NetworkX:
Network Analysis in Python

Stepan Kuznetsov

Computer Science Department, Higher School of Economics

Outline

Social Network Graphs

NetworkX

Visualization

Computing Graph Parameters

Social Network Analysis

- The study of social structures using graph theory is called *social network analysis* (SNA).

Social Network Analysis

- The study of social structures using graph theory is called *social network analysis* (SNA).
- Thus, SNA is an area on the border of discrete maths and sociology.

Social Network Analysis

- The study of social structures using graph theory is called *social network analysis* (SNA).
- Thus, SNA is an area on the border of discrete maths and sociology.
- Vertices in social network graphs represent *actors:* people, social entities etc.

Social Network Analysis

- The study of social structures using graph theory is called *social network analysis* (SNA).
- Thus, SNA is an area on the border of discrete maths and sociology.
- Vertices in social network graphs represent *actors:* people, social entities etc.
- Edges (also called *ties* or *links*) represent various *relations* between actors.

Social Network Analysis

- The study of social structures using graph theory is called *social network analysis* (SNA).
- Thus, SNA is an area on the border of discrete maths and sociology.
- Vertices in social network graphs represent *actors:* people, social entities etc.
- Edges (also called *ties* or *links*) represent various *relations* between actors.
- The standard example is the friendship relation in social networks.

Parameters of Social Network Graphs

- Graph parameters of social network graphs are important for sociologists studying these networks.

Parameters of Social Network Graphs

- Graph parameters of social network graphs are important for sociologists studying these networks.
- We are going to get acquainted with specialized software for calculating them.

Parameters of Social Network Graphs

- Notice how some parameters of the graph behave specifically in the social network case (if compared to a random graph, for example).

Parameters of Social Network Graphs

- Notice how some parameters of the graph behave specifically in the social network case (if compared to a random graph, for example).
- As we discussed earlier, the clustering coefficients tend to be quite high.

Parameters of Social Network Graphs

- Notice how some parameters of the graph behave specifically in the social network case (if compared to a random graph, for example).
- As we discussed earlier, the clustering coefficients tend to be quite high.
- This reflects the fact that friends of one person are much more likely to be friends also.

Parameters of Social Network Graphs

- On the other hand, being highly clusterized, the social network happens to be tightly connected.

Parameters of Social Network Graphs

- On the other hand, being highly clusterized, the social network happens to be tightly connected.
- The well-known theory of *six degrees of separation* (“six handshakes”) claims that any two people in the world are no more than six social connections from each other.

Parameters of Social Network Graphs

- On the other hand, being highly clusterized, the social network happens to be tightly connected.
- The well-known theory of *six degrees of separation* (“six handshakes”) claims that any two people in the world are no more than six social connections from each other.
- In graph-theoretic terms, this means that the **diameter** of the social connections graph should be ≤ 6.

Dataset

- In our examples, we are going to use the SNAP dataset.

Dataset

- In our examples, we are going to use the SNAP dataset.
- SNAP = Stanford Network Analysis Project.

Dataset

- In our examples, we are going to use the SNAP dataset.
- SNAP = Stanford Network Analysis Project.
- The dataset we use includes friendship relations between friends of given 10 Facebook users (so-called *ego networks*).

Dataset

- In our examples, we are going to use the SNAP dataset.
- SNAP = Stanford Network Analysis Project.
- The dataset we use includes friendship relations between friends of given 10 Facebook users (so-called *ego networks*).
- This makes the dataset relatively small.

Dataset

- In our examples, we are going to use the SNAP dataset.
- SNAP = Stanford Network Analysis Project.
- The dataset we use includes friendship relations between friends of given 10 Facebook users (so-called *ego networks*).
- This makes the dataset relatively small.
- All data is of course anonymized.

Outline

NetworkX

- NetworkX is a Python library for graph analysis and visualization.

NetworkX

- NetworkX is a Python library for graph analysis and visualization.
- Free software, released under BSD-new license.

NetworkX

- NetworkX is a Python library for graph analysis and visualization.
- Free software, released under BSD-new license.
- Capable of handling big graphs (real-world datasets): 10M nodes / 100M edges and more.

