Lecture 10 Small World Phenomenon and Paths

July 14, 2021

1 Lecture 10: Small World and Hamiltonians

1.1 Small-World Phenomenon

Before we state the small-world phenomenon let us start with a couple of useful parameters in graphs.

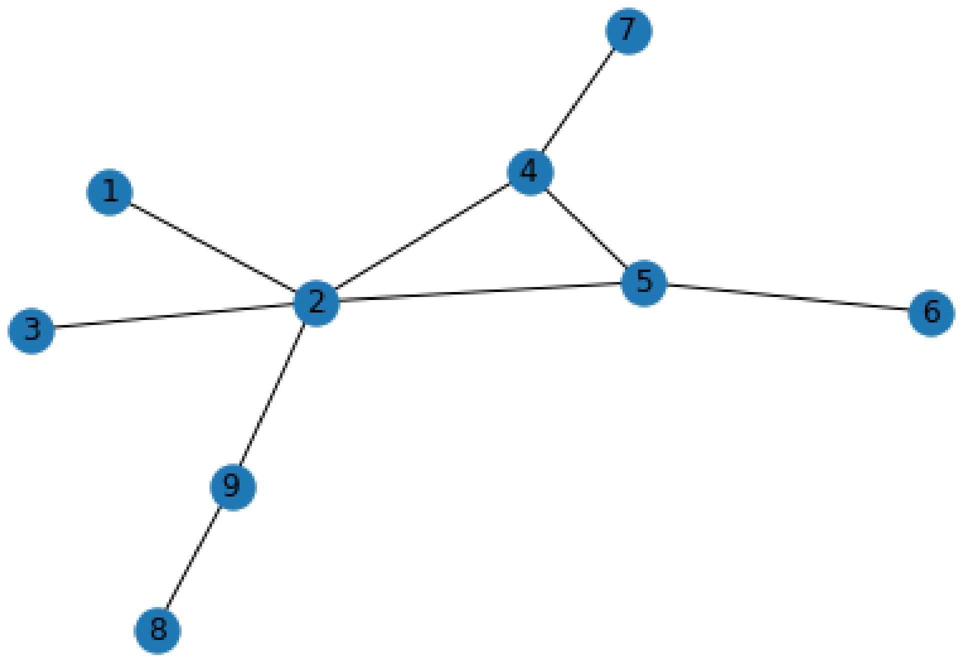
1. Characteristic Path Length (L): For a graph G, the characteristic path length (L) is the number of edges in the shortest path between two vertices, averaged over all pairs of vertices.

   Recall that this is similar to the average path length from last class.

2. Clustering Coefficient (C): Suppose a vertex v has k neighbors, then we ask the question, how connected are these neighbors among each other? Potentially these k neighbors can all be connected to each other and have $\binom{k}{2}$ edges amongst them. Then C_v is the fraction of these edges that actually exist. Furthermore, C is the average of C_v over the entire graph.

   Let us illustrate this with an example:

[1]:
```
import networkx as nx
G = nx.Graph()

G.add_nodes_from([1,2,3,4,5,6,7,8,9])
G.add_edges_from([(1,2), (2,3), (2,4), (2,5), (4,5), (4,7), (8,9), (2,9),␣
 ↪(5,6)])
nx.draw(G, with_labels=True)
```



Suppose we select vertex 2. Then vertex 2 has neighbors 1,3,4,5,9. However, these vertices are not connected among each other except vertex 4 and 5. Therefore

$$C_2 = \frac{\text{number of edges among neighbors}}{\text{maximum number of edges possible}} = \frac{1}{\binom{5}{2}} = \frac{1}{10}$$

Then the clustering coefficient for the entire graph is the average of C_v averaged over the entire graph.

