Complex networks
Phys 7682: Computational Methods for Nonlinear Systems

- networks are everywhere (and always have been)
  - relationships (edges) among entities (nodes)
- explosion of interest in network structure, function, and evolution over the last decade or so
  - technology: Internet, World Wide Web
  - biology: genomics, gene expression, protein-protein interactions, physiology
  - sociology: online communities, gossip & rumors, epidemiology, etc.
- interest in mathematical characterization fueled by many common properties among diverse networks
  - degree distributions
  - clustering
  - small-worlds, cliques, and communities

World-wide internet traffic, by Stephen G. Eick (via Barabasi)
Software class relationships in VTK, by C. Myers
Data structures

- network = graph (a set of nodes connected by edges)
- interested here in **undirected graphs**
  - edge is symmetric between two connecting nodes
- data structures for undirected graph?
  - we use a dictionary of connections
    - dictionary maps *key* to *value*
    - connections[i] = [j_0, j_1, j_2, ...]
    - connections dictionary is directed (asymmetric), so we need to duplicate connections
  - use object-oriented programming to encapsulate internal implementation (dictionary) from external interface (AddNode, AddEdge, etc.)
  - another common implementation is *adjacency matrix*
    - a_{ij} = 1 if nodes i,j connected; 0 otherwise
An aside on object-oriented programming

• primary goal is to support the representation of new types of data, with their own behaviors
• often built up as an aggregation of other data types with associated logic about their relationships
• in Python, we define a new **class** to allow for the creation of objects of that class
  • e.g., UndirectedGraph (nodes, edges; support graph traversal)
  • e.g., Employee (name, ID number, salary; support accounting)
  • e.g., array (data, shape; support slicing, arithmetic operations)
• we define a class, but typically work with **instances** of that class
  • graph1 = UndirectedGraph(); graph1.AddEdge(1,2)
  • graph2 = UndirectedGraph(); graph2.AddEdge(3,4)
• encapsulation separates external behavior of what it means to look like a graph (AddNode, AddEdge, etc.) from how that behavior is implemented internally (e.g., via dictionary, adjacency matrix, etc.)
An aside on object-oriented programming

• in defining a class, we declare internally a parameter (conventionally called `self`) to indicate that we are talking about this instance, but that parameter is hidden when we call a method on the object

```python
class UndirectedGraph:
    #...
    def AddNode(self, node):
        # add node to graph self
    def AddEdge(self, node1, node2):
        # add edge betw. node1 & node2 in graph self
```

```python
>>> g = UndirectedGraph()
>>> g.AddNode(1)  # => UndirectedGraph.AddNode(g,1)
>>> g.AddNode(2)  # => UndirectedGraph.AddNode(g,2)
>>> g.AddEdge(1,2)  # => UndirectedGraph.AddEdge(g,1,2)
```
An aside on object-oriented programming

- why worry about self? among other things, to distinguish self from nonself

```python
class list:
    def __add__(self, other):
        # concatenate other after self,
        # returning longer list

class array:
    def __add__(self, other):
        # add self and other element-wise,
        # returning array of same length
```

```none
>>> [1,2,3]+[4,5,6] => [1,2,3,4,5,6]
>>> array([1,2,3])+array([4,5,6]) => array([5,7,9])
```

- list.__add__ is different function than array.__add__: operator overloading
- in Python, double underscores used to associate functions with operators
Graph traversal algorithms

- graph traversal
  - iterating through a graph (i.e., over its nodes and edges) and calculating some quantity of interest
    - average shortest path: shortest paths between all pairs of nodes in a graph
    - node and edge betweenness: what fraction of shortest paths each node or edge participates in
    - connected clusters (percolation)
  - traversing nodes and edges, marking nodes as visited so they get visited only once
    - most common: breadth-first and depth-first

- breadth-first search
  - involves iterating through the neighbors of all the nodes in the current shell, and adding to the next shell all subsequent neighbors which have not already been visited
Small-world networks

- motivated by phenomenon of “six degrees of separation”
- studied at Cornell by Duncan Watts and Steve Strogatz
  - Nature 393, 440-442 (1998)
  - simple model of networks with regular short-range bonds and random long-range bonds
  - examination of path lengths and clustering in model and in real-world networks

<table>
<thead>
<tr>
<th>Table 1 Empirical examples of small-world networks</th>
</tr>
</thead>
<tbody>
<tr>
<td>L_{actual}</td>
</tr>
<tr>
<td>Film actors</td>
</tr>
<tr>
<td>Power grid</td>
</tr>
<tr>
<td>C. elegans</td>
</tr>
</tbody>
</table>

