Module II - Why do SNA in NetworkX

Drew Conway — Department of Politics

NEW YORK UNIVERSITY

June 29, 2010

Agenda for Module II

Speed and scalability

- Why speed and scalability matter
- NetworkX is designed for large data sets
- Comparing NetworkX to other SNA tools

Agenda for Module II

Speed and scalability

- Why speed and scalability matter
- NetworkX is designed for large data sets
- Comparing NetworkX to other SNA tools

How NetworkX complements Python's scientific computing suite

- SciPy/NumPy
- Matplotlib
- and others...

Agenda for Module II

Speed and scalability

- Why speed and scalability matter
- NetworkX is designed for large data sets
- Comparing NetworkX to other SNA tools

How NetworkX complements Python's scientific computing suite

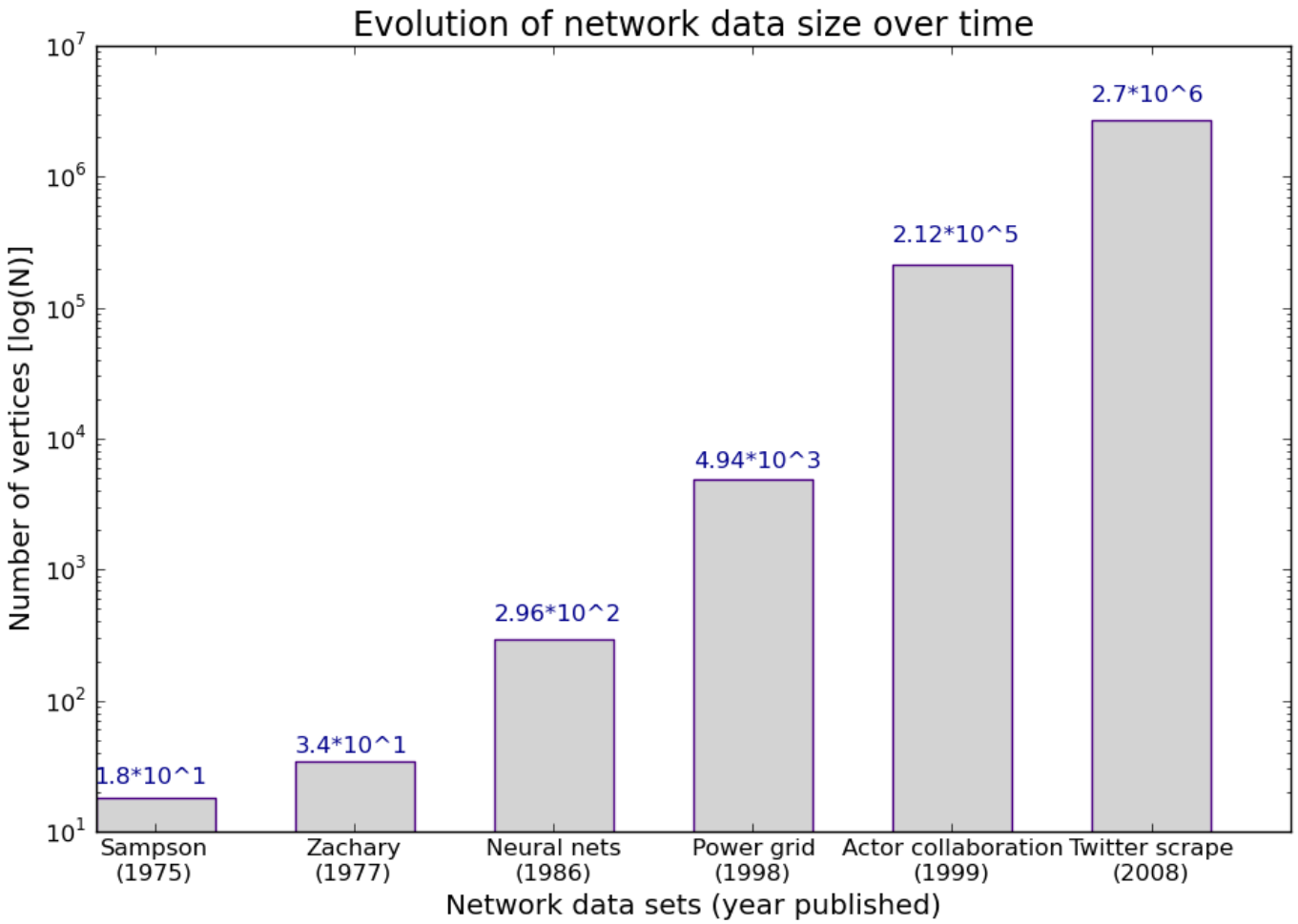
- SciPy/NumPy
- Matplotlib
- and others...