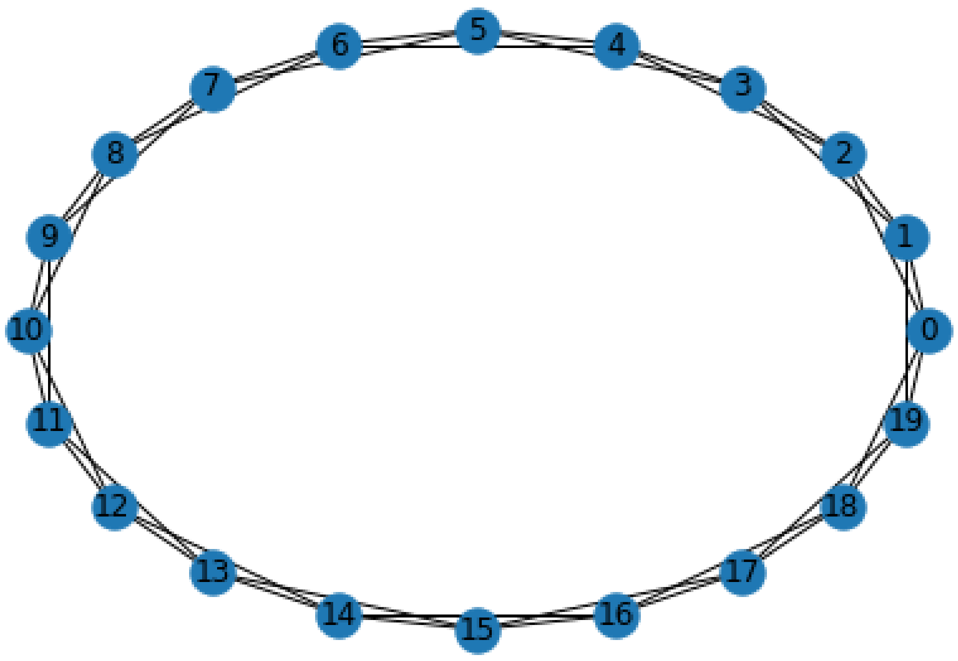
1.1.1 Interpretation for friendship networks

For friendship networks: * L is the average number of friendships in the shortest chain connecting two people. * C_v measures the extent to which friends of v are also friends of each other, ie the closeness among friends of v. C is the average of C_v over the entire graph.

Let us demonstrate the small world phenomenon now:

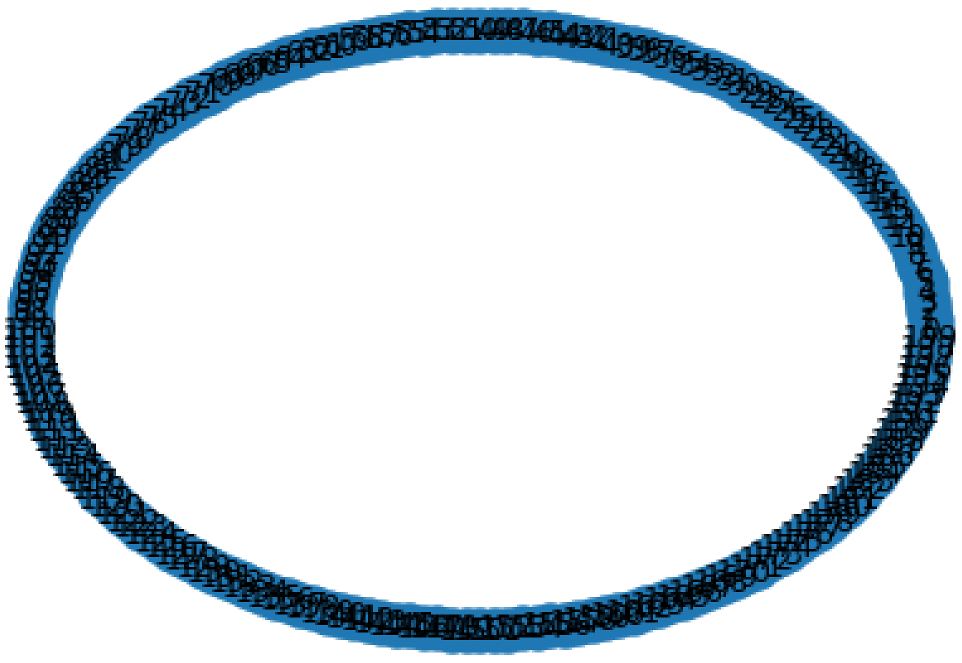
We start with a graph, that represents a community where people know their neighbors. We will explain the all the notation as we move ahead, suppose a group of 20 people, each person knows 4 of their neighbors.

