In this session:

- We will learn about the igraph package for analyzing networks
- We will compute several descriptive measures of networks
- We will work on several network models seen in the theory class

1 Introduction

In this session we will introduce the igraph software package for network analysis. This guide uses R as the platform to using igraph, but there exist python bindings for igraph as well in case you prefer to use that. RStudio is an excellent IDE for R.

There are igraph bindings for Python and C++; it should be easy to translate the code in this guide into these other languages.

R is "a language and environment for statistical computing and graphics"[1] with a very active community of contributors. Available from http://www.r-project.org/.

RStudio is "a free and open source integrated development environment for R"[2]. It makes working with R more pleasant. Available from http://www.rstudio.com/.

igraph is "a free software package for creating and manipulating undirected and directed graphs"[3]. Available from http://igraph.sourceforge.net/.

The computers in the PC Lab should have these three components installed. If igraph is not installed, then you can do so through RStudio's install manager or directly through the command line with the install.packages instruction. Then, to load the library so that you can use its functionality you should introduce the following command into the console:

```
> library(igraph)
```

2 Basics

This section will cover the basic commands for creating, manipulating and visualizing graphs using igraph. It should also help as an introduction to the main R commands. If you are unfamiliar with R, there are many online tutorials. However, for this session there is very little programming involved so you are not required to learn a new programming language from scratch.

2.1 Creating graphs

The objects we study in this course are *graphs* (or *networks*). They consist of a set of *nodes* and a set of *edges*. As an example, if you type into the RStudio console the following command

```
g <- graph( c(1,2, 1,3, 2,3, 3,5), n=5 )
```

In this command, we are assigning to the variable g a graph that has nodes $V = \{1, 2, 3, 4, 5\}$ and has edges $E = \{(1, 2), (1, 3), (2, 3), (3, 5)\}$

The commands V(g) and E(g) print the list of nodes and edges of the graph g:

[1] From http://www.r-project.org/about.html
[2] From http://www.rstudio.com/ide/
[3] From http://igraph.sourceforge.net/introduction.html

```
> V(g)
Vertex sequence:
[1] 1 2 3 4 5
> E(g)
Edge sequence:

[1] 1 -> 2
[2] 1 -> 3
[3] 2 -> 3
[4] 3 -> 5
```

You can always try

```
> plot(g)
```

We will see later additional parameters to `plot`

You can add nodes and edges to an already existing graph, e.g.:

```
> g <- graph.empty() + vertices(letters[1:10], color="red")
> g <- g + vertices(letters[11:20], color="blue")
> g <- g + edges(sample(V(g), 30, replace=TRUE), color="green")
> V(g)
Vertex sequence:
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t"
> E(g)
Edge sequence:

[1]  q -> q
[2]  n -> a
[3]  j -> p
[4]  s -> h
[5]  f -> c
[6]  h -> f
[7]  g -> r
[8]  t -> t
[9]  j -> o
[10] c -> f
[11] i -> a
[12] o -> q
[13] c -> j
[14] r -> i
[15] a -> b
```

These lines create a graph g with 20 nodes and 15 random edges. Notice that nodes and edges can also have attributes, e.g. in this example we are assigning different colors to nodes. The command `sample` returns a vector containing a sample of 30 random vertices from g.

2.1.1 Loading graphs

We have seen how to create graphs from scratch, but most often we will be loading them from a file containing the graph in some sort of format. `igraph` handles many graph formats already. The simplest one is a file containing the edge list. For example, create a file `graph.txt` containing the following three edges:

```
0 1
1 2
2 3
```

We can create a graph using the command

```
> g <- read.graph("graph.txt", format="edgelist")
> V(g)
```

```
Vertex sequence:
[1] 1 2 3 4
> E(g)
Edge sequence:

[1] 1 -> 2
[2] 2 -> 3
[3] 3 -> 4
```

Notice that the node ids within `igraph` start with 1, but the input file expects the first id to be 0. We believe this is a bug in the implementation of `igraph`, but you should keep this in mind.