Characteristic path length $L$ and clustering coefficient $C$ for three real networks, compared to random graphs with the same number of vertices ($n$) and average number of edges per vertex ($k$). (Actors: $n = 225,226$, $k = 61$. Power grid: $n = 4,941$, $k = 2.67$. C. elegans: $n = 282$, $k = 14$.) The graphs are defined as follows. Two actors are joined by an edge if they have acted in a film together. We restrict attention to the giant connected component $G$ of this graph, which includes $\sim 90\%$ of all actors listed in the Internet Movie Database (available at http://us.imdb.com), as of April 1997. For the power grid, vertices represent generators, transformers and substations, and edges represent high-voltage transmission lines between them. For C. elegans, an edge joins two neurons if they are connected by either a synapse or a gap junction. We treat all edges as undirected and unweighted, and all vertices as identical, recognizing that these are crude approximations. All three networks show the small-world phenomenon: $L \gg L_{\text{random}}$ but $C \ll C_{\text{random}}$.

decrease in average path length with increasing # of long-range bonds, from Watts & Strogatz
Percolation

• Some applications
  - flow in porous media (e.g., pressure-driven flow in rock)
  - conductivity of disordered systems (e.g., random resistor networks)
  - forest fires
  - disease transmission in social networks
Power laws, correlation lengths, finite-size scaling & universality

Critical points imply scale invariance and power laws

Phase transitions often involve a diverging correlation length $\xi \sim |p-p_c|^{-\nu}$

Diverging correlation length constrained by finite system size $\rightarrow$ finite-size scaling

Microscopically different systems can exhibit the same critical properties $\rightarrow$ universality

$P(p) \sim (p-p_c)^\beta$  \hspace{1cm} $D(s) \sim s^{-\tau}$

probability of being in infinite cluster  \hspace{1cm} cluster size distribution
Neutral evolutionary networks in biology: what are the mutationally-connected networks of genotypes that produce the same phenotype?

The bow-tie structure of the World Wide Web (SCC = strongly connected component in a directed graph)


Broder et al., Computer Networks (2000)
Random network ensembles

Configuration model graphs: random networks consistent with specified degree distribution $P(k)$

Grown graphs, e.g., Barabasi-Albert preferential attachment graph

mathinsight.org

Poisson

Heavy-tailed

graphstream-project.org
Types of network structures

• Undirected graphs
  - edges symmetric between nodes
    ‣ e.g., protein-protein interaction network: what interacts with what?
    ‣ e.g., scientific collaboration network: who has written a paper together?

• Directed graphs
  - edges directed from a source node to destination node
    - e.g., web pages: what links to what?
    - e.g., food webs: who eats whom?
    - e.g., river networks: what flows where?
    - e.g., software systems: what classes contain other classes?
Types of network structures

- Weighted graphs
  - edge-weighted
    - e.g., traffic flow: traffic capacity and/or travel time of each road/edge (directed)
    - e.g., protein-protein interaction network: probability of two proteins interacting (undirected)
  - node-weighted
    - e.g., resistance to migration of land parcels

- Multipartite graphs
  - e.g., bipartite graph of places and transitions in a Petri net
  - e.g., actors and movies in an actor collaboration network
  - e.g., bees and flowers in a plant-pollinator network
NetworkX: a Python package for creating, manipulating, and analyzing networks (networkx.github.io)
Network growth, structure, etc.

• Other papers/projects for further consideration (or maybe you have your own in mind)
  - Barabasi and Albert, “Emergence of scaling in random networks”
    ‣ power-law degree distributions (actor network with bipartite graph?)
  - Callaway et al., “Are randomly grown graphs really random?”
    ‣ essential singularity for onset of connected cluster
  - Girvan and Newman, “Community structure in social and biological networks”
    ‣ quantifying tightly-knit groups in large networks
  - Yu et al., “The importance of bottlenecks in protein networks: Correlation with gene essentiality and expression dynamics”
    ‣ role of betweenness in organizing biological networks
  - Kaiser and Hilgetag, “Nonoptimal Component Placement, but Short Processing Paths, due to Long-Distance Projections in Neural Systems”
    ‣ investigation of wiring lengths and processing paths from neural network data
  - graph layout is also an interesting problem
    ‣ how to optimally place graph nodes and edges (e.g., on a 2D display) when there is no intrinsic geometric information attached to graph