Social Network Analysis in R

Drew Conway

New York University - Department of Politics

August 6, 2009

Introduction

Why use R to do SNA?

- Review of SNA software
- Pros and Cons of SNA in R
- Comparison of SNA in R vs. Python

Examples of SNA in R

- Basic SNA - computing centrality metrics and identifying key actors
- Visualization - examples using igraph's built-in viz functions

Additional Resources

- Online Tutorials
- Helpful experts

SNA software landscape

The number of software suites and packages available for conducting social network analysis has exploded over the past ten years

- In general, this software can be categorized in two ways:
  - **Type** - many SNA tools are developed to be standalone applications, while others are language specific packages
  - **Intent** - consumers and producer of SNA come from a wide range of technical expertise and/or need, therefore, there exist simple tools for data collection and basic analysis, as well as complex suites for advanced research

	Standalone Apps	*Modules & Packages*
Basic	- ORA (Windows) - Analyst Notebook (Windows) - KrakPlot (Windows)	- libSNA (Python) - UrlNet (Python) - NodeXL (MS Excel)
Advanced	- UCINet (Windows) - Pajek (Multi) - Network Workbench (Multi)	- NetworkX (Python) - JUNG (Java) - igraph (Python, R & Ruby)

Pros and Cons of SNA in R

Pros

Cons

Pros and Cons of SNA in R

Pros

Diversity of tools available in R

- Analysis - sna: sociometric data; `RBGL`: Binding to Boost Graph Lib
- Simulation - `ergm`: exponential random graph; `networksis`: bipartite networks
- Specific use - `degreenet`: degree distribution; `tnet`: weighted networks

Cons

Pros and Cons of SNA in R

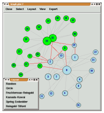
Pros

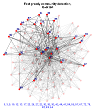
Diversity of tools available in R

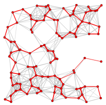
- Analysis - sna: sociometric data; `RBGL`: Binding to Boost Graph Lib
- Simulation - `ergm`: exponential random graph; `networksis`: bipartite networks
- Specific use - `degreenet`: degree distribution; `tnet`: weighted networks

Built-in visualization tools

- Take advantage of R's built-in graphics tools







Cons

Pros and Cons of SNA in R

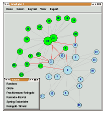
Pros

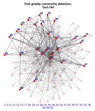
Diversity of tools available in R

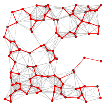
- Analysis - sna: sociometric data; `RBGL`: Binding to Boost Graph Lib
- Simulation - `ergm`: exponential random graph; `networksis`: bipartite networks
- Specific use - `degreenet`: degree distribution; `tnet`: weighted networks

Built-in visualization tools

- Take advantage of R's built-in graphics tools







Immediate access to more statistical analysis

- Perform SNA and network based econometrics "under the same roof"

Cons

Pros and Cons of SNA in R

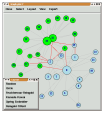
Pros

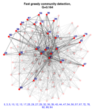
Diversity of tools available in R

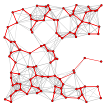
- Analysis - sna: sociometric data; `RBGL`: Binding to Boost Graph Lib
- Simulation - `ergm`: exponential random graph; `networksis`: bipartite networks
- Specific use - `degreenet`: degree distribution; `tnet`: weighted networks

Built-in visualization tools

- Take advantage of R's built-in graphics tools







Immediate access to more statistical analysis

- Perform SNA and network based econometrics "under the same roof"

Cons

Steep learning curve for SNA novices

- As with most things in R, the network analysis packages were designed by analysts for analysts
- These tools require at least a moderate familiarity with network structures and basic metrics

Structural Holes

Burt's constraint is higher if ego has less, or mutually stronger related (i.e. more redundant) contacts. Burt's measure of constraint, C[i], of vertex i's ego network V[i]

Pros and Cons of SNA in R

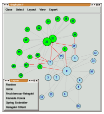
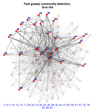
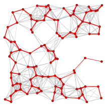
Pros