If you want the graph to be undirected, add the parameter `directed=FALSE`.

We can also access online graphs, e.g. the following command loads a *Pajek* graph from an online site

```
karate <- read.graph("http://cneurocvs.rmki.kfki.hu/igraph/karate.net", format="pajek")
```

2.1.2 Graph generators

`igraph` implements also many useful graph generators. We have already seen – or will see very soon – a few models in class, in particular: the Edös-Rényi model (ER), the Barabasi-Albert model (BA), and the Watts-Strogratz model (WS). The following commands generate graphs using these models:

```
er_graph <- erdos.renyi.game(100, 2/100)
ws_graph <- watts.strogatz.game(1, 100, 4, 0.05)
ba_graph <- barabasi.game(100)
```

Try changing the parameters in `erdos.renyi.game` (called n and p in theory), and see how many connected components we get. A fascinating theoretical result, called "The Birth of the Giant Component" is that in this model, above a certain threshold p in the order of $1/n$, there is a single large connected component with $\Omega(n)$ vertices, with every other having at most $O(\log n)$ vertices. For p slightly larger, the Giant Component quickly gathers almost all vertices.

2.2 Manipulating attributes in graphs

We can add attributes to nodes and edges of the graphs. These are useful for selecting certain types of nodes, and for visualization purposes.

```
> g <- erdos.renyi.game(10, 0.5)
> V(g)$color <- sample( c("red", "black"), vcount(g), rep=TRUE)
> E(g)$color <- "grey"
> red <- V(g)[ color == "red" ]
> bl <- V(g)[ color == "black" ]
> E(g)[ red %--% red ]$color <- "red"
> E(g)[ bl %--% bl ]$color <- "black"
```

What these commands do is to generate a random graph with 10 nodes, assigns random colors to the nodes, colors edges joining red nodes in red, and edges joining black nodes in black. All remaining edges are colored grey.

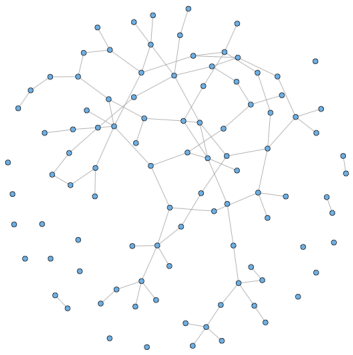
The next example assigns random weights to a lattice graph and then colors the ones having weight over 0.9 red, and the rest grey.

```
> g <- graph.lattice( c(10,10) )
> E(g)$weight <- runif(ecount(g))
> E(g)$color <- "grey"
> E(g)[ weight > 0.9 ]$color <- "red"
```

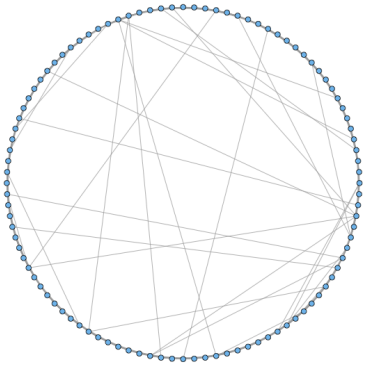
2.3 Visualizing graphs

A very important part in the analysis of networks is being able to *visualize* them. As an example the following commands render the three graphs depicted in the figure below.