Getting data in and out of NetworkX

- I/O basics
- Pulling non-local data
  - Directly from the web
  - External databases

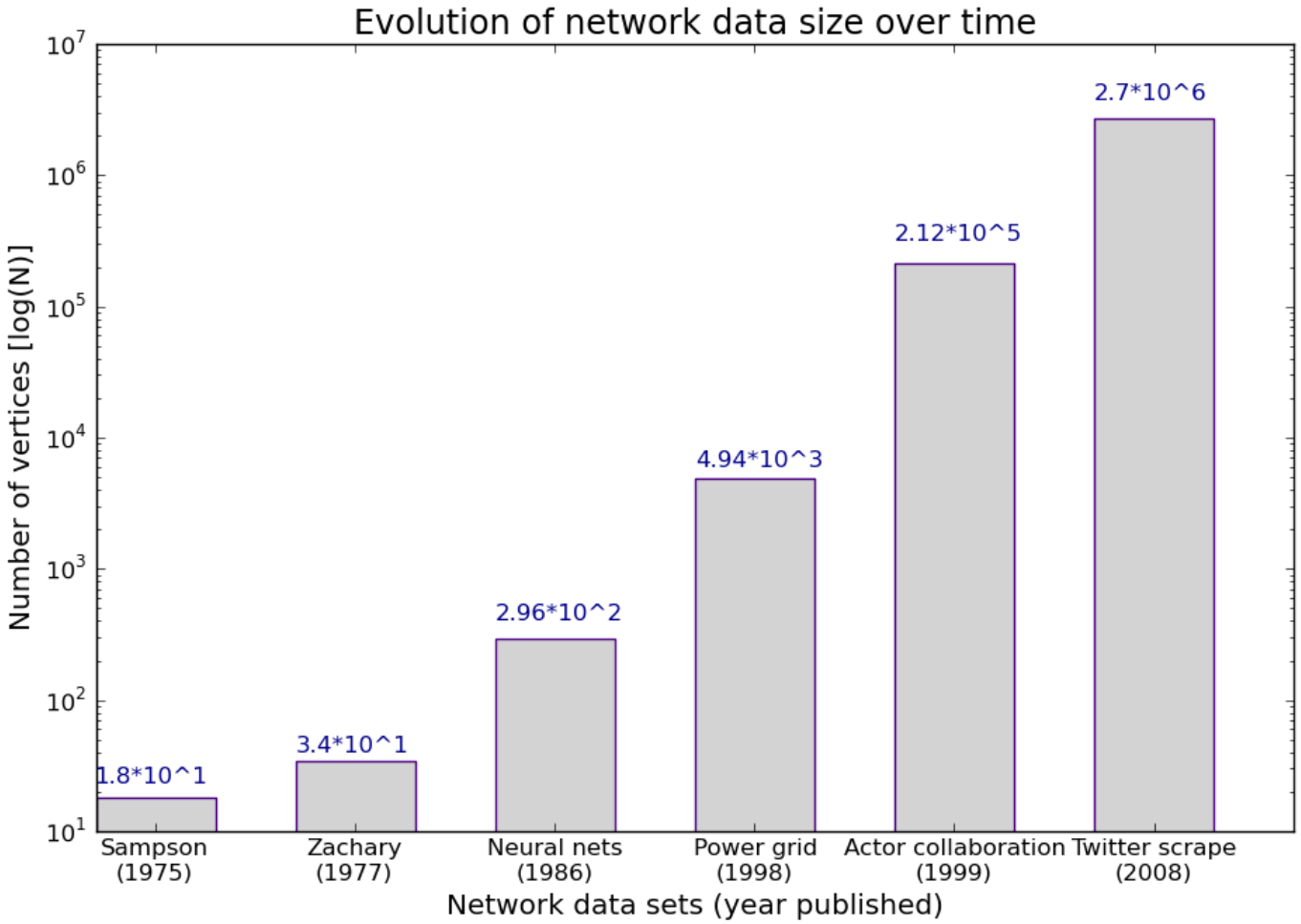
Why should we worry about scalability?

The size of networks being studying has increased rapidly over the years...



Why should we worry about scalability?

The size of networks being studying has increased rapidly over the years...



