A quick View on igraph
by
Pattabiraman

igraph

- **igraph is an open source collection of libraries for network analysis**
  - http://igraph.org

- **Supports: R, python and C**

- **Disclaimer: This tutorial is a very basic tutorial that serves only as an introduction to igraph**
  - We will focus our descriptions on R, but the usage of the library is very similar to other platforms

Network representation

- **There are many various ways to represent a network**
  - Most “popular”
    - ✓ Adjacency matrix
    - ✓ Edgelist
    - ✓ Adjacency list
- **Adjacency matrix representation is good for mathematical derivations but it is not very efficient for computations especially if networks are sparse**
  - In a sparse network most of the entries of the matrix will be 0

Network representation

- **Edgelist**
  - Store the edges of a network in a tuple format <node ID1, node ID2>
  - For example,

1 2
1 5
2 3
2 4
3 4
3 5
3 6

Network representation

- **Adjacency list**
  - Easier to work if network is large and sparse
  - There is one line for every node in the network that includes the IDs of the nodes he is connected to
  - For example, the previous network in adjacency list formaet would be:

1: 2,5
2: 1,3,4
3: 2,4,5,6
4: 2,3
5: 1,3
6: 3

Network representation

- **igraph supports multiple network representations**
  - Details of the actual format (e.g., punctuation etc.) might be different for different network libraries
    - ✓ However, the main format will be the same

- **Assuming that we have the previous network in an edgelist we can “read” it in igraph**

```
> library(igraph)
> g <- read.graph("sample_net",format="ncol”,directed=F)
```

Network representation

- **The network has been read an undirected (UN) and it has 6 nodes and 7 edges**
  - V(g) gives the nodes of the network
  - E(g) gives the edges of the network

```
> g
IGRAPH UN-- 6 7 --
+ attr: name (v/c)
```

```
> V(g)
Vertex sequence:
[1] "1" "2" "5" "3" "4" "6"
```

```
> E(g)
Edge sequence:

[1] 2 -- 1
[2] 5 -- 1
[3] 3 -- 2
[4] 4 -- 2
[5] 4 -- 3
[6] 3 -- 5
[7] 6 -- 3
```

Network representation

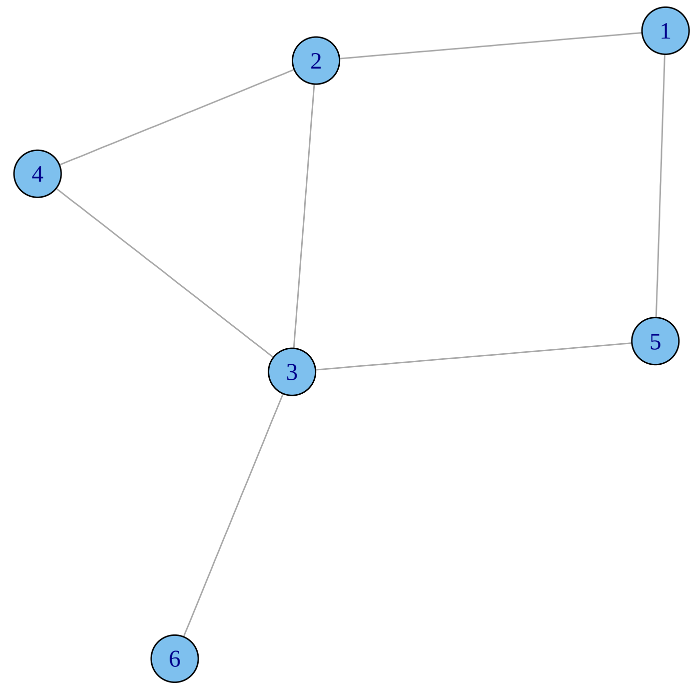
- **You can also get the adjacency matrix**

  > m <-get.adjacency(g)

```
6 x 6 sparse Matrix of class "dgCMatrix"
  1 2 5 3 4 6
1 . 1 1 . . .
2 1 . . 1 1 .
5 1 . . 1 . .
3 . 1 1 . 1 1
4 . 1 . 1 . .
6 . . . 1 . .
```

Visualize the network

> plot(g)



- **plot provides a simple network visualization**
  - There are more *advanced* libraries that give more control on the way the network is visualized

Connectance

```
> graph.density(g,loops=F)
[1] 0.4666667
```

- **The argument loops dictates whether to allow loop edges in the graph or not**
  - If it is true self-edges are considered possible

```
> graph.density(g,loops=T)
[1] 0.3333333
```