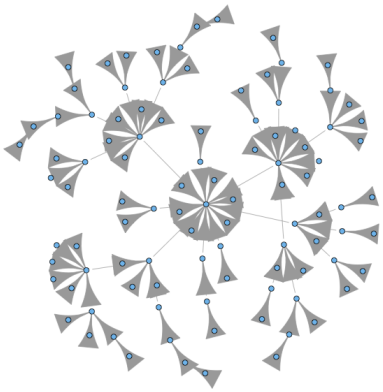
```
> er_graph <- erdos.renyi.game(100, 2/100)
> plot(er_graph, vertex.label=NA, vertex.size=3)
> ws_graph <- watts.strogatz.game(1, 100, 4, 0.05)
> plot(ws_graph, layout=layout.circle, vertex.label=NA, vertex.size=3)
> ba_graph <- barabasi.game(100)
> plot(ba_graph, vertex.label=NA, vertex.size=3)
```



Erdös-Rényi



Watts-Strogatz



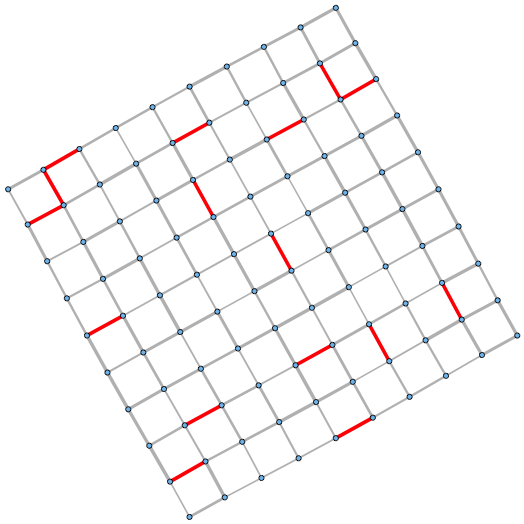
Barabási-Albert

The plot command is very flexible and has many parameters that control the behavior of the visualization. You can already see a few in the example above. For example, `vertex.label` controls the label written in the nodes, if set to `NA` then no text label is written. You can access all the parameters and their possible values through the help system by typing

```
> help(igraph.plotting)
```

As another example, consider adding attributes to edges for a nicer visualization:

```
> g <- graph.lattice( c(10,10) )
> E(g)$weight <- runif(ecount(g))
> E(g)$color <- "grey"
> E(g)[ weight > 0.9 ]$color <- "red"
> plot(g, vertex.size=2, vertex.label=NA, layout=layout.kamada.kawai,
edge.width=2+3*E(g)$weight)
```



2.4 Measuring graphs

There are many measures that help us understand and characterize networks. We have seen three in class – or will see them shortly: diameter (and average path length), clustering coefficient (or transitivity), and degree distribution. `igraph` provides functions that compute these measures for you. The functions are: `diameter`, `transitivity`,

average.path.length, degree, and degree.distribution. The examples below illustrate the usage of these functions.

For diameter and average.path.length

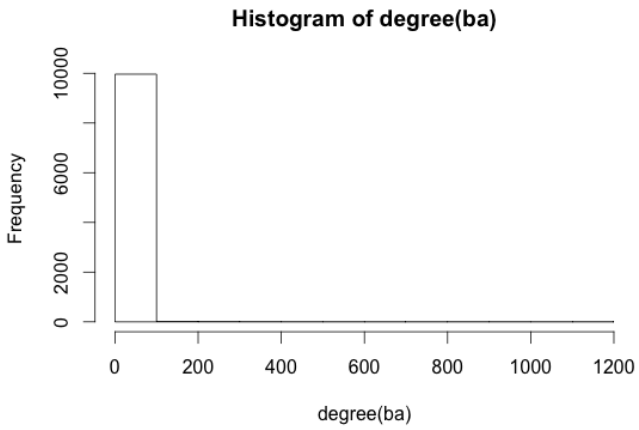
```
> g <- graph.lattice( length=100, dim=1, nei=4 )
> average.path.length(g)
[1] 8.79798
> diameter(g)
[1] 25
> g <- rewire.edges( g, each_edge(prob=0.05, loops = FALSE) )
> average.path.length(g)
[1] 3.132323
> diameter(g)
[1] 6
```