Diversity of tools available in R

- Analysis - sna: sociometric data; `RBGL`: Binding to Boost Graph Lib
- Simulation - `ergm`: exponential random graph; `networksis`: bipartite networks
- Specific use - `degreenet`: degree distribution; `tnet`: weighted networks

Built-in visualization tools

- Take advantage of R's built-in graphics tools

Immediate access to more statistical analysis

- Perform SNA and network based econometrics "under the same roof"

Cons

Steep learning curve for SNA novices

- As with most things in R, the network analysis packages were designed by analysts for analysts
- These tools require at least a moderate familiarity with network structures and basic metrics

Structural Holes

Burt's constraint is higher if ego has less, or mutually stronger related (i.e. more redundant) contacts. Burt's measure of constraint, C[i], of vertex i's ego network V[i]

Duplication and Interoperability

- Large variety of packages creates unnecessary duplication, which can be confusing
- Users may have to switch between packages because some function is supported in one but not the other
  - Ex. blockmodeling built into `sna` but not `igraph`

Direct Comparison of NetworkX (Python) vs. igraph

Using a randomly generated Barabasi-Albert network with 2,500 nodes and 4,996 edges we perform a side-by-side comparison of these two network analysis packages.[1]

[1] All tests performed on a 2.5 GHz Intel Core 2 Duo MacBook Pro with 4GB 667 MHz DDR2

Direct Comparison of NetworkX (Python) vs. igraph

Using a randomly generated Barabasi-Albert network with 2,500 nodes and 4,996 edges we perform a side-by-side comparison of these two network analysis packages.[1]

Test 1: Betweenness centrality

NX Code 1

```
def betweenness_test(G):
    start=time.clock()
    B=networkx.brandes_betweenness_centrality(G)
    return time.clock()-start
```

igraph Code 1

```
betweenness_test<-function(graph) {
    return(betweenness(graph)) }
system.time(B<-betweenness_test(G))
```

[1] All tests performed on a 2.5 GHz Intel Core 2 Duo MacBook Pro with 4GB 667 MHz DDR2

Direct Comparison of NetworkX (Python) vs. igraph

Using a randomly generated Barabasi-Albert network with 2,500 nodes and 4,996 edges we perform a side-by-side comparison of these two network analysis packages.[1]

Test 1: Betweenness centrality

NX Code 1

```
def betweenness_test(G):
    start=time.clock()
    B=networkx.brandes_betweenness_centrality(G)
    return time.clock()-start
```

Runtime: 55.89 sec

igraph Code 1

```
betweenness_test<-function(graph) {
    return(betweenness(graph)) }
system.time(B<-betweenness_test(G))
```

Runtime: 1.12 sec ✓

[1] All tests performed on a 2.5 GHz Intel Core 2 Duo MacBook Pro with 4GB 667 MHz DDR2

Direct Comparison of NetworkX (Python) vs. igraph

Using a randomly generated Barabasi-Albert network with 2,500 nodes and 4,996 edges we perform a side-by-side comparison of these two network analysis packages.[1]

Test 1: Betweenness centrality

NX Code 1

```
def betweenness_test(G):
    start=time.clock()
    B=networkx.brandes_betweenness_centrality(G)
    return time.clock()-start
```

Runtime: 55.89 sec

igraph Code 1

```
betweenness_test<-function(graph) {
    return(betweenness(graph)) }
system.time(B<-betweenness_test(G))
```

Runtime: 1.12 sec ✓

Test 2: Fruchterman-Reingold force-directed layout

NX Code 2

```
def layout_test(G,i=50):
    start=time.clock()
    v=networkx.layout.spring_layout(G,iterations=i)
    return time.clock()-start
```

igraph Code 2

```
layout_test<-function(graph,i=50) {
    return(layout.fruchterman.reingold(graph,niter=i)) }
system.time(v<-layout_test(G))
```

[1] All tests performed on a 2.5 GHz Intel Core 2 Duo MacBook Pro with 4GB 667 MHz DDR2

Direct Comparison of NetworkX (Python) vs. igraph

Using a randomly generated Barabasi-Albert network with 2,500 nodes and 4,996 edges we perform a side-by-side comparison of these two network analysis packages.[1]

Test 1: Betweenness centrality

NX Code 1

```
def betweenness_test(G):
    start=time.clock()
    B=networkx.brandes_betweenness_centrality(G)
    return time.clock()-start
```

