CS4423 - Networks

Prof. Götz Pfeiffer
School of Mathematics, Statistics and Applied Mathematics
NUI Galway

4. Small Worlds

Lecture 15: Characteristic Path Length and Clustering Coefficient

Many real world networks are **small world networks**, where most pairs of nodes are only a few steps away from each other.

More precisely, a network is a small world network if it has

1. a small **average shortest path length** (scaling with $\log n$, where n is the number of nodes), and
2. a high **clustering coefficient**.

Random networks do have a small average shortest path length, but not a high clustering coefficient. This observation justifies the need for a different model of random networks, if they are to be used to model the clustering behavior of real world networks.

In [1]:
```
import networkx as nx
import matplotlib.pyplot as plt
```

Characteristic Path Length

Let $\mathcal{D} = (d_{ij})$ be the **distance matrix** of a connected graph $G = (X, E)$, whose entry d_{ij} is the length of the shortest path from node $i \in X$ to node $j \in X$. (Hence $d_{ii} = 0$ for all i.)

Definition. Let $G = (X, E)$ be a connected graph. * The **eccentricity** e_i of a node $i \in X$ is the maximum distance between i and any other vertex in G,

$$e_i = \max_j d_{ij}.$$

* The **graph radius** R is the minimum eccentricity,

$$R = \min_i e_i.$$

* The **graph diameter** D is the maximum eccentricity,

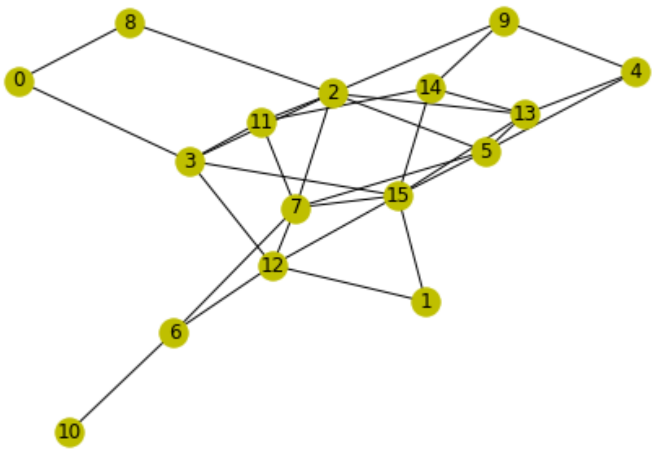
$$D = \max_i e_i.$$

* The **characteristic path length** L of G is the average distance between pairs of distinct nodes,

$$L = \frac{1}{n(n-1)} \sum_{i \neq j} d_{ij}.$$

- The characteristic path length of a random graph $G(n, m)$, or $G(n, p)$ is
$$L = \frac{\ln n}{\ln \bar{k}}$$
(... express $\bar{k}$ in terms of n, m, p ...)

In [2]:
```
n, m = 16, 32
while True:
    G = nx.gnm_random_graph(n, m)
    if nx.is_connected(G):
        break
nx.draw(G, with_labels=True, node_color='y')
```



In [3]:
```
dist = dict(nx.shortest_path_length(G))
dist = [[dist[i][j] for j in range(n)] for i in range(n)]
```

In [4]:
```
dist
```

Out[4]:
```
[[0, 3, 2, 1, 4, 3, 3, 3, 1, 3, 4, 2, 2, 3, 3, 2],
 [3, 0, 3, 2, 3, 2, 2, 2, 4, 3, 3, 3, 1, 2, 2, 1],
 [2, 3, 0, 1, 2, 1, 2, 1, 1, 1, 3, 1, 2, 1, 2, 2],
 [1, 2, 1, 0, 3, 2, 2, 2, 2, 2, 3, 1, 1, 2, 2, 1],
 [4, 3, 2, 3, 0, 1, 3, 2, 3, 1, 4, 3, 3, 1, 2, 2],
 [3, 2, 1, 2, 1, 0, 2, 1, 2, 2, 3, 2, 2, 1, 2, 1],
 [3, 2, 2, 2, 3, 2, 0, 1, 3, 3, 1, 2, 1, 3, 3, 2],
 [3, 2, 1, 2, 2, 1, 1, 0, 2, 2, 2, 1, 1, 2, 2, 1],
 [1, 4, 1, 2, 3, 2, 3, 2, 0, 2, 4, 2, 3, 2, 3, 3],
 [3, 3, 1, 2, 1, 2, 3, 2, 2, 0, 4, 2, 3, 2, 1, 2],
 [4, 3, 3, 3, 4, 3, 1, 2, 4, 4, 0, 3, 2, 4, 4, 3],
 [2, 3, 1, 1, 3, 2, 2, 1, 2, 2, 3, 0, 2, 2, 1, 2],
 [2, 1, 2, 1, 3, 2, 1, 1, 3, 3, 2, 2, 0, 2, 2, 1],
 [3, 2, 1, 2, 1, 1, 3, 2, 2, 2, 4, 2, 2, 0, 1, 1],
 [3, 2, 2, 2, 2, 2, 3, 2, 3, 1, 4, 1, 2, 1, 0, 1],
 [2, 1, 2, 1, 2, 1, 2, 1, 3, 2, 3, 2, 1, 1, 1, 0]]
```

