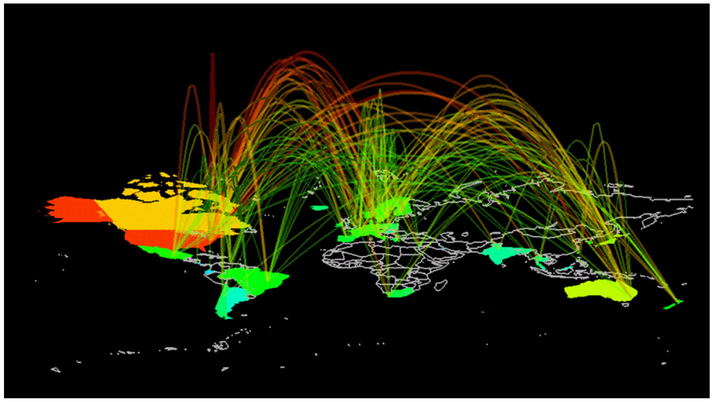
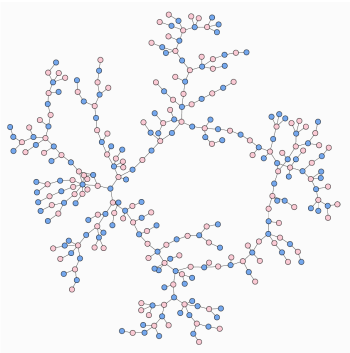
Complex networks

Phys 682 / CIS 629: Computational Methods for Nonlinear Systems

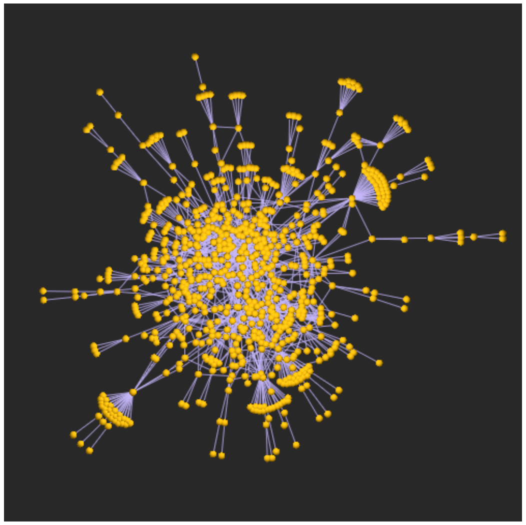
- networks are everywhere (and always have been)
  - relationships (edges) among entities (nodes)
- explosion of interest in network structure, function, and evolution over the last decade
  - technology: Internet, World Wide Web
  - biology: genomics, gene expression, protein-protein interactions, physiology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



World-wide internet traffic, by Stephen G. Eick (via Barabasi)



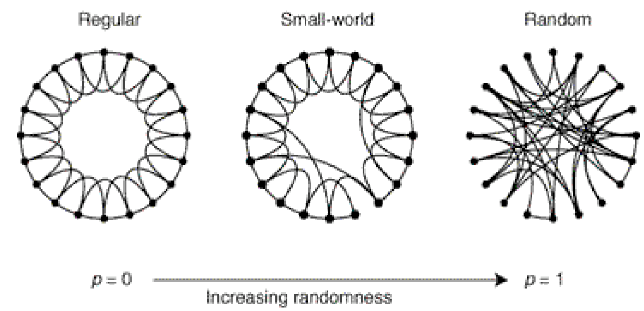
High school dating, from Bearman 2004
Image by Mark Newman

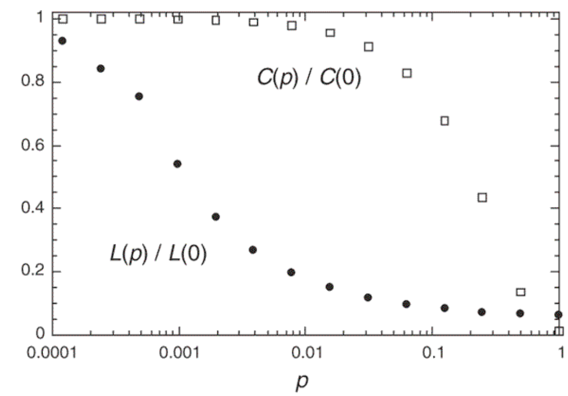


Software class relationships in VTK, by C. Myers

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
- Course exercise
  - calculation of shortest path lengths in randomly wired graphs
  - scaling collapse for various p,Z,L
  - application to real network data
  - calculation of node and edge betweenness
  - provided with simple visualization tool





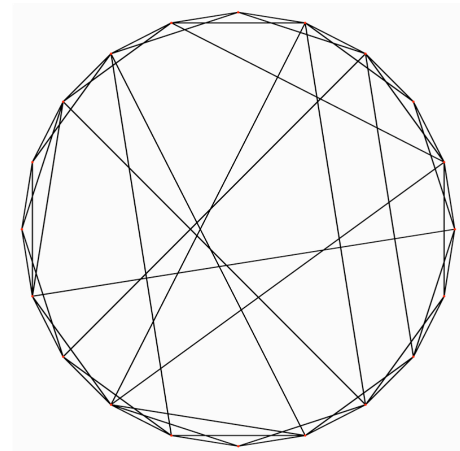
decrease in average path length with
increasing # of long-range bonds, from Watts & Strogatz

Table 1 Empirical examples of small-world networks

	L_{actual}	L_{random}	C_{actual}	C_{random}
Film actors	3.65	2.99	0.79	0.00027
Power grid	18.7	12.4	0.080	0.005
C. elegans	2.65	2.25	0.28	0.05

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[16] of this graph, which includes ~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: $L \gtrsim L_{random}$ but $C \gg C_{random}$.

