NetworkX and Graph Theory

Adam Richards

Galvanize, Inc

May 2017

Objectives

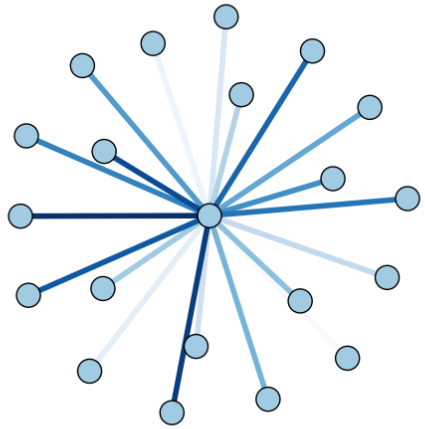
Morning

- Understand different application of graphs
- Components and types of graphs
- Measure of centrality of a node (Degree, Betweenness, Eigenvector)
- Graph search algorithm: Breadth First Search

Afternoon

- Python libraries and other tools / Datasets
- Applications of community detection
- Measuring quality of communities
- Algorithms for dividing graphs into communities (Divisive, Agglomerative

Terms



- network and graph are used interchangeably
- a graph is a set of nodes or vertices joined by lines or edges.
- networks are all around us (roads, friendships, collaborations)
- Provide a data structure that is intuitive

NetworkX

- Python language data structures for graphs, digraphs, and multigraphs.
- Many standard graph algorithms
- Network structure and analysis measures
- Generators for classic graphs, random graphs, and synthetic networks
- Nodes can be änything"(e.g. text, images, XML records)
- Edges can hold arbitrary data (e.g. weights, time-series)

Originally developed by Aric Hagberg, Dan Schult, and Pieter Swart at Los Alamos National Laboratory, but now it is the main package to work with Network in Python and is developed by many others including Google.

(Hagberg et al., 2008)

How to choose?

When to USE NetworkX to perform network analysis?

- Unlike many other tools, it is designed to handle data on a scale relevant to modern problems.
- Most of the core algorithms rely on extremely fast legacy code
- Highly flexible graph implementations (a graph/node can be anything!)
- Extensive set of native readable and writable formats
- Takes advantage of Python's ability to pull data from the Internet or databases

When to AVOID NetworkX to perform network analysis?

- Large-scale problems that require faster approaches (i.e. Facebook/Twitter whole social network...)
- Better use of resources/threads than Python

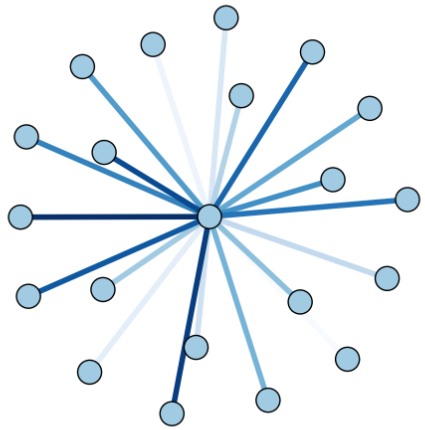
https://www.cl.cam.ac.uk/ cm542/teaching/2010/stna-pdfs/stna-lecture8.pdf

Types of networks

Graph Type	NetworkX class
Undirected Simple	`Graph`
Directed Simple	`DiGraph`
With Self-loops	`Graph`, `DiGraph`
With Parallel edges	`MultiGraph`, `MultiDiGraph`

drawing and plotting

- Graphs are intuitive in part due to ease of visualization
- It is possible to draw in graphs in:
  - NetworkX
  - GraphViz
  - matplotlib



Checkout the NetworkX Gallery for some examples

More terminology

- Neighbors: The neighbors of a node are the nodes that it is connected to.

  `G.neighbors(node)`

- Degree: The degree of a node is the number of neighbors for a given node

  `G.degree(node)`

- Directed graphs can be split into indegree and outdegree.

  `DiGraph.in_degree(node), DiGraph.out_degree(node)`

- Walk: A walk is a sequence of nodes and edges that connect
- Path: A path is a walk where no node is crossed twice

  `nx.shortest_path(G)`

- A closed path is known as a cycle

  see the docs

Even more terminology

- Connected: A graph is connected if every pair of vertices is connected by some path

```
nx.is_connected(G)
```

- Subgraph: A subgraph is a subset of the nodes of a graph and all the edges between them

```
G.subgraph([0,1,2])
```

- Graph Diameter: The diameter of a graph is the largest number of vertices that much be traversed in order to get from one vertex to any other vertex.

```
nx.diameter(G)
```

Resources

- NetworkX algorithms
- graph theory tutorial
- NetworkX tutorial

References I

Hagberg, A. A., Schult, D. A., and Swart, P. J. (2008). Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy 2008)*, pages 11–16.