As network data becomes more readily available this trend will continue!

How network size affects tools

While the data continues to scale up, many tools have not kept pace

Standard Tool Limitations

Tool	Node Limit	Platforms
UCINet	$\sim V = 5K$	Windows only
Pajek	$\sim V = 100K$	Windows only
Statnet	$\sim V =$	Multi-platform
ORA	$\sim V =$	Window & Linux

How network size affects tools

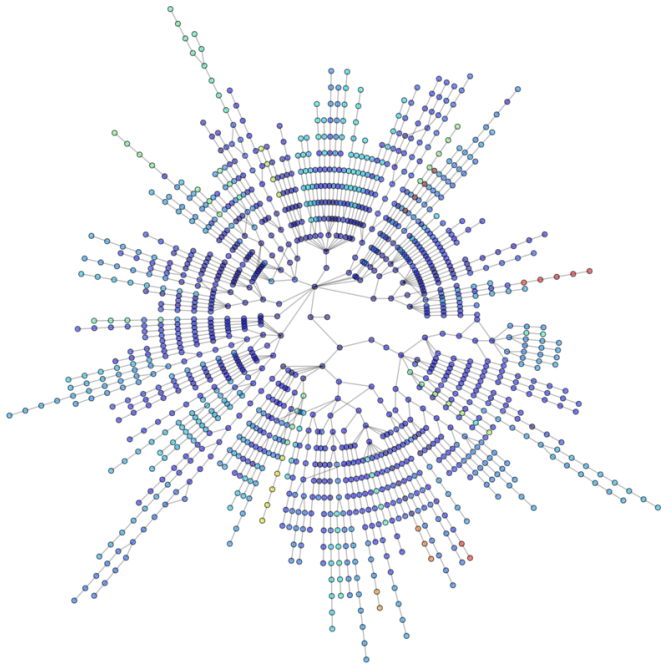
While the data continues to scale up, many tools have not kept pace

Standard Tool Limitations

Tool	Node Limit	Platforms
UCINet	$\sim V = 5K$	Windows only
Pajek	$\sim V = 100K$	Windows only
Statnet	$\sim V =$	Multi-platform
ORA	$\sim V =$	Window & Linux

NetworkX is designed to handle data sets of the scale being generated today

- 10's of millions or nodes and 100's of millions of edges
- Can read network data from local files, or from external sources
  - Internet
  - Relational databases



Moving beyond basic concepts of the "graph"

In a more fundamental way, however, most network tools are limited in their concept of what can be a network

- Networks are collections of nodes and edges
- Nodes are static integers or strings, and edges are binary or continuous values

NetworkX can represent **ANY** relationship supported by Python data types

Suppose we had data, or a data generating process, that was a time-series

- Current tools need kludges or hacks to add this data
- In NetworkX, we simply use the built-in Python `datetime` package to create a network of time-stamps

Simple time-series network

```
G=DiGraph()
# Create datetime object nodes
for v in xrange(num_nodes):
    G.add_node(datetime.now())
time_nodes=G.nodes()
# Add edges with 'time' attribute
for i in xrange(num_nodes):
    draws=random.uniform(0,1,num_nodes)
    for j in xrange(num_nodes):
        if i!=j and draws[j]<=p:
            G.add_edge(time_nodes[i],time_nodes[j],time=datetime.now())
...
# target source datetime_created
2010-05-25 13:38:42.515323 2010-05-25 13:38:42.515492
    {'time': datetime.datetime(2010, 5, 25, 13, 38, 42, 515752)}
...
```

Moving beyond basic concepts of the "graph"

In a more fundamental way, however, most network tools are limited in their concept of what can be a network

- Networks are collections of nodes and edges
- Nodes are static integers or strings, and edges are binary or continuous values

NetworkX can represent **ANY** relationship supported by Python data types

Suppose we had data, or a data generating process, that was a time-series

- Current tools need kludges or hacks to add this data
- In NetworkX, we simply use the built-in Python `datetime` package to create a network of time-stamps

Simple time-series network

```
G=DiGraph()
# Create datetime object nodes
for v in xrange(num_nodes):
    G.add_node(datetime.now())
time_nodes=G.nodes()
# Add edges with 'time' attribute
for i in xrange(num_nodes):
    draws=random.uniform(0,1,num_nodes)
    for j in xrange(num_nodes):
        if i!=j and draws[j]<=p:
            G.add_edge(time_nodes[i],time_nodes[j],time=datetime.now())
...
# target source datetime_created
2010-05-25 13:38:42.515323 2010-05-25 13:38:42.515492
    {'time': datetime.datetime(2010, 5, 25, 13, 38, 42, 515752)}
...
```