Runtime: 55.89 sec

igraph Code 1

```
betweenness_test<-function(graph) {
    return(betweenness(graph)) }
system.time(B<-betweenness_test(G))
```

Runtime: 1.12 sec ✓

Test 2: Fruchterman-Reingold force-directed layout

NX Code 2

```
def layout_test(G,i=50):
    start=time.clock()
    v=networkx.layout.spring_layout(G,iterations=i)
    return time.clock()-start
```

Runtime: 1 min 6.13 sec

igraph Code 2

```
layout_test<-function(graph,i=50) {
    return(layout.fruchterman.reingold(graph,niter=i)) }
system.time(v<-layout_test(G))
```

Runtime: 9.03 sec ✓

[1] All tests performed on a 2.5 GHz Intel Core 2 Duo MacBook Pro with 4GB 667 MHz DDR2

Direct Comparison of NetworkX (Python) vs. igraph

Test 3: Graph diameter (maximum shortest path)

Direct Comparison of NetworkX (Python) vs. igraph

Test 3: Graph diameter (maximum shortest path)

NX Code 3

```
def diameter_test(G):
    start=time.clock()
    D=networkx.distance.diameter(G)
    return time.clock()-start
```

igraph Code 3

```
diameter_test<-function(graph) {
    return(diameter(graph)) }
system.time(D<-diameter_test(G))
```

Direct Comparison of NetworkX (Python) vs. igraph

Test 3: Graph diameter (maximum shortest path)

NX Code 3

```
def diameter_test(G):
    start=time.clock()
    D=networkx.distance.diameter(G)
    return time.clock()-start
```

Runtime: 15.66 sec

igraph Code 3

```
diameter_test<-function(graph) {
    return(diameter(graph)) }
system.time(D<-diameter_test(G))
```

Runtime: 0.42 sec ✓

Direct Comparison of NetworkX (Python) vs. igraph

Test 3: Graph diameter (maximum shortest path)

NX Code 3

```
def diameter_test(G):
    start=time.clock()
    D=networkx.distance.diameter(G)
    return time.clock()-start
```

Runtime: 15.66 sec

igraph Code 3

```
diameter_test<-function(graph) {
    return(diameter(graph)) }
system.time(D<-diameter_test(G))
```

Runtime: 0.42 sec ✓

Test 4: Find maximal cliques

NX Code 4

```
def max_clique_test(G):
    start=time.clock()
    C=networkx.clique.find_cliques(G)
    return time.clock()-start
```

igraph Code 4

```
max_clique_test<-function(graph) {
    return(maximal.cliques(graph)) }
system.time(M<-max_clique_test(G))
```

Direct Comparison of NetworkX (Python) vs. igraph

Test 3: Graph diameter (maximum shortest path)

NX Code 3

```
def diameter_test(G):
    start=time.clock()
    D=networkx.distance.diameter(G)
    return time.clock()-start
```

Runtime: 15.66 sec

igraph Code 3

```
diameter_test<-function(graph) {
    return(diameter(graph)) }
system.time(D<-diameter_test(G))
```

Runtime: 0.42 sec ✓

Test 4: Find maximal cliques

NX Code 4

```
def max_clique_test(G):
    start=time.clock()
    C=networkx.clique.find_cliques(G)
    return time.clock()-start
```

Runtime: 1.27 sec ✓

igraph Code 4

```
max_clique_test<-function(graph) {
    return(maximal.cliques(graph)) }
system.time(M<-max_clique_test(G))
```

Runtime: 8 min 24.95 sec

Finding maximal cliques can require several nested loops, which may contribute to R's poor performance

Comparing two network metrics to find key actors

Often social network analysis is used to identify key actors within a social group. To identify these actors, various centrality metrics can be computed based on a network's structure

- Degree (number of connections)
- Betweenness (number of shortest paths an actor is on)
- Closeness (relative distance to all other actors)
- Eigenvector centrality (leading eigenvector of sociomatrix)

One method for using these metrics to identify key actors is to plot actors' scores for Eigenvector centrality versus Betweenness. Theoretically, these metrics should be approximately linear; therefore, any non-linear outliers will be of note.