Node degree

```
> degree(g)
1 2 5 3 4 6
2 3 2 4 2 1
```

- **This provides a list with the degree of every node in the network**

Paths

```
> shortest.paths(g)
  1 2 5 3 4 6
1 0 1 1 2 2 3
2 1 0 2 1 1 2
5 1 2 0 1 2 2
3 2 1 1 0 1 1
4 2 1 2 1 0 2
6 3 2 2 1 2 0
```

- **shortest.paths provides the length of the shortest path between any two nodes in g**
- **If you want to get the actual paths (and not just their length) you can use the function get.shortest.paths**

Paths

- **We can also get the number of paths of a given length between two nodes by utilizing the adjacency matrix**

```
> m%*%m

6 x 6 sparse Matrix of class "dgCMatrix"
  1 2 5 3 4 6
1 2 . . 2 1 .
2 . 3 2 1 1 1
5 . 2 2 . 1 1
3 2 1 . 4 1 .
4 1 1 1 1 2 1
6 . 1 1 . 1 1
```

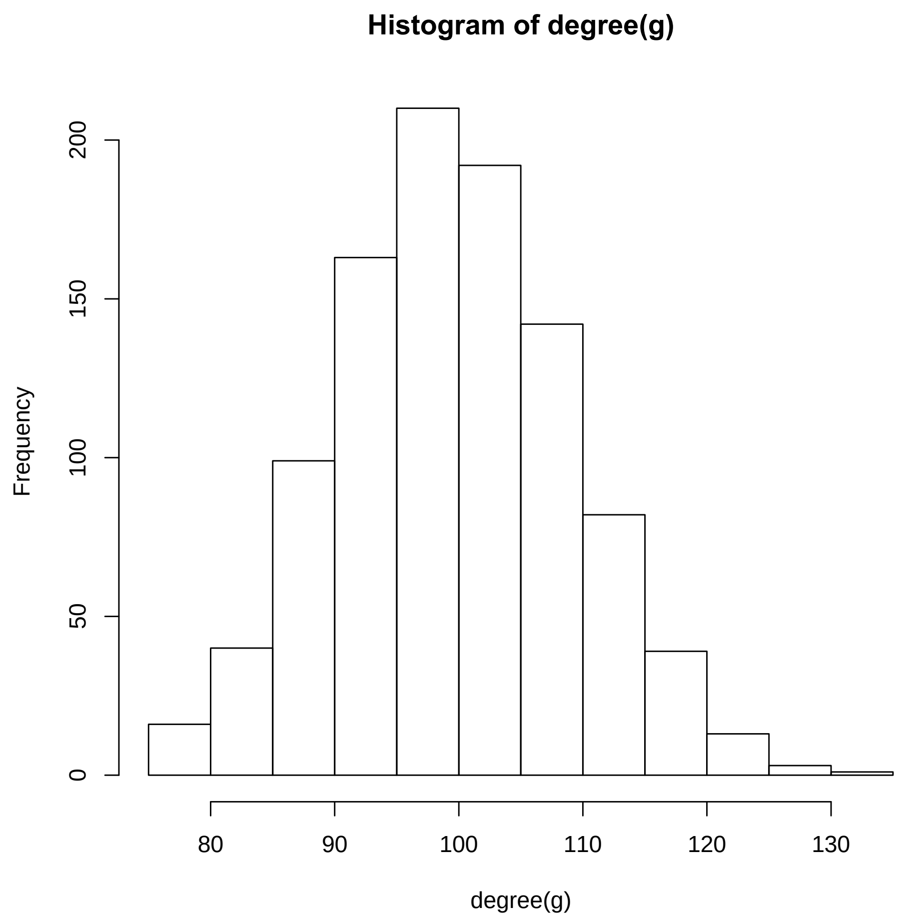
Random graphs

> g=erdos.renyi.game(1000,0.1,type="gnp")