Moving beyond basic concepts of the "graph"

In a more fundamental way, however, most network tools are limited in their concept of what can be a network

- Networks are collections of nodes and edges
- Nodes are static integers or strings, and edges are binary or continuous values

NetworkX can represent **ANY** relationship supported by Python data types

Suppose we had data, or a data generating process, that was a time-series

- Current tools need kludges or hacks to add this data
- In NetworkX, we simply use the built-in Python `datetime` package to create a network of time-stamps

Simple time-series network

```
G=DiGraph()
# Create datetime object nodes
for v in xrange(num_nodes):
    G.add_node(datetime.now())
time_nodes=G.nodes()
# Add edges with 'time' attribute
for i in xrange(num_nodes):
    draws=random.uniform(0,1,num_nodes)
    for j in xrange(num_nodes):
        if i!=j and draws[j]<=p:
            G.add_edge(time_nodes[i],time_nodes[j],time=datetime.now())
...
# target source datetime_created
2010-05-25 13:38:42.515323 2010-05-25 13:38:42.515492
    {'time': datetime.datetime(2010, 5, 25, 13, 38, 42, 515752)}
...
```

Moving beyond basic concepts of the "graph"

In a more fundamental way, however, most network tools are limited in their concept of what can be a network

- Networks are collections of nodes and edges
- Nodes are static integers or strings, and edges are binary or continuous values

NetworkX can represent **ANY** relationship supported by Python data types

Suppose we had data, or a data generating process, that was a time-series

- Current tools need kludges or hacks to add this data
- In NetworkX, we simply use the built-in Python `datetime` package to create a network of time-stamps

Simple time-series network

```
G=DiGraph()
# Create datetime object nodes
for v in xrange(num_nodes):
    G.add_node(datetime.now())
time_nodes=G.nodes()
# Add edges with 'time' attribute
for i in xrange(num_nodes):
    draws=random.uniform(0,1,num_nodes)
    for j in xrange(num_nodes):
        if i!=j and draws[j]<=p:
            G.add_edge(time_nodes[i],time_nodes[j],time=datetime.now())
...
# target source datetime_created
2010-05-25 13:38:42.515323 2010-05-25 13:38:42.515492
    {'time': datetime.datetime(2010, 5, 25, 13, 38, 42, 515752)}
...
```

Python's scientific computing holy trinity

Python's scientific computing holy trinity

Python's primary library for **mathematical and statistical** computing. Containing sub-libs for

- Numeric optimization
- Clustering
- Linear algebra
- ..and many others

Python's scientific computing holy trinity

Python's primary library for **mathematical and statistical** computing. Containing sub-libs for

- Numeric optimization
- Clustering
- Linear algebra
- ..and many others

The primary data type in `SciPy` is an array

- Data manipulation is similar to that of MATLAB

Python's scientific computing holy trinity

Python's primary library for **mathematical and statistical** computing. Containing sub-libs for

- Numeric optimization
- Clustering
- Linear algebra
- ..and many others

The primary data type in `SciPy` is an array

- Data manipulation is similar to that of MATLAB

`NumPy` is an extension of the `SciPy` data type to include **multidimensional arrays and matrices**

- Provides many functions for working on arrays and matrices
- Very useful for representing relational data

Python's scientific computing holy trinity

Python's primary library for **mathematical and statistical** computing. Containing sub-libs for

- Numeric optimization
- Clustering
- Linear algebra
- ..and many others

The primary data type in `SciPy` is an array

- Data manipulation is similar to that of MATLAB

`NumPy` is an extension of the `SciPy` data type to include **multidimensional arrays and matrices**

- Provides many functions for working on arrays and matrices
- Very useful for representing relational data

Both SciPy and `NumPy` rely on the C library `LAPACK` for very fast implementation

Python's scientific computing holy trinity

Python's primary library for **mathematical and statistical** computing. Containing sub-libs for

- Numeric optimization
- Clustering
- Linear algebra
- ..and many others

The primary data type in `SciPy` is an array

- Data manipulation is similar to that of MATLAB

`NumPy` is an extension of the `SciPy` data type to include **multidimensional arrays and matrices**

- Provides many functions for working on arrays and matrices
- Very useful for representing relational data

Both SciPy and `NumPy` rely on the C library `LAPACK` for very fast implementation