- An actor with very high betweenness but low EC may be a critical gatekeeper to a central actor
- Likewise, an actor with low betweenness but high EC may have unique access to central actors

Finding Key Actors with R

For this example, we will use the main component of the social network collected on drug users in Hartford, CT. The network has 194 nodes and 273 edges.

Finding Key Actors with R

For this example, we will use the main component of the social network collected on drug users in Hartford, CT. The network has 194 nodes and 273 edges.

Load the data into igraph

```
library(igraph)
G<-read.graph("drug_main.txt",format="edgelist")
G<-as.undirected(G)
# By default, igraph inputs edgelist data as a directed graph.
# In this step, we undo this and assume that all relationships are reciprocal.
```

Finding Key Actors with R

For this example, we will use the main component of the social network collected on drug users in Hartford, CT. The network has 194 nodes and 273 edges.

Load the data into igraph

```
library(igraph)
G<-read.graph("drug_main.txt",format="edgelist")
G<-as.undirected(G)
# By default, igraph inputs edgelist data as a directed graph.
# In this step, we undo this and assume that all relationships are reciprocal.
```

Store metrics in new data frame

```
cent<-data.frame(bet=betweenness(G),eig=evcent(G)$vector)
# evcent returns lots of data associated with the EC, but we only need the
# leading eigenvector
res<-lm(eig~bet,data=cent)$residuals
cent<-transform(cent,res=res)
# We will use the residuals in the next step
```

Finding Key Actors with R

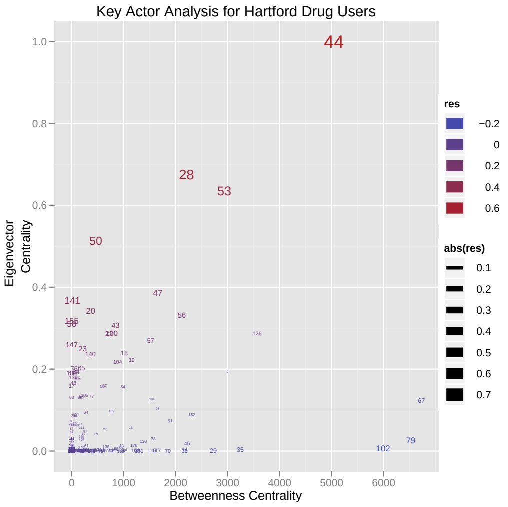
Plot the data

```
library(ggplot2)
# We use ggplot2 to make things a
# bit prettier
p<-ggplot(cent,aes(x=bet,y=eig,
    label=rownames(cent),colour=res,
    size=abs(res)))+xlab("Betweenness
    Centrality")+ylab("Eigenvector
    Centrality")
# We use the residuals to color and
# shape the points of our plot,
# making it easier to spot outliers.
p+geom_text()+opts(title="Key Actor
    Analysis for Hartford Drug Users")
# We use the geom_text function to plot
# the actors' ID's rather than points
# so we know who is who
```

Finding Key Actors with R

Plot the data

```
library(ggplot2)
# We use ggplot2 to make things a
# bit prettier
p<-ggplot(cent,aes(x=bet,y=eig,
    label=rownames(cent),colour=res,
    size=abs(res)))+xlab("Betweenness
    Centrality")+ylab("Eigenvector
    Centrality")
# We use the residuals to color and
# shape the points of our plot,
# making it easier to spot outliers.
p+geom_text()+opts(title="Key Actor
    Analysis for Hartford Drug Users")
# We use the geom_text function to plot
# the actors' ID's rather than points
# so we know who is who
```



Highlighting Key Actors

Using the drug network data, we will now identify the location of the key actors from the previous analysis

- We will use the same residual data from before to size the nodes and locate the key actors

First, however, we'll look at the network as a whole using igraph's Tcl/Tk interface

