Module II - Basic Analysis

Drew Conway — Department of Politics

NEW YORK UNIVERSITY

June 29, 2010

Agenda for Module IV

Loading data from multiple sources

- Local network data files
- Connecting to a database
- Building directly from the Internet

Agenda for Module IV

Loading data from multiple sources

- Local network data files
- Connecting to a database
- Building directly from the Internet

Brief review of Python `dict` data type

- Why it is so useful
- How `NetworkX` utilizes it

Agenda for Module IV

Loading data from multiple sources

- Local network data files
- Connecting to a database
- Building directly from the Internet

Brief review of Python `dict` data type

- Why it is so useful
- How `NetworkX` utilizes it

Running basic centralities

- Degree, Closeness, Betweeness Eigenvector
- Calculating degree distribution
- Plotting statistics using `matplotlib`
- Calculating cliques, clustering and transitivity

Agenda for Module IV

Loading data from multiple sources

- Local network data files
- Connecting to a database
- Building directly from the Internet

Brief review of Python `dict` data type

- Why it is so useful
- How `NetworkX` utilizes it

Running basic centralities

- Degree, Closeness, Betweeness Eigenvector
- Calculating degree distribution
- Plotting statistics using `matplotlib`
- Calculating cliques, clustering and transitivity

Outputting data into multiple formats

- Writing network data
- Saving network analysis statistics

Agenda for Module IV

Loading data from multiple sources

- Local network data files
- Connecting to a database
- Building directly from the Internet

Brief review of Python `dict` data type

- Why it is so useful
- How `NetworkX` utilizes it

Running basic centralities

- Degree, Closeness, Betweeness Eigenvector
- Calculating degree distribution
- Plotting statistics using `matplotlib`
- Calculating cliques, clustering and transitivity

Outputting data into multiple formats

- Writing network data
- Saving network analysis statistics

Basic visualization

- Review of `NetworkX`'s plotting algorithms
- Adding analysis to visualization

Loading a network file

As we have seen, one of the main advantages of working with `NetworkX` is that it can read many different network formats

- For those that are unfamiliar with working at the **command-line**, however, the process can be confusing

NX syntax for loading a file

$$>>> G \quad = \quad \text{read\_format}(\text{“path/to/file.txt”}, \quad \textit{...options...})$$

- G: Net variable
- read_format: NX function
- “path/to/file.txt”: file directory path
- *...options...*: Graph type, nodes type, etc.

Loading a network file

As we have seen, one of the main advantages of working with `NetworkX` is that it can read many different network formats

- For those that are unfamiliar with working at the **command-line**, however, the process can be confusing

NX syntax for loading a file

$$>>> G \quad = \quad \text{read\_format}(\text{“path/to/file.txt”}, \quad \textit{...options...})$$

- G: Net variable
- read_format: NX function, file directory path
- “path/to/file.txt”: file directory path
- *...options...*: Graph type, nodes type, etc.

Loading a network file

As we have seen, one of the main advantages of working with `NetworkX` is that it can read many different network formats

- For those that are unfamiliar with working at the **command-line**, however, the process can be confusing

NX syntax for loading a file

$$>>> G = \text{read\_format}(\text{“path/to/file.txt”}, \textit{...options...})$$

- G: Net variable
- read_format: NX function, file directory path
- “path/to/file.txt”: file directory path
- *...options...*: Graph type, nodes type, etc.

Loading a network file

As we have seen, one of the main advantages of working with `NetworkX` is that it can read many different network formats

- For those that are unfamiliar with working at the **command-line**, however, the process can be confusing

NX syntax for loading a file

```
>>> G  =  read_format("path/to/file.txt", ...options...)
```

- `G`: Net variable
- `read_format`: NX function
- `"path/to/file.txt"`: file directory path
- `...options...`: Graph type, nodes type, etc.

Loading a network file

As we have seen, one of the main advantages of working with `NetworkX` is that it can read many different network formats

- For those that are unfamiliar with working at the **command-line**, however, the process can be confusing

NX syntax for loading a file

```
>>> G    =    read_format("path/to/file.txt",        ...options...)
```

Net variable	NX function, file directory path	Graph type, nodes type, etc.

Let's try!

- We will load the edge list of Hartford drug users network
- Specify that the network be a directed graph, and the nodes be integers
- Use `info()` to check that data has been loaded correctly

Loading a network file

As we have seen, one of the main advantages of working with `NetworkX` is that it can read many different network formats

- For those that are unfamiliar with working at the **command-line**, however, the process can be confusing

NX syntax for loading a file

```
>>> G  =  read_format("path/to/file.txt",  ...options...)
```

Net variable	NX function, file directory path	Graph type, nodes type, etc.