`matplotlib` is **primary plotting library in Python**

- Supports 2- and 3-D plotting
- API allows embedding in apps

Python's scientific computing holy trinity

Python's primary library for **mathematical and statistical** computing. Containing sub-libs for

- Numeric optimization
- Clustering
- Linear algebra
- ..and many others

The primary data type in `SciPy` is an array

- Data manipulation is similar to that of MATLAB

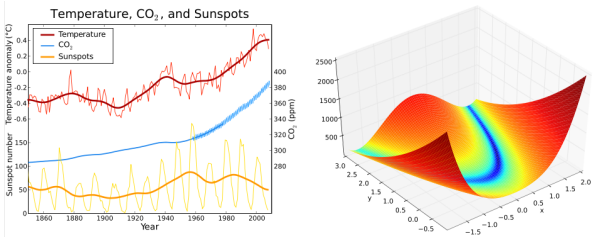
`NumPy` is an extension of the `SciPy` data type to include **multidimensional arrays and matrices**

- Provides many functions for working on arrays and matrices
- Very useful for representing relational data

Both SciPy and `NumPy` rely on the C library `LAPACK` for very fast implementation

`matplotlib` is **primary plotting library in Python**

- Supports 2- and 3-D plotting
- API allows embedding in apps



Python's scientific computing holy trinity

Python's primary library for **mathematical and statistical** computing. Containing sub-libs for

- Numeric optimization
- Clustering
- Linear algebra
- ..and many others

The primary data type in `SciPy` is an array

- Data manipulation is similar to that of MATLAB

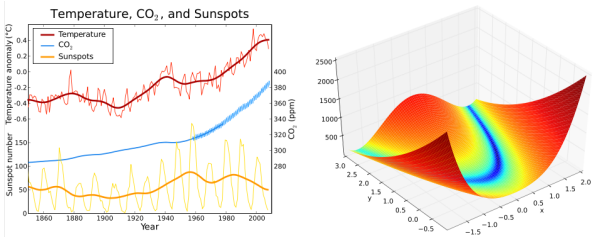
`NumPy` is an extension of the `SciPy` data type to include **multidimensional arrays and matrices**

- Provides many functions for working on arrays and matrices
- Very useful for representing relational data

Both SciPy and `NumPy` rely on the C library `LAPACK` for very fast implementation

`matplotlib` is **primary plotting library in Python**

- Supports 2- and 3-D plotting
- API allows embedding in apps



All graphics are highly customizable and professional publication ready

Python's scientific computing holy trinity

SciPy.org Sponsored By ENTHOUGHT

Python's primary library for **mathematical and statistical** computing. Containing sub-libs for

- Numeric optimization
- Clustering
- Linear algebra
- ..and many others

The primary data type in `SciPy` is an array

- Data manipulation is similar to that of MATLAB

NumPy

`NumPy` is an extension of the `SciPy` data type to include **multidimensional arrays and matrices**

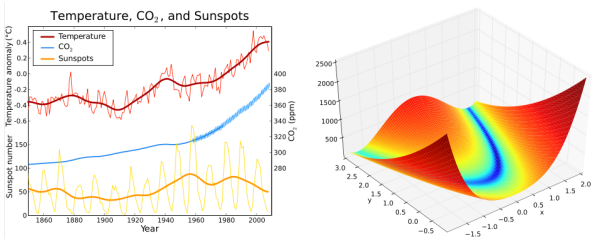
- Provides many functions for working on arrays and matrices
- Very useful for representing relational data

Both SciPy and `NumPy` rely on the C library `LAPACK` for very fast implementation

matplotlib

`matplotlib` is **primary plotting library in Python**

- Supports 2- and 3-D plotting
- API allows embedding in apps



All graphics are highly customizable and professional publication ready

NetworkX uses this entire suite to store, analyze and visualize networks

Exporting to GraphViz in NetworkX

NetworkX is designed to be an open-source all-purpose network manipulation and analysis tool

- Historically, the focus has not been on visualization

While there are several options for visualization in NetworkX, perhaps the best is its ability to read and write `GraphViz` files

- GraphViz is an open-source tool designed specifically for drawing graphs from the DOT language
- NetworkX works directly with GV using the `pygraphviz` package