[2]:
```
G = nx.watts_strogatz_graph(20, 4, 0)
nx.draw_circular(G, with_labels=True)
```



Let us now model a population of 200 people who start of with knowing 4 of their neighbors and then evolve.

[3]:
```
G = nx.watts_strogatz_graph(200, 4, 0)
nx.draw_circular(G, with_labels=True)
```



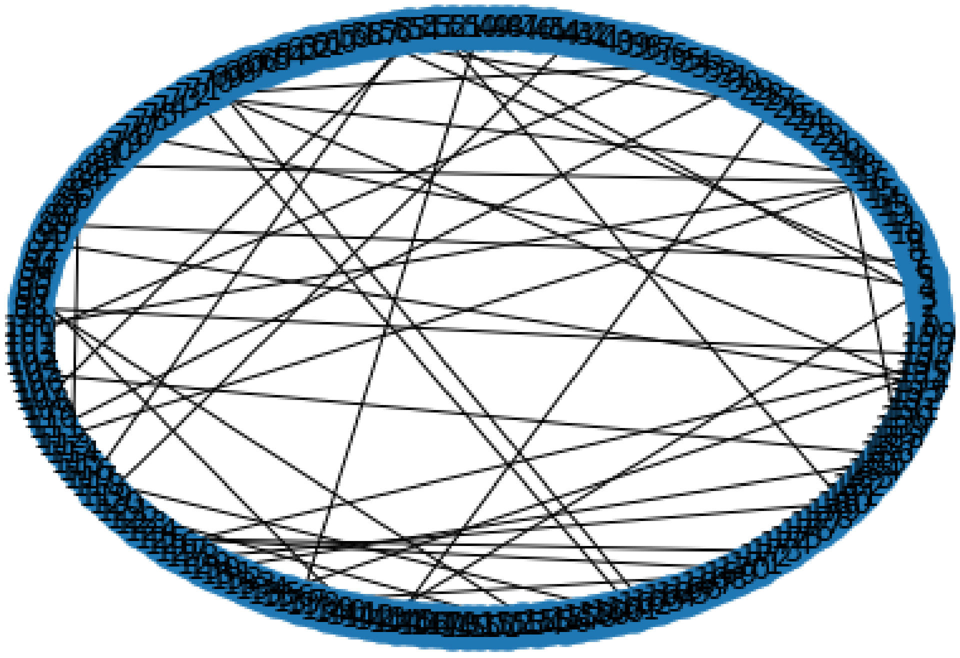
Let us compute L and C.

```
[4]: def detailsG(G):
         L = nx.average_shortest_path_length(G)
         C = nx.average_clustering(G)

         print("Characteristic Path Length (L): ", L)
         print("Clustering Coefficient (C): ", C)

     detailsG(G)
```

```
Characteristic Path Length (L):  25.376884422110553
Clustering Coefficient (C):  0.5
```

Now let us evolve this graph, by allowing people to create new friendships. We select and edge u, v and with probability p say 0.1 or smaller, we remove the edge and instead join u to not just its neighbor, but to any vertex possible.

```
[5]: G = nx.watts_strogatz_graph(200, 4, 0.1)
     nx.draw_circular(G, with_labels=True)
```



Observe that most connections are intact, only some have been rewired to an arbitrary vertex selected at random. This can model people moving around their neighborhood and making new connections, although very slowly, they create a new connection only 1/10th the time.

Now let us see what L and C look like.

