relation, is this network possible? Why, or why not? (*Hint: recall that the sum of all node degrees is twice the number of edges in the graph*).

Finished here Wednesday