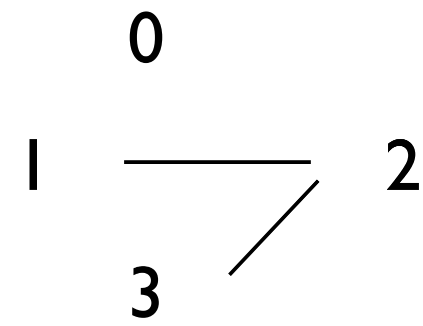
Computing for small-world networks: data structures



- network = graph (a set of nodes connected by edges)
- interested here in *undirected graphs* (edge is symmetric in two connecting nodes
- data structures for undirected graph?
  - some use adjacency matrix
    - a_{ij} = 1 if nodes i,j connected; 0 otherwise
  - we will use a neighbor dictionary
    - dictionary maps *key* to *value*
    - neighbor_dict[i] = [j_0, j_1, j_2, ...]
    - i.e., for a node i, we store a list [j_0, j_1, j_2, ...] of nodes that i is connected to
    - neighbor dictionary is directed (asymmetric), so we need to duplicate connections
      - if i points to j, then j must point to i
    - add a new entry to the dictionary when a new node is added, append to an existing entry when an existing node is connected to

Computing for small-world networks: object-oriented programming

- object-oriented programming
  - definition of new datatypes, along with associated behavior
  - encapsulate details of internal implementation (e.g., neighbor dictionary vs. adjacency matrix) without modifying external interface
- python `class` keyword allows definition of new class of objects




```
class UndirectedGraph:
   def __init__(self):
      self.neighbor_dict = {}

   def AddNode(self, node):
      # code to add a node

   def AddEdge(self, node1, node2):
      # code to add an edge connecting two nodes

   def HasNode(self, node):
      # return True if graph has specified node

   # etc.
```

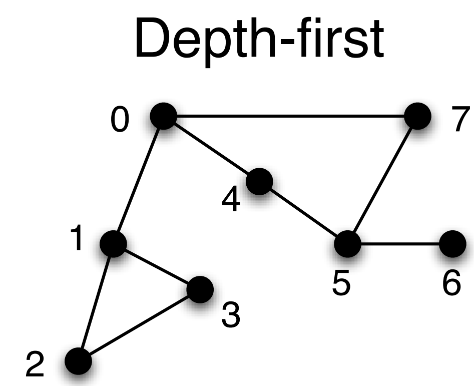
```
>>> g = UndirectedGraph()
>>> g.AddNode(0)
>>> g.AddEdge(1,2)
>>> g.AddEdge(2,3)
>>> g.HasNode(4)
False
```

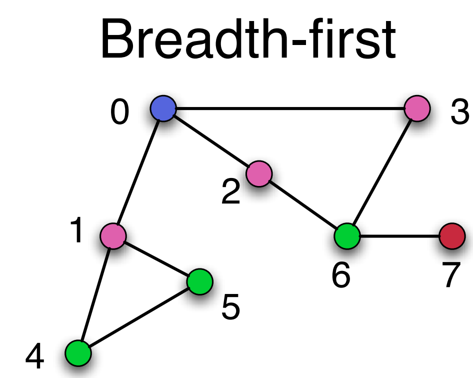
“self” refers to the particular object instance we are working with, in this case the graph “g”

`g.AddNode(0)` is shorthand for `UndirectedGraph.AddNode(g,0)`

Computing for small-world networks: 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 path 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





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

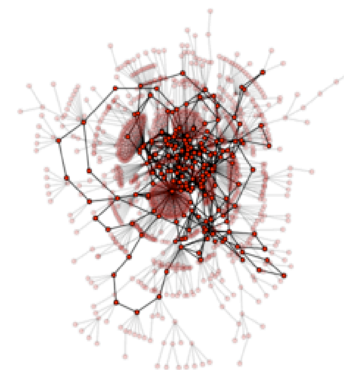
NetworkX: a Python package for creating, manipulating, and analyzing networks (networkx.lanl.gov)

NetworkX

NetworkX

High productivity software for complex networks

About



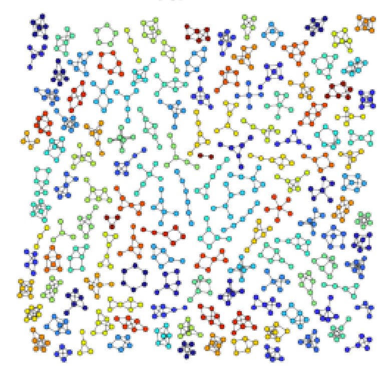
NetworkX (NX) is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.

Features:

- Includes standard graph-theoretic and statistical physics functions
- Easy exchange of network algorithms between applications, disciplines, and platforms
- Includes many classic graphs and synthetic networks
- Nodes and edges can be "anything" (e.g. time-series, text, images, XML records)
- Exploits existing code from high-quality legacy software in C, C++, Fortran, etc.
- Open source (encourages community input)
- Unit-tested

Additional benefits due to Python:

- Allows fast prototyping of new algorithms
- Easy to teach
- Multi-platform
- Allows easy access to almost any database



Quick Example

Just write in Python

```
>>> import networkx as NX
>>> G=NX.Graph()
>>> G.add_edge(1,2)
>>> G.add_node("spam")
>>> print G.nodes()
[1, 2, 'spam']
>>> print G.edges()
[(1, 2)]
```