Highlighting Key Actors

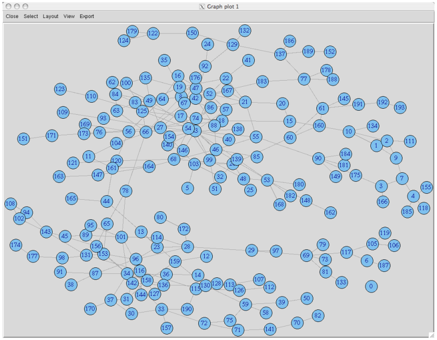
Using the drug network data, we will now identify the location of the key actors from the previous analysis

- We will use the same residual data from before to size the nodes and locate the key actors

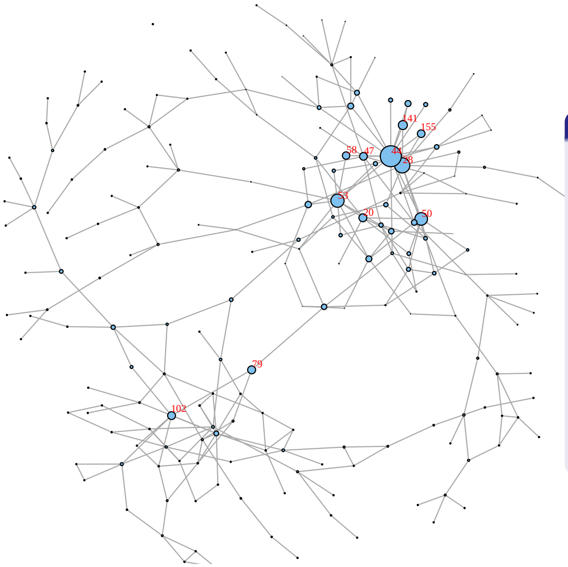
First, however, we'll look at the network as a whole using igraph's Tcl/Tk interface

Visualizing a network in igraph

```
library(igraph)
G<-as.undirected(read.graph(
    "drug_main.txt",type="edgelist"))
tklplot(G,layout=layout.fruchterman.reingold)
# This will open a new X11 window plot of G
```



Key Actor Plot



Network plot

```
# Create positions for all of
# the nodes w/ force directed
l<-layout.fruchterman.reingold(G,
    niter=500)
# Set the nodes’ size relative to
# their residual value
V(G)$size<-abs(res)*10
# Only display the labels of key
# players
nodes<-as.vector(V(G)+1)
# Key players defined as have a
# residual value >.25
nodes[which(abs(res)<.25)]<-NA
# Save plot as PDF
pdf(‘actor_plot.pdf’,pointsize=7)
plot(G,layout=l,vertex.label=nodes,
    vertex.label.dist=0.25,
    vertex.label.color=‘red’,edge.width=1)
dev.off()
```

Other Useful SNA Plots

Highlight the graph's longest geodesic

Find diameter

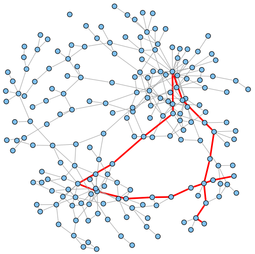
```
d<-get.diameter(G) # Find nodes on diameter path
# Reset G's node/width size for new graph
V(G)$size<-4
E(G)$width<-1
E(G)$color<-'dark grey'
E(G, path=d)$width<-3 # Set diameter path width to 3
E(G, path=d)$color<-'red' # and change color to red
# Save plot as PDF
pdf('diameter_plot.pdf')
plot(G,layout=l,vertex.label=NA)
dev.off()
```

Other Useful SNA Plots

Highlight the graph's longest geodesic

Find diameter

```
d<-get.diameter(G) # Find nodes on diameter path
# Reset G's node/width size for new graph
V(G)$size<-4
E(G)$width<-1
E(G)$color<-'dark grey'
E(G, path=d)$width<-3 # Set diameter path width to 3
E(G, path=d)$color<-'red' # and change color to red
# Save plot as PDF
pdf('diameter_plot.pdf')
plot(G,layout=l,vertex.label=NA)
dev.off()
```