Let's try!

- We will load the edge list of Hartford drug users network
- Specify that the network be a directed graph, and the nodes be integers
- Use `info()` to check that data has been loaded correctly

It's time to fire up your console and load Python!

Loading the Hartford drug users network

Starting NetworkX and loading data

```
>>> from networkx import *
>>> hartford=read_edgelist("../../data/hartford_drug.txt",create_using=DiGraph(),nodetype=int)
>>> info(hartford)
Name:
Type:                   DiGraph
Number of nodes:        212
Number of edges:        337
Average in degree:      1.5896
Average out degree:     1.5896
```

Loading the Hartford drug users network

Starting NetworkX and loading data

```
>>> from networkx import *
>>> hartford=read_edgelist("../../data/hartford_drug.txt",create_using=DiGraph(),nodetype=int)
>>> info(hartford)
Name:
Type:                   DiGraph
Number of nodes:        212
Number of edges:        337
Average in degree:      1.5896
Average out degree:     1.5896
```

What did we just do?

Loading the Hartford drug users network

Starting NetworkX and loading data

```
>>> from networkx import *
>>> hartford=read_edgelist("../../data/hartford_drug.txt",create_using=DiGraph(),nodetype=int)
>>> info(hartford)
Name:
Type:                   DiGraph
Number of nodes:        212
Number of edges:        337
Average in degree:      1.5896
Average out degree:     1.5896
```

What did we just do?

- Used the `read_edgelist` function to load EL file

Loading the Hartford drug users network

Starting NetworkX and loading data

```
>>> from networkx import *
>>> hartford=read_edgelist("../../data/hartford_drug.txt",create_using=DiGraph(),nodetype=int)
>>> info(hartford)
Name:
Type:                   DiGraph
Number of nodes:        212
Number of edges:        337
Average in degree:      1.5896
Average out degree:     1.5896
```

What did we just do?

- Used the `read_edgelist` function to load EL file
- Specified path to Hartford drug users file

Loading the Hartford drug users network

Starting NetworkX and loading data

```
>>> from networkx import *
>>> hartford=read_edgelist("../../data/hartford_drug.txt",create_using=DiGraph(),nodetype=int)
>>> info(hartford)
Name:
Type:                   DiGraph
Number of nodes:        212
Number of edges:        337
Average in degree:      1.5896
Average out degree:     1.5896
```

What did we just do?

- Used the `read_edgelist` function to load EL file
- Specified path to Hartford drug users file
- Used the `create_using` option to force NX to create as a directed graph

Loading the Hartford drug users network

Starting NetworkX and loading data

```
>>> from networkx import *
>>> hartford=read_edgelist("../../data/hartford_drug.txt",create_using=DiGraph(),nodetype=int)
>>> info(hartford)
Name:
Type:                   DiGraph
Number of nodes:        212
Number of edges:        337
Average in degree:      1.5896
Average out degree:     1.5896
```

What did we just do?

- Used the `read_edgelist` function to load EL file
- Specified path to Hartford drug users file
- Used the `create_using` option to force NX to create as a directed graph
- Used the `nodetype` option to force NX to store nodes as integers

Loading the Hartford drug users network

Starting NetworkX and loading data

```
>>> from networkx import *
>>> hartford=read_edgelist("../../data/hartford_drug.txt",create_using=DiGraph(),nodetype=int)
>>> info(hartford)
Name:
Type:                   DiGraph
Number of nodes:        212
Number of edges:        337
Average in degree:      1.5896
Average out degree:     1.5896
```

What did we just do?

- Used the `read_edgelist` function to load EL file
- Specified path to Hartford drug users file
- Used the `create_using` option to force NX to create as a directed graph
- Used the `nodetype` option to force NX to store nodes as integers
- Used the `info` function to check that it all worked

Loading the Hartford drug users network

Starting NetworkX and loading data

```
>>> from networkx import *
>>> hartford=read_edgelist("../../data/hartford_drug.txt",create_using=DiGraph(),nodetype=int)
>>> info(hartford)
Name:
Type:                   DiGraph
Number of nodes:        212
Number of edges:        337
Average in degree:      1.5896
Average out degree:     1.5896
```

What did we just do?

- Used the `read_edgelist` function to load EL file
- Specified path to Hartford drug users file
- Used the `create_using` option to force NX to create as a directed graph
- Used the `nodetype` option to force NX to store nodes as integers
- Used the `info` function to check that it all worked

Some formats may have more or less options, **always check the documentations!**