> g

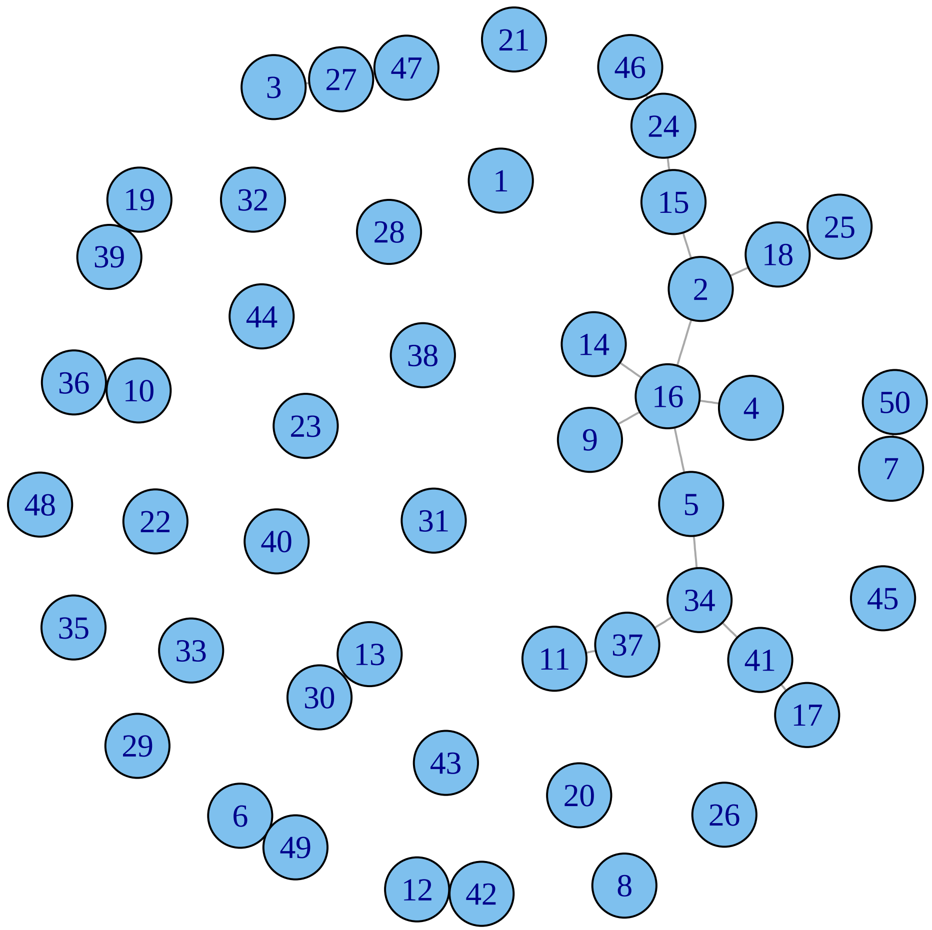
IGRAPH U--- 1000 50042 -- Erdos renyi (gnp) graph
+ attr: name (g/c), type (g/c), loops (g/x), p (g/n)

> hist(degree(g))