[6]:
```
detailsG(G)
```

```
Characteristic Path Length (L):  5.552914572864322
Clustering Coefficient (C):  0.34680952380952407
```

Do you notice what happened?

Even though we allowed new connections only 1/10th of the time (with 9/10 probability the connections remained the same), yet the characteristic path length (average number of edges connecting two arbitrary people) dropped from 25.38 to 5.5 !

However, the clustering coefficient (how close friendship circles are) dropped only slightly from 0.5 to 0.35

Can you explain the reason?

1.2 A larger experiment:

Let us now take do a larger experiment. This is inspired from a Nature paper by Watts and Strogatz. The paper has 43000+ citations and is the most influential work by either authors.

We start of with a population of 1000 individuals, where each person knows 10 people around them. Then we provide a tiny probability (say 0.1, 0.01) to allow them to evolve creating new frienships. We study L and C

[7]:
```
n = 1000
k = 10
listp = [0, 0.001, 0.003, 0.005, 0.01, 0.03, 0.05, 0.1, 0.3, 0.5, 1]
dictG = {}
dictL = {}
dictC = {}
dictD = {}

# create a graph for each mixing probability
for p in listp:
    dictG[p] = nx.watts_strogatz_graph(1000, 10, p)
```

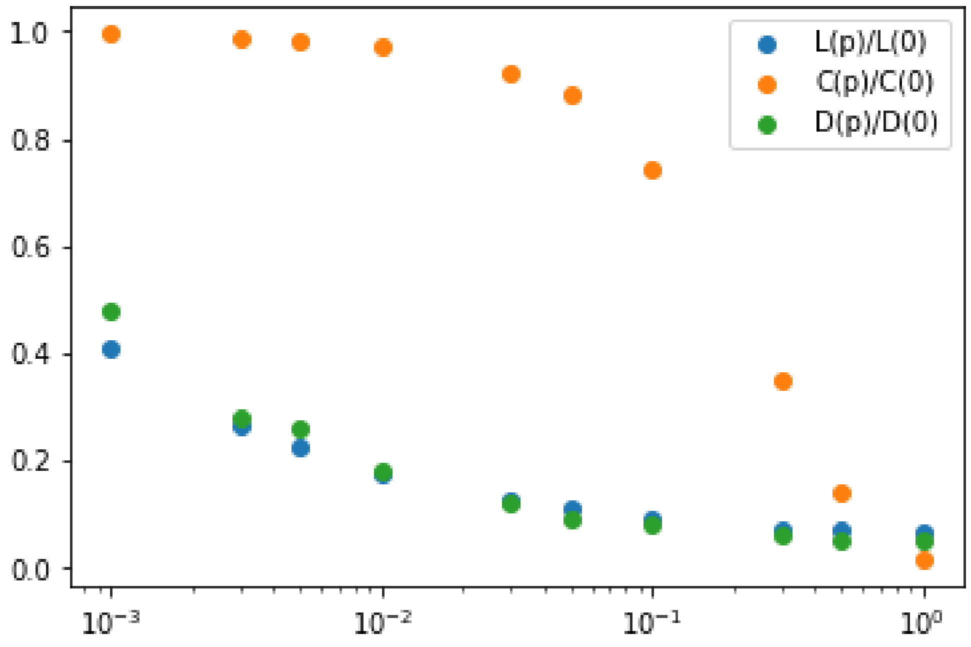
[8]:
```
# calculate parameters (note this can be slow for large n)
for p in listp:
    dictL[p] = nx.average_shortest_path_length(dictG[p])
    dictC[p] = nx.average_clustering(dictG[p])
    dictD[p] = nx.diameter(dictG[p])
```