NetworkX

- NetworkX is a Python library for graph analysis and visualization.
- Free software, released under BSD-new license.
- Capable of handling big graphs (real-world datasets): 10M nodes / 100M edges and more.
- Highly portable and scalable.

Getting NetworkX

- NetworkX, along with libraries necessary for visualization, can be installed with `pip`:

```
pip install networkx
pip install matplotlib
pip install scipy
```

Getting NetworkX

- NetworkX, along with libraries necessary for visualization, can be installed with `pip`:

```
pip install networkx
pip install matplotlib
pip install scipy
```

- NetworkX is then imported:

```
import networkx as nx
```

Getting NetworkX

- NetworkX, along with libraries necessary for visualization, can be installed with `pip`:

```
pip install networkx
pip install matplotlib
pip install scipy
```

- NetworkX is then imported:

```
import networkx as nx
```

- We've renamed `networkx` to `nx` for convenience.

Defining a Graph: Manual

- In NetworkX, one can define a graph manually, by adding edges one by one.

```
mygraph = nx.Graph()

mygraph.add_edge('A','B')
mygraph.add_edge('B','C')
mygraph.add_edge('C','A')
mygraph.add_edge('B','D')
```

Defining a Graph: Manual

- In NetworkX, one can define a graph manually, by adding edges one by one.

```
mygraph = nx.Graph()

mygraph.add_edge('A','B')
mygraph.add_edge('B','C')
mygraph.add_edge('C','A')
mygraph.add_edge('B','D')
```

- Vertices can be of arbitrary type (strings, numbers, ...).

Other Types of Graphs

- NetworkX can also handle directed graphs, multigraphs etc.

Other Types of Graphs

- NetworkX can also handle directed graphs, multigraphs etc.
- For a directed graph, use `nx.DiGraph` instead of `nx.Graph`.

Other Types of Graphs

- NetworkX can also handle directed graphs, multigraphs etc.
- For a directed graph, use `nx.DiGraph` instead of `nx.Graph`.
- Graphs in NetworkX can also be *weighted*.

Other Types of Graphs

- NetworkX can also handle directed graphs, multigraphs etc.
- For a directed graph, use `nx.DiGraph` instead of `nx.Graph`.
- Graphs in NetworkX can also be *weighted*.
- In a weighted graph, each edge receives a number called its weight.

Other Types of Graphs

- NetworkX can also handle directed graphs, multigraphs etc.
- For a directed graph, use `nx.DiGraph` instead of `nx.Graph`.
- Graphs in NetworkX can also be *weighted*.
- In a weighted graph, each edge receives a number called its weight.
- Example: time (or cost) of driving along a road.

Other Types of Graphs

- NetworkX can also handle directed graphs, multigraphs etc.
- For a directed graph, use `nx.DiGraph` instead of `nx.Graph`.
- Graphs in NetworkX can also be *weighted*.
- In a weighted graph, each edge receives a number called its weight.
- Example: time (or cost) of driving along a road.
- Weight is added just as an optional parameter to `add_edge`:

Reading a Graph from File

- NetworkX is also capable of reading graphs from files (datasets).

Reading a Graph from File

- NetworkX is also capable of reading graphs from files (datasets).
- In our example, we use SNAP's Facebook dataset (10 ego networks combined).

Reading a Graph from File

- NetworkX is also capable of reading graphs from files (datasets).
- In our example, we use SNAP's Facebook dataset (10 ego networks combined).
- In the file `facebook_combined.txt` one finds the list of edges as pairs of numbers (vertices are numbered).

Reading a Graph from File

- NetworkX is also capable of reading graphs from files (datasets).
- In our example, we use SNAP's Facebook dataset (10 ego networks combined).
- In the file `facebook_combined.txt` one finds the list of edges as pairs of numbers (vertices are numbered).
- The data gets imported by the `nx.read_edgelist` method.

Outline

- Social Network Graphs
- NetworkX
- Visualization
- Computing Graph Parameters

Visualizing Graphs

- Graphs are abstract objects, but they have nice geometric representations.