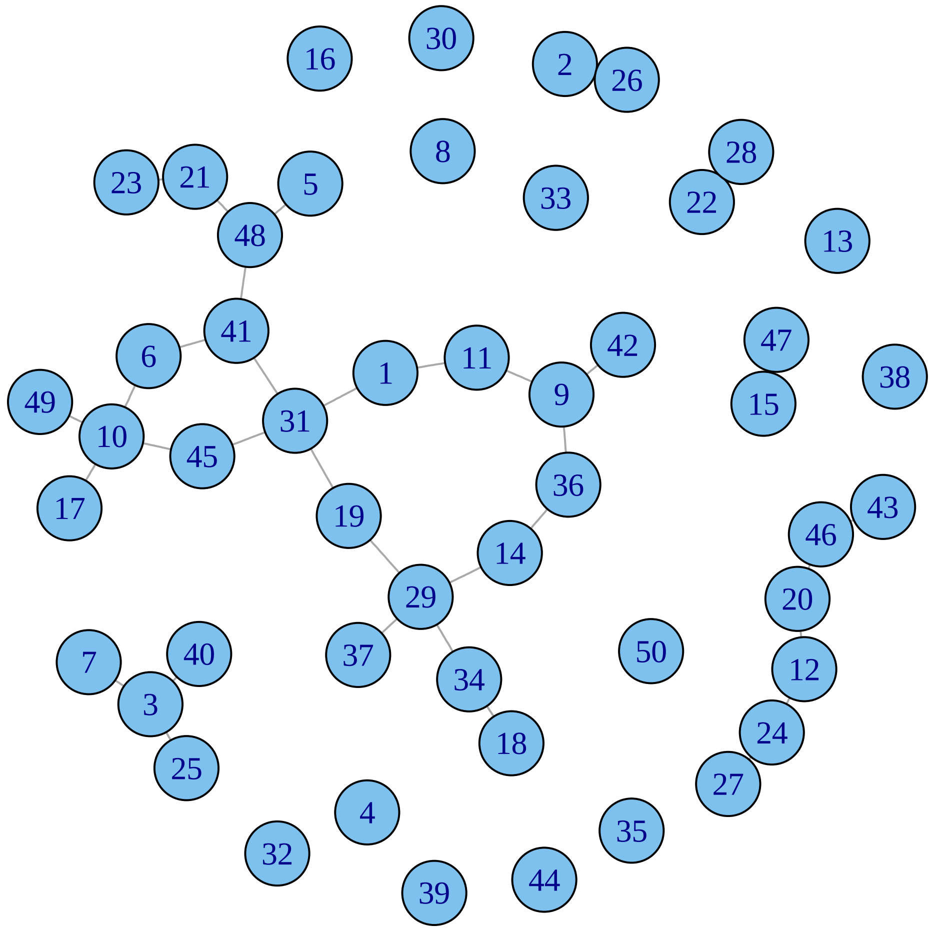
Random graphs

- **n=50 and p=0.01**



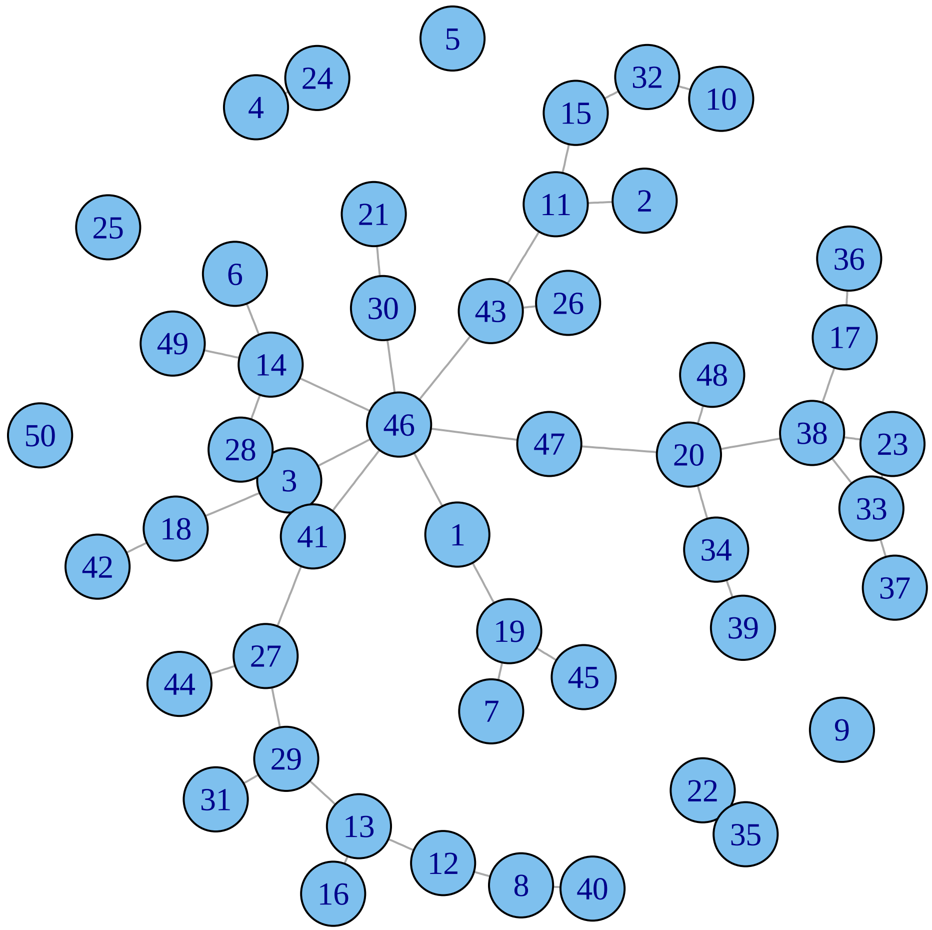
Random graphs

- n=50 and p=0.03



Random graphs

- **n=50 and p=0.05**



Random graphs

```
> for (p in 0:100){
g=erdos.renyi.game(50,p/100,"gnp")
s[p+1]=max(clusters(g)$csize)/50
}
plot(s, xlab="p", ylab="Giant component fraction")
lines(s)
```