[9]:
```
print(dictL)
print(dictC)
print(dictD)
```

```
{0: 50.450450450450454, 0.001: 20.499955955955954, 0.003: 13.357021021021021,
0.005: 11.361461461461461, 0.01: 8.842304304304305, 0.03: 6.291513513513514,
0.05: 5.417025025025025, 0.1: 4.467309309309309, 0.3: 3.60422022022022, 0.5:
3.3792692692692694, 1: 3.268984984984985}
{0: 0.6666666666666636, 0.001: 0.6635904040404009, 0.003: 0.6593444444444414,
0.005: 0.655389610389606, 0.01: 0.6475050505050471, 0.03: 0.61394271284271,
0.05: 0.5873476245976232, 0.1: 0.49412108447108427, 0.3: 0.23388535908535896,
0.5: 0.09181034684270026, 1: 0.009340954960072617}
{0: 100, 0.001: 48, 0.003: 28, 0.005: 26, 0.01: 18, 0.03: 12, 0.05: 9, 0.1: 8,
0.3: 6, 0.5: 5, 1: 5}
```

Let us now plot $C(p)/C(0)$ and similarly for L, D. This will show the ratio of C, L, D after mix. Remember p = 0 means no mixing.

[10]:
```
import matplotlib.pyplot as plt

listL = list(dictL.values())
listC = list(dictC.values())
listD = list(dictD.values())

plt.xscale("log")
plt.scatter(listp, [i/listL[0] for i in listL], label='L(p)/L(0)')
plt.scatter(listp, [i/listC[0] for i in listC], label='C(p)/C(0)')
plt.scatter(listp, [i/listD[0] for i in listD], label='D(p)/D(0)')
plt.legend()
plt.show()
```



The sudden drop in $L(p)$ and $D(p)$ whereas $C(p)$ being practically unchanged is called the Small World Phenomenon.

As we allow mixing and a tiny probability to make friends outside the frienship circles, the closeness among friendship circles remains the same, however the average path length between two people (via friendships) drastically reduces. So does the diameter which represents the longest distance between 2 people.

Read the paper "Collective Dynamics of small world networks" by Watts and Strogatz (It is 3 pages!). The paper is available here

1.3 Hamiltonians on graphs

Let us now discuss a new topic.

All graphs we discuss now are connected.

Question 1: Given a graph, can you find a path (a sequence of connected edges) that visits every vertex exactly once?

Question 2: Can you find a cycle (start and end at the same vertex) that visits every vertex exactly once? (except of course the starting point which is also visited at the end of the cycle).

A path answering question 1 is known as a **Hamiltonian Path** and

A path answering question 2 is known as a **Hamiltonian Cycle**

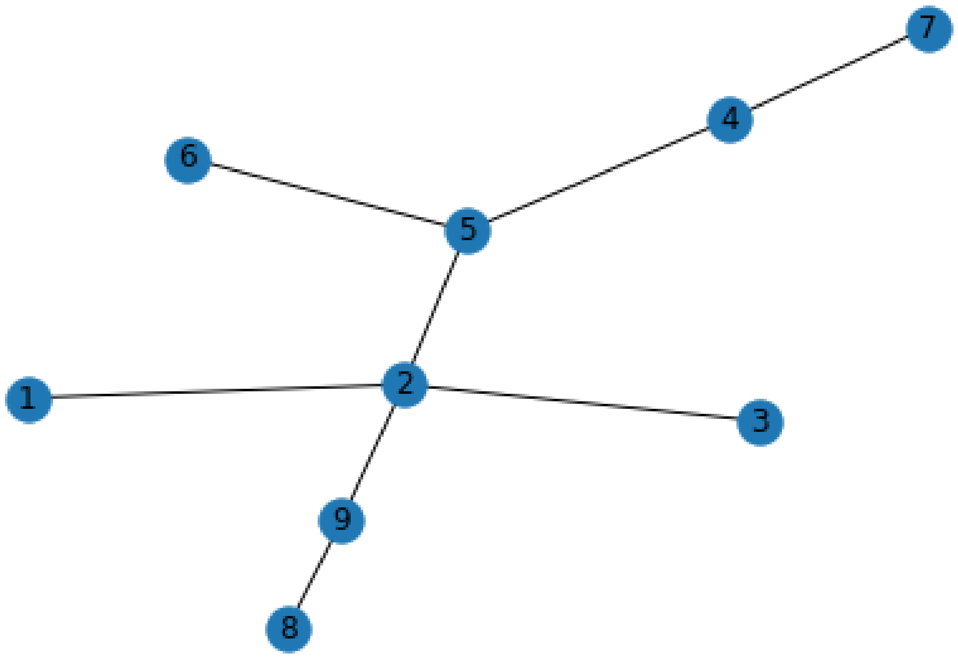
Not all graph poses Hamiltonian paths or cycles.