Visualizing Graphs

- Graphs are abstract objects, but they have nice geometric representations.
- In many cases, it is very helpful to **see** how the graph looks like.

Visualizing Graphs

- Graphs are abstract objects, but they have nice geometric representations.
- In many cases, it is very helpful to **see** how the graph looks like.
- Rendering an abstract graph to a picture is called *visualization*.

Visualizing Graphs

- Graphs are abstract objects, but they have nice geometric representations.
- In many cases, it is very helpful to **see** how the graph looks like.
- Rendering an abstract graph to a picture is called *visualization*.
- NetworkX is capable of visualizing graphs, both in 2D and 3D.

Visualization: Small Example

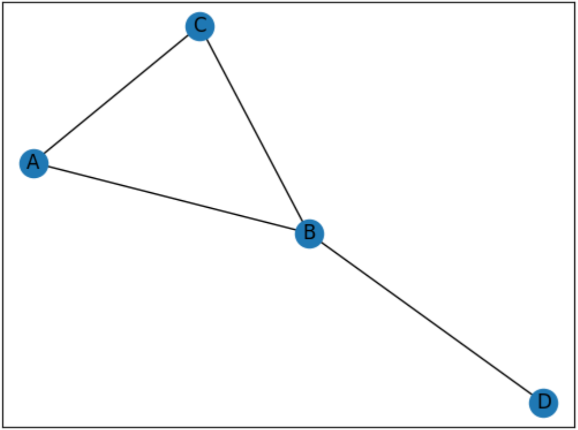
- NetworkX visualizes graphs via Matplotlib (a Python library for plotting).

Visualization: Small Example

- NetworkX visualizes graphs via Matplotlib (a Python library for plotting).
- The method is called `nx.draw_networkx`:

```
nx.draw_networkx(mygraph)
matplotlib.pyplot.savefig("mygraph.png")
```

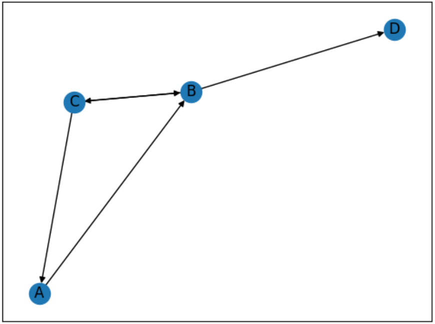
Visualization: Small Example



NetworkX output

Visualization: Small Example

This is how a directed graph is visualized. Two opposite edges between B and C are drawn as one edge with two arrows.



NetworkX output

Visualization of Real Data

- We remove labels, because there are too many vertices:

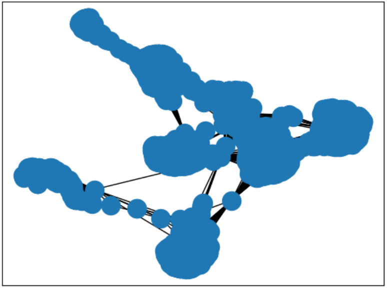
```
nx.draw_networkx(fb_gr, with_labels=False)
```

Visualization of Real Data

- We remove labels, because there are too many vertices:

```
nx.draw_networkx(fb_gr, with_labels=False)
```

- Visualization makes clustering visible:



NetworkX output

Outline

Social Network Graphs

NetworkX

Visualization

Computing Graph Parameters

Graph Parameters in NetworkX

- NetworkX provides a convenient interface to algorithms computing graph parameters.

Graph Parameters in NetworkX

- NetworkX provides a convenient interface to algorithms computing graph parameters.
- Global parameters of the graph are just functions of it.

Graph Parameters in NetworkX

- NetworkX provides a convenient interface to algorithms computing graph parameters.
- Global parameters of the graph are just functions of it.
- For example, if we wish to calculate the *average clustering coefficient* (the average value of local clustering coefficients), we just run

```
av_clust = nx.average_clustering(fb_gr)
```

Graph Parameters in NetworkX

- Suppose we want to check "six handshakes."