In [5]:
```
eccentricity = [max(d) for d in dist]
eccentricity
```

Out[5]:
```
[4, 4, 3, 3, 4, 3, 3, 3, 4, 4, 4, 3, 3, 4, 4, 3]
```

In [6]:
```
nx.eccentricity(G)
```

Out[6]:
```
{0: 4,
 1: 4,
 2: 3,
 3: 3,
 4: 4,
 5: 3,
 6: 3,
 7: 3,
 8: 4,
 9: 4,
 10: 4,
 11: 3,
 12: 3,
 13: 4,
 14: 4,
 15: 3}
```

In [7]:
```
radius = min(eccentricity)
diameter = max(eccentricity)
radius, diameter
```

Out[7]:
```
(3, 4)
```

In [8]:
```
sum([sum(d) for d in dist]) / n / (n - 1)
```

Out[8]:
```
2.1166666666666667
```

In [9]:
```
nx.average_shortest_path_length(G)
```

Out[9]:
```
2.1166666666666667
```

In [10]:
```
from math import log
kbar = sum(dict(G.degree()).values()) / n
log(n) / log(kbar)
```

Out[10]:
```
2.0
```

Definition (Small-world behaviour). A network $G = (X, E)$ is said to exhibit a **small world behaviour** if its characteristic path length L grows proportionally to the logarithm of the number n of nodes of G:

$$L \sim \ln n.$$

In this sense, the ensembles $G(n, m)$ and $G(n, p)$ of random graphs do exhibit small workd behavior (as $n \to \infty$).

Clustering

Small world networks contain many triangles: it is not uncommon that a friend of one of my friends is my friend, too. This **degree of transitivity** can be measured in several different ways.

Definition (Graph transitivity). A **triad** is a tree of 3 nodes or, equivalently, a graph consisting of 2 adjacent edges (and the nodes they connect). The transitivity T of a graph $G = (X, E)$ is the proportion of **transitive** triads, i.e., triads which are subgraphs of **triangles**:

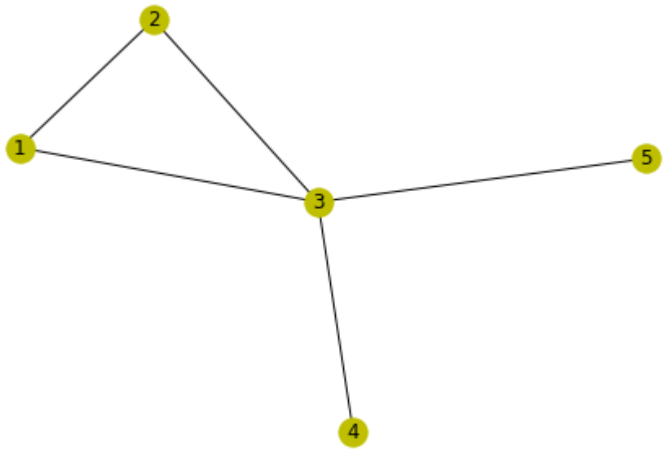
$$T = \frac{3n_{\Delta}}{n_{\wedge}},$$

where n_{Δ} is the number of triangles in G, and $n_{\wedge}$ is the number of triads.

By definition, $0 \leq T \leq 1$.

Example.

In [11]:
```
G = nx.Graph(((1,2), (2,3), (3,1), (3,4), (3,5)))
nx.draw(G, with_labels=True, node_color='y')
```



The function `nx.triangles(G)` returns a `python` dictionary reporting for each node of the graph `G` the number of triangles it is contained in.

In [12]:
```
print(nx.triangles(G))
```

```
{1: 1, 2: 1, 3: 1, 4: 0, 5: 0}
```

Overall, each triangle in `G` is thus accounted for 3 times, once for each of its vertices. The following sum determines this number $3n_{\Delta}$.

In [13]:
```
triple_nr_triangles = sum(nx.triangles(G).values())
print(triple_nr_triangles)
```

```
3
```

The number $n_{\wedge}$ of triads in `G` can be determined from the graph's degree sequence, as each node of degree k is the central node of exactly $\binom{k}{2}$ triads. (Why?)