A graph that possesses a Hamiltonian cycle is known as a **Hamiltonian Graph**.

Observe that a necessary requirement is that a graph must have a cycle to have a hamiltonian cycle. A graph with no cycles is called a tree. This means that a tree is never hamiltonian.

An example of a tree:

[11]:
```
import networkx as nx
G = nx.Graph()

G.add_nodes_from([1,2,3,4,5,6,7,8,9])
G.add_edges_from([(1,2), (2,3), (2,5), (4,5), (4,7), (8,9), (2,9), (5,6)])
nx.draw(G, with_labels=True)
```

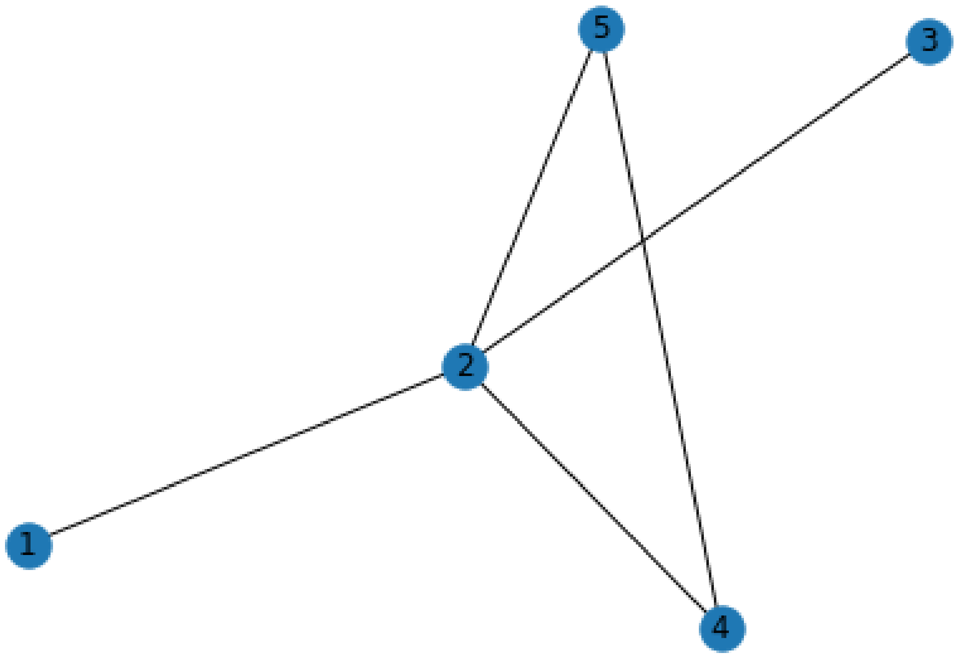


Given n vertices, how many edges does a tree have?

A graph that has a cycle (therefore not a tree) but is not Hamiltonian (no hamiltonian cycle).