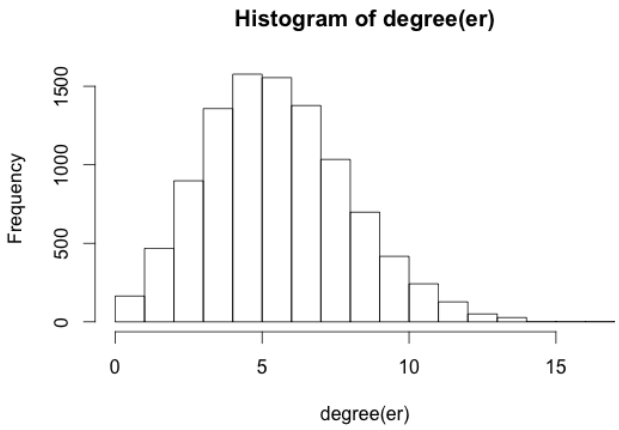
For transitivity

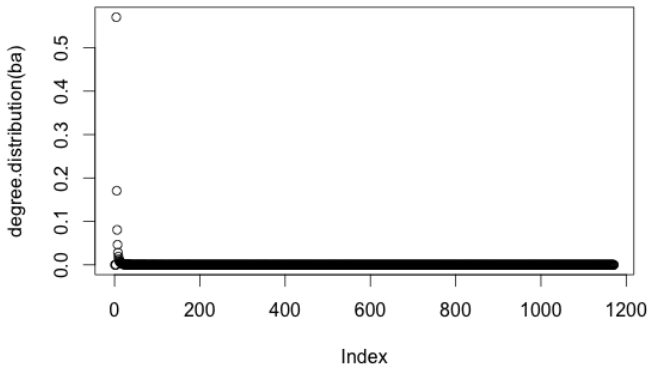
```
> ws <- watts.strogatz.game(1, 100, 4, 0.05)
> transitivity(ws)
[1] 0.5466147
> p_hat <- ecount(ws)/((vcount(ws)-1)*(vcount(ws))/2)
> p_hat
[1] 0.08
> er <- erdos.renyi.game(100, p_hat)
> transitivity(er)
[1] 0.08411215
```

For degree and degree.distribution

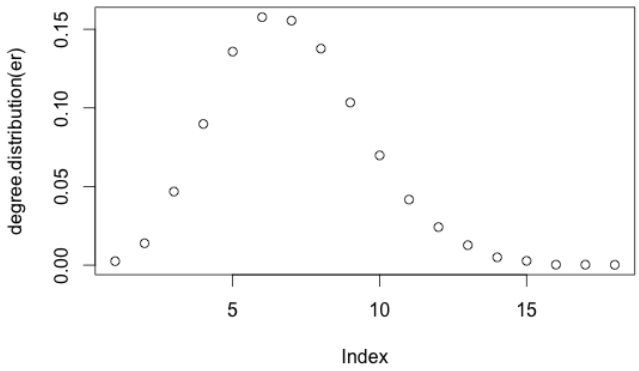
```
> g <- graph.ring(10)
> plot(g)
> degree(g)
 [1] 2 2 2 2 2 2 2 2 2 2

> ba <- barabasi.game(10000, m=3)
> p_hat <- ecount(ba)/ ((vcount(ba)-1)*vcount(ba)/2)
> er <- erdos.renyi.game(10000, p_hat)
> degree.distribution(er)
 [1] 0.0025 0.0139 0.0468 0.0898 0.1358 0.1577 0.1555 0.1377 0.1034 0.0698 0.0417 0.0242
[13] 0.0127 0.0050 0.0027 0.0003 0.0003 0.0002

> hist(degree(er))
> hist(degree(ba))
> plot(degree.distribution(er))
> plot(degree.distribution(ba))
```