In [14]:
```
print(G.degree())
print({k : v * (v-1) // 2 for k, v in dict(G.degree()).items()})
nr_triads = sum([v * (v-1) // 2 for v in dict(G.degree()).values()])
print(nr_triads)
```

```
[(1, 2), (2, 2), (3, 4), (4, 1), (5, 1)]
{1: 1, 2: 1, 3: 6, 4: 0, 5: 0}
8
```

The transitivity T of `G` is the quotient of these two quantities, $T = 3n_{\Delta}/n_{\wedge}$.

In [15]:
```
print(triple_nr_triangles / nr_triads )
print(nx.transitivity(G))
```

```
0.375
0.375
```

Definition (Clustering coefficient). For a node $i \in X$ of a graph $G = (X, E)$, denote by G_i the subgraph induced on the neighbours of i in G, and by $m(G_i)$ its number of edges. The **node clustering coefficient** c_i of node i is defined as

$$c_i = \begin{cases} \binom{k_i}{2}^{-1} m(G_i), & k_i \geq 2, \\ 0, & \text{else.} \end{cases}$$

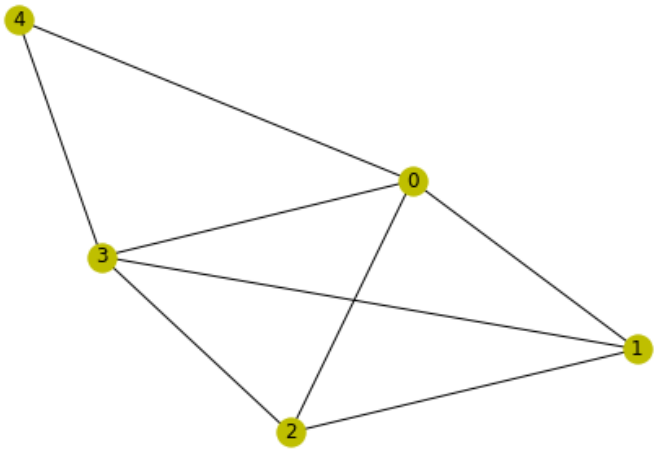
The **graph clustering coefficient** C of G is the average node clustering coefficient,

$$C = \langle c \rangle = \frac{1}{n} \sum_{i=1}^{n} c_i.$$

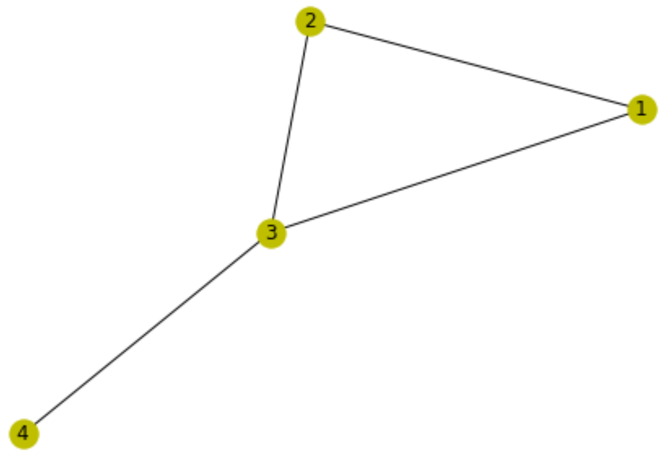
By definition, $0 \leq c_i \leq 1$ for all nodes $i \in X$, and $0 \leq C \leq 1$.

Example.

In [16]:
```
G = nx.Graph([(0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (2,3), (3,4)])
nx.draw(G, with_labels=True, node_color='y')
```



In [17]:
```
N = nx.neighbors(G, 0)
S = G.subgraph(list(N))
nx.draw(S, with_labels=True, node_color='y')
```



In [18]:
```
nS = S.number_of_nodes()
nS_choose_2 = nS * (nS - 1) // 2
mS = S.number_of_edges()
print(nS, mS, mS / nS_choose_2 )
```

```
4 4 0.6666666666666666
```

In [19]:
```
nx.clustering(G)
```

Out[19]:
```
{0: 0.6666666666666666, 1: 1.0, 2: 1.0, 3: 0.6666666666666666, 4: 1.0}
```

In [20]:
```
nx.average_clustering(G)
```

Out[20]:
```
0.8666666666666666
```

- The clustering coefficient of a $G(n, p)$ random graph is
$$C = p.$$
Note that when $p(n) = \bar{k}/n$ for a fixed expected average degree $\bar{k}$ then $C = \bar{k}/n \to \infty$ for $n \to \infty$: in large random graphs the number of triangles is negligible.
- In real world networks, one often observes that $C/\bar{k}$ does not depend on n (as $n \to \infty$?)

Exercises

1. Design an experiment with random graphs to verify the predicted characteristic path length.
2. Design an experiment with random graphs to verify the predicted graph clustering coefficient.

In [ ]:
```
```