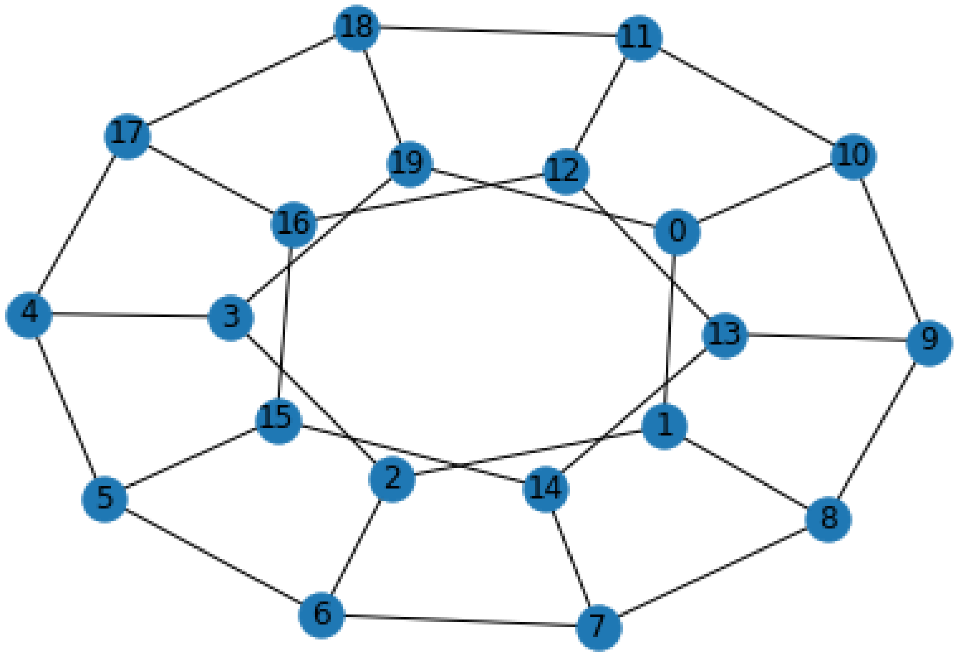
[12]:
```
import networkx as nx
G = nx.Graph()

G.add_nodes_from([1,2,3,4,5])
```

```
G.add_edges_from([(1,2), (2,3), (2,4), (2,5), (4,5)])
nx.draw(G, with_labels=True)
```