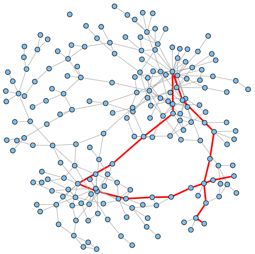


Other Useful SNA Plots

Highlight the graph's longest geodesic

Find diameter

```
d<-get.diameter(G) # Find nodes on diameter path
# Reset G's node/width size for new graph
V(G)$size<-4
E(G)$width<-1
E(G)$color<-'dark grey'
E(G, path=d)$width<-3 # Set diameter path width to 3
E(G, path=d)$color<-'red' # and change color to red
# Save plot as PDF
pdf('diameter_plot.pdf')
plot(G,layout=l,vertex.label=NA)
dev.off()
```



Extract the 2-core

K-core Analysis

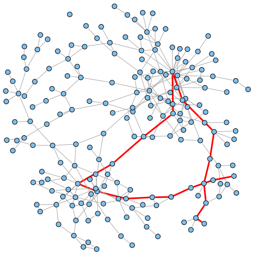
```
# Find each actor's coreness
cores<-graph.coreness(G)
# Extract 2-core, to eliminate pendants and pendant chains
G2<-subgraph(G,as.vector(which(cores>1))-1)
V(G2)$size<-4
l2<-layout.fruchterman.reingold(G2,niter=500)
# Save plot as a PDF
pdf('2core.pdf',pointsize=7)
plot(G2,layout=l2)
dev.off()
```

Other Useful SNA Plots

Highlight the graph's longest geodesic

Find diameter

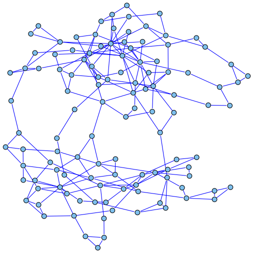
```
d<-get.diameter(G) # Find nodes on diameter path
# Reset G's node/width size for new graph
V(G)$size<-4
E(G)$width<-1
E(G)$color<-‘dark grey’
E(G, path=d)$width<-3 # Set diameter path width to 3
E(G, path=d)$color<-‘red’ # and change color to red
# Save plot as PDF
pdf(‘diameter_plot.pdf’)
plot(G,layout=l,vertex.label=NA)
dev.off()
```



Extract the 2-core

K-core Analysis

```
# Find each actor's coreness
cores<-graph.coreness(G)
# Extract 2-core, to eliminate pendants and pendant chains
G2<-subgraph(G,as.vector(which(cores>1))-1)
V(G2)$size<-4
l2<-layout.fruchterman.reingold(G2,niter=500)
# Save plot as a PDF
pdf(‘2core.pdf’,pointsize=7)
plot(G2,layout=l2)
dev.off()
```



Online Resources

igraph

- Network Analysis with igraph
- Excellent resource for learning how to use igraph in R, but also reviews many of the basic concepts of SNA

statnet

- Statnet Users Guide
- This package combines functionality from several popular R packages for SNA, and the online users guide contains reference material for:
  - network: A package for managing relational data in R
  - ergm: A package to fit, simulate and diagnose exponential family models for networks
  - latentnet: a package for fitting latent cluster models for networks
  - sna: A package for social network analysis
  - dynamicnetwork and rSoNIA: Prototype packages for managing and animating longitudinal network data
  - networksis: A Package to Simulate Bipartite Graphs with Fixed Marginals Through Sequential Importance Sampling

Material from this presentation

- These slides are available for download at the NY HackR website under files
- The R and Python code and data used for the benchmarking and analysis examples are also available for download

Helpful Experts

Several experts in both SNA in R, and SNA more general are active online and can be very helpful for those trying these methods for the first time

- SNA in R Experts
  - Nicole Radziwill - networks researcher
    Web: http://qualityandinnovation.com/
    Twitter: @nicoleradziwill
  - Michael Bommarito - PhD student in political science at U Michigan
    Web: http://computationallegalstudies.com/
    Twitter: @mjbommar
- General SNA Help
  - Valdis Krebs - Business networks researcher and developer of InFlow
    Web: http://www.orgnet.com/
    Twitter: @valdiskrebs
  - Steve Borgatti - Professor at U Kentucky Business school and UCINET developer
    Web: http://www.steveborgatti.com/
    Twitter: @ittagroB

Drew's contact info

- **Email:** drew.conway@nyu.edu
- **Web:** http://www.drewconway.com/zia
- **Twitter**: @drewconway