Barabási-Albert

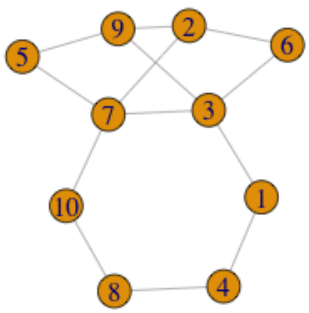


Erdös-Rényi

3 Node centrality

There are commands for computing degree, closeness and betweenness centrality, rank, and other measures. Some examples:

```
> set.seed(1)
> g <- sample_gnp(10, 3/10)
> plot(g, vertex.size=25, layout=layout.kamada.kawai)
```



```
> betweenness(g)
 [1]  6.0000000  3.2500000 13.2500000  2.5833333  0.9166667
 [6] 0.9166667 13.2500000  2.5833333  3.2500000
[10]  6.0000000

> edge_betweenness(g)
 [1] 12.500000  8.500000  4.250000  6.583333  7.166667
 [6] 9.250000  6.583333  5.666667  4.083333  7.166667
[11]  4.250000 12.500000  8.500000

> degree(g)
 [1] 2 3 4 2 2 2 4 2 3 2

> closeness(g)
 [1] 0.05263158 0.05263158 0.06666667 0.04347826 0.04761905
 [6] 0.04761905 0.06666667 0.04347826 0.05263158 0.05263158

> page.rank(g)$vector
 [1] 0.08274197 0.10913228 0.14429762 0.08724407 0.07658406
 [6] 0.07658406 0.14429762 0.08724407 0.10913228 0.08274197
```

4 Analyzing network models

In class you have seen three main random network models:

Erdős-Rényi model (ER model). The ER model takes two parameters: n, the number of vertices in the resulting network, and p, the probability of having an edge between any two pairs of nodes. A graph following this model is generated by connecting pairs of vertices with probability p, independently for each pair of vertices. Erdős-Rényi graphs have $\binom{n}{2}p$ edges in expectation.

Watts-Strogatz model (WS model). The WS model takes two parameters as well: n, the number of vertices in the resulting network, and p, the probability of rewiring the edges in the initial network. A graph following this model is generated by initially laying all nodes out in a circle, and connecting each node to its four closest nodes. After that, we randomly reconnect each edge with probability p.

Barabasi-Albert model (BA model). The BA model takes two parameters: n, the number of vertices in the resulting network, and m, the number of edges a "new" vertex brings to attach itself to existing nodes. A graph in this model is generated by adding new nodes according to the preferential attachment principle until the resulting graph has the desired size.

Your task is to do two of the three following alternatives:

1. Plot the clustering coefficient and the average shortest-path as a function of the parameter p of the WS model.
2. Plot the average shortest-path length as a function of the network size of the ER model.
3. Plot a histogram of the degree distribution of a BA network. What distribution does this follow? Can you describe it?

For option (1), notice that in order to include both values — average shortest path and clustering coefficient — in the same figure, the clustering coefficient and the average shortest-path values are normalized to be within the range $[0, 1]$. This is achieved by dividing the values by the value obtained at the left-most point, that is, when $p = 0$.

For option (2), you will have to experiment with appropriate values of p which may depend on the parameter n. You will notice that for large values of n your code may take too long, compute values for n that are reasonable for you. Also, make sure that you chose values for p that result (with high probability) in connected graphs. To achieve this, you can use a result from [**?**] stating (in the following, think of ϵ as a small positive real number):

- If $p < \frac{(1-\epsilon)\ln n}{n}$ then a graph in $G(n, p)$ will almost surely contain isolated vertices, and thus be disconnected

- If $p > \frac{(1+\epsilon)\ln n}{n}$ then a graph in $G(n,p)$ will almost surely be connected

For option (3), choose a network that is large enough so that results are what is expected from this model.

5 Deliverables

Rules: Same rules as in previous labs about working solo/in pairs and plagiarism.

To deliver: Please deliver your code and a brief report (2-3 pages max.) describing your results.

Procedure: Submit your work through the Racó.

Deadline: Work must be delivered within **3 weeks** from the lab. Late deliveries risk being penalized or not accepted at all. If you anticipate problems with the deadline, tell me as soon as possible.