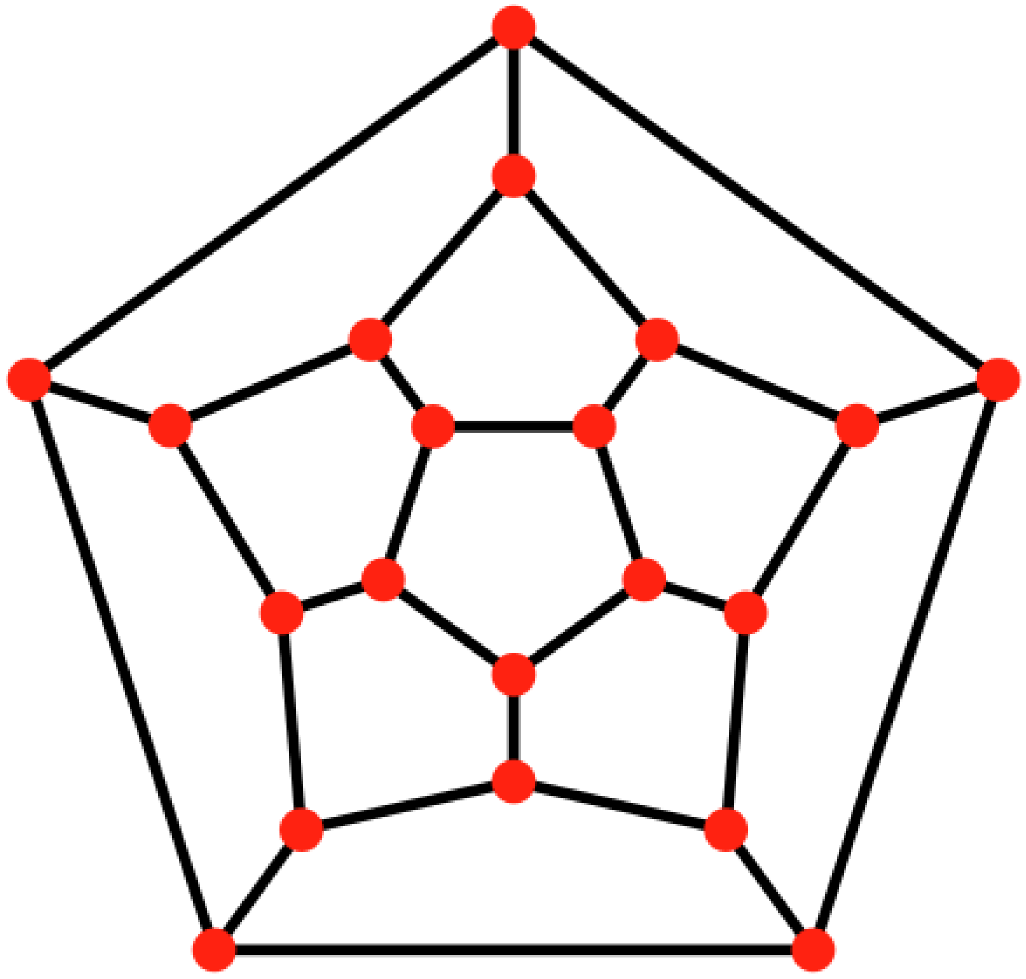


An example of a hamiltonian graph is a dodecahedron graph

[13]:
```
import networkx as nx
G = nx.dodecahedral_graph()
nx.draw(G, with_labels=True)
```

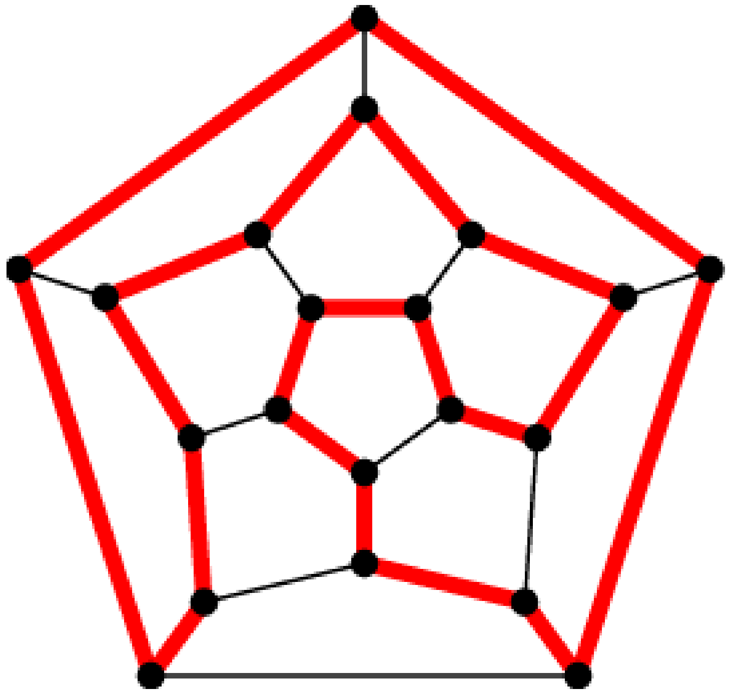


This graph can be drawn on a plane and looks like this (from Wikipedia):



Question: Can you find a Hamiltonian cycle? This will show that the graph is Hamiltonian. Notice that there can be many hamiltonian cycles, you need to find just one.

Answer:



Given an arbitrary graph, to determine if it is a hamiltonian (possessess a hamiltonian cycle) is an NP-complete problem. (Given a solution (a hamiltonian cycle), it is easy to check a hamiltonian graph, but given a graph to check that there are no hamiltonians takes exponential time)

The best yet is by Andreas Björklund $O(1.657^n)$ here

Next class we will discuss the travelling salesman problem. Which is somehow getting the "best" hamiltonian cycle.

Question: Given a weighted graph, to find a hamiltonian cycle (visit every vertex exactly once and reach starting vertex) that has minimum total weight.

Notice that this has numerous applications like what sequence of delivery locations to choose to make all the deliveries and as quickly/cheaply as possible.

```
[ ]:
```