Graph Parameters in NetworkX

- Suppose we want to check “six handshakes.”
- That is, we have to calculate the diameter of our graph:

```
diam = nx.diameter(fb_gr)
```

Graph Parameters in NetworkX

- Suppose we want to check “six handshakes.”
- That is, we have to calculate the diameter of our graph:

```
diam = nx.diameter(fb_gr)
```

- The calculation takes quite long… and on our data it yields 8.

Graph Parameters in NetworkX

- Suppose we want to check "six handshakes."
- That is, we have to calculate the diameter of our graph:

```
diam = nx.diameter(fb_gr)
```

- The calculation takes quite long... and on our data it yields 8.
- This is quite a good result, recalling that we have just a fusion of 10 ego nets, not the full Facebook graph.

Graph Parameters in NetworkX

- Computing the diameter (and also some other parameters, such as radius) is based on computing eccentricities of vertices.

Graph Parameters in NetworkX

- Computing the diameter (and also some other parameters, such as radius) is based on computing eccentricities of vertices.
- If we need to compute several parameters of this sort, we can precompute the dictionary of eccentricities by the `nx.eccentricity` function.

Graph Parameters in NetworkX

- Computing the diameter (and also some other parameters, such as radius) is based on computing eccentricities of vertices.
- If we need to compute several parameters of this sort, we can precompute the dictionary of eccentricities by the `nx.eccentricity` function.
- This function returns the dictionary of eccentricities, keyed by vertices.

Graph Parameters in NetworkX

- Computing the diameter (and also some other parameters, such as radius) is based on computing eccentricities of vertices.
- If we need to compute several parameters of this sort, we can precompute the dictionary of eccentricities by the `nx.eccentricity` function.
- This function returns the dictionary of eccentricities, keyed by vertices.
- If we pass this dictionary to the diameter computing function, it will run much faster.

Distances

- By definition, the *distance* between two vertices is the length of the shortest path connecting them.

Distances

- By definition, the *distance* between two vertices is the length of the shortest path connecting them.
- This can be computed by `nx.shortest_path_length`

Distances

- By definition, the *distance* between two vertices is the length of the shortest path connecting them.
- This can be computed by `nx.shortest_path_length`
- In directed graphs, the path should also be directed—thus, sometimes $d(a, b) \neq d(b, a)$.

Distances

- By definition, the *distance* between two vertices is the length of the shortest path connecting them.
- This can be computed by `nx.shortest_path_length`
- In directed graphs, the path should also be directed—thus, sometimes $d(a, b) \neq d(b, a)$.
- **Caveat!** If there is no path, NetworkX throws an exception.

Distances

- By definition, the *distance* between two vertices is the length of the shortest path connecting them.
- This can be computed by `nx.shortest_path_length`
- In directed graphs, the path should also be directed—thus, sometimes $d(a,b) \neq d(b,a)$.
- **Caveat!** If there is no path, NetworkX throws an exception.
- To be on the safe side, use `nx.has_path` before.

Traversing

- One can also get the shortest path itself: `nx.shortest_path`

Traversing

- One can also get the shortest path itself: `nx.shortest_path`
- The path is given as a list of vertices.

Traversing

- One can also get the shortest path itself: `nx.shortest_path`
- The path is given as a list of vertices.
- Traversal algorithms are implemented as functions which return *generators*.

Traversing

- One can also get the shortest path itself: `nx.shortest_path`
- The path is given as a list of vertices.
- Traversal algorithms are implemented as functions which return *generators*.
- For example, `nx.dfs_preorder_nodes` returns a generator which yields the vertices of the graph in the preorder DFS traversing order.

Traversing: Example

```
G = nx.Graph()

G.add_edge('A','B')
G.add_edge('B','C')
G.add_edge('C','A')
G.add_edge('B','D')
G.add_edge('D','E')
G.add_edge('E','A')

print(list(nx.dfs_preorder_nodes(G,
    source='C')))
```

Traversing: Example

This yields the following result:

```
['C', 'B', 'A', 'E', 'D']
```

Conclusion

- More information is available in NetworkX documentation.

Conclusion

- More information is available in NetworkX documentation.
- Please consult it when accomplishing the programming task.

Conclusion

- More information is available in NetworkX documentation.
- Please consult it when accomplishing the programming task.
- Good luck!