16 - Networks and PageRank

ST 697 - Fall 2017

Contents

1 Network Intro **2**
- 1.1 Required R Packages . . . 2
- 1.2 Example: Zachary's karate club network . . . 2

2 Basic Network Concepts **3**
- 2.1 Basic Definitions . . . 3
- 2.2 Subgraphs . . . 4
- 2.3 Representations for Graphs . . . 4
- 2.4 Movement about a graph . . . 6
- 2.5 Bipartite graphs . . . 7
- 2.6 Graph Layout . . . 8

3 Graphs and Matrix Algebra **10**

4 Descriptive Analysis of Network Graph Characteristics **11**
- 4.1 Vertex Degree Distribution . . . 11
- 4.2 Density . . . 12
- 4.3 Vertex Centrality . . . 12

5 PageRank **15**
- 5.1 Random Surfer . . . 15
- 5.2 PageRank Details . . . 16

1 Network Intro

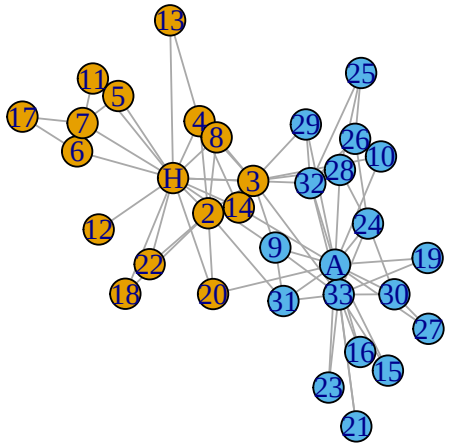
1.1 Required R Packages

We will be using the R packages of:
- `igraph` for network modeling
- `sand` for data (supplement to book *Statistical Analysis of Network Data with R* by Kolaczyk and Csárdi)

```
library(igraph)     # install.packages('igraph') if not installed
library(sand)       # install.packages('sand') if not installed
```

1.2 Example: Zachary's karate club network

```
library(igraphdata)
data(karate)
library(igraph)
plot(karate, layout=layout_with_fr(karate))
```



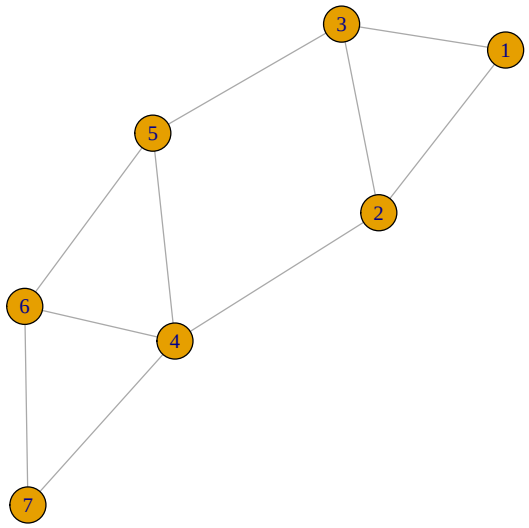

```
?karate
```

2 Basic Network Concepts

2.1 Basic Definitions

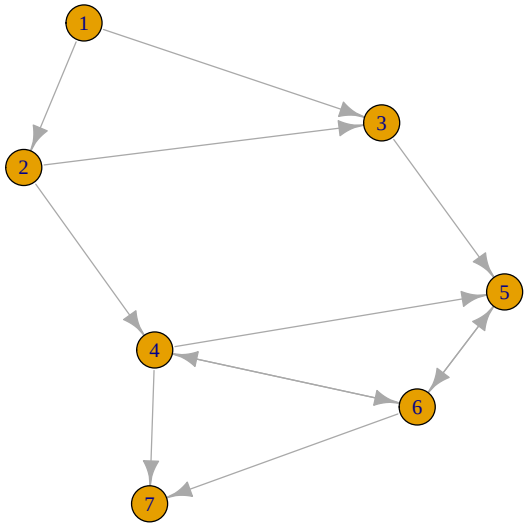
- A graph can be represented by $G = (V, E)$ where V are the set of vertices (also called nodes) and E is a set of edges (also called links).
- There are $|V|$ nodes and $|E|$ edges in G
- The edge set E is a collection of pairs, (u, v) where $u, v \in V$
  - For undirected graphs, (u, v) is same as (v, u).
  - For directed graphs (digraph), (u, v) is distinct from (v, u)

```
g <- graph.formula(1-2, 1-3, 2-3, 2-4, 3-5, 4-5, 4-6,
                   4-7, 5-6, 6-7)
E(g)                       # edges
#> + 10/10 edges from 75722d0 (vertex names):
#>  [1] 1--2 1--3 2--3 2--4 3--5 4--5 4--6 4--7 5--6 6--7
ecount(g)                  # number of edges
#> [1] 10
V(g)                       # vertices
#> + 7/7 vertices, named, from 75722d0:
#> [1] 1 2 3 4 5 6 7
vcount(g)                  # number of vertices
#> [1] 7
g.layout = layout_with_fr(g) # create layout (node coordinates)
plot(g, layout=g.layout)     # plot graph
```



We can consider the same graph, but with the edges directed

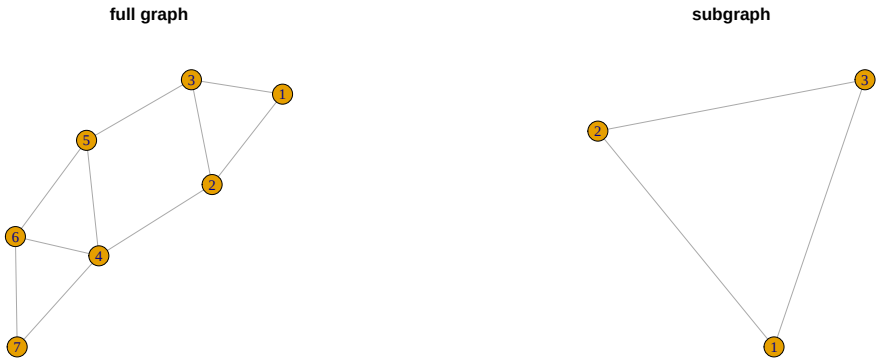
```
g2 <- graph.formula(1-+2, 1-+3, 2-+3, 2-+4, 3-+5, 4-+5, 4++6,
                    4-+7, 5++6, 6-+7)
E(g2)                      # edges
#> + 12/12 edges from 7598a10 (vertex names):
#>  [1] 1->2 1->3 2->3 2->4 3->5 4->5 4->6 4->7 5->6 6->4 6->5 6->7
V(g2)                      # vertices
#> + 7/7 