Load Sampson data and visualize with graphviz

```
# Load Sampson monastery data from edglist
>>> g2=read_edgelist("samp_like_el.txt",delimiter="\ t",create_using=DiGraph())
>>> info(g2)
Name:
Type:                   DiGraph
Number of nodes:        18
Number of edges:        55
Average in degree:      3.0556
Average out degree:     3.0556
# Convert to pygraphviz type
>>> g2_gv=to_agraph(g2)
# Output DOT file and draw using dot layout
>>> g2_gv.write("samp_like_dot.dot")
>>> g2_gv.draw('samp_like.png',prog='dot')
```

Exporting to GraphViz in NetworkX

NetworkX is designed to be an open-source all-purpose network manipulation and analysis tool

- Historically, the focus has not been on visualization

While there are several options for visualization in NetworkX, perhaps the best is its ability to read and write `GraphViz` files

- GraphViz is an open-source tool designed specifically for drawing graphs from the DOT language
- NetworkX works directly with GV using the `pygraphviz` package

Load Sampson data and visualize with graphviz

```
# Load Sampson monastery data from edglist
>>> g2=read_edgelist("samp_like_el.txt",delimiter="\ t",create_using=DiGraph())
>>> info(g2)
Name:
Type:                   DiGraph
Number of nodes:        18
Number of edges:        55
Average in degree:      3.0556
Average out degree:     3.0556
# Convert to pygraphviz type
>>> g2_gv=to_agraph(g2)
# Output DOT file and draw using dot layout
>>> g2_gv.write("samp_like_dot.dot")
>>> g2_gv.draw('samp_like.png',prog='dot')
```

Exporting to GraphViz in NetworkX

NetworkX is designed to be an open-source all-purpose network manipulation and analysis tool

- Historically, the focus has not been on visualization

While there are several options for visualization in NetworkX, perhaps the best is its ability to read and write `GraphViz` files

- GraphViz is an open-source tool designed specifically for drawing graphs from the DOT language
- NetworkX works directly with GV using the `pygraphviz` package

Load Sampson data and visualize with graphviz

```
# Load Sampson monastery data from edglist
>>> g2=read_edgelist("samp_like_el.txt",delimiter="\ t",create_using=DiGraph())
>>> info(g2)
Name:
Type:                   DiGraph
Number of nodes:        18
Number of edges:        55
Average in degree:      3.0556
Average out degree:     3.0556
# Convert to pygraphviz type
>>> g2_gv=to_agraph(g2)
# Output DOT file and draw using dot layout
>>> g2_gv.write("samp_like_dot.dot")
>>> g2_gv.draw('samp_like.png',prog='dot')
```

Exporting to GraphViz in NetworkX

NetworkX is designed to be an open-source all-purpose network manipulation and analysis tool

- Historically, the focus has not been on visualization

While there are several options for visualization in NetworkX, perhaps the best is its ability to read and write `GraphViz` files

- GraphViz is an open-source tool designed specifically for drawing graphs from the DOT language
- NetworkX works directly with GV using the `pygraphviz` package

Load Sampson data and visualize with graphviz

```
# Load Sampson monastery data from edglist
>>> g2=read_edgelist("samp_like_el.txt",delimiter="\ t",create_using=DiGraph())
>>> info(g2)
Name:
Type:                   DiGraph
Number of nodes:        18
Number of edges:        55
Average in degree:      3.0556
Average out degree:     3.0556
# Convert to pygraphviz type
>>> g2_gv=to_agraph(g2)
# Output DOT file and draw using dot layout
>>> g2_gv.write("samp_like_dot.dot")
>>> g2_gv.draw('samp_like.png',prog='dot')
```

Exporting to GraphViz in NetworkX

NetworkX is designed to be an open-source all-purpose network manipulation and analysis tool

- Historically, the focus has not been on visualization

While there are several options for visualization in NetworkX, perhaps the best is its ability to read and write `GraphViz` files

- GraphViz is an open-source tool designed specifically for drawing graphs from the DOT language
- NetworkX works directly with GV using the `pygraphviz` package

Load Sampson data and visualize with graphviz

```
# Load Sampson monastery data from edglist
>>> g2=read_edgelist("samp_like_el.txt",delimiter="\ t",create_using=DiGraph())
>>> info(g2)
Name:
Type:                   DiGraph
Number of nodes:        18
Number of edges:        55
Average in degree:      3.0556
Average out degree:     3.0556
# Convert to pygraphviz type
>>> g2_gv=to_agraph(g2)
# Output DOT file and draw using dot layout
>>> g2_gv.write("samp_like_dot.dot")
>>> g2_gv.draw('samp_like.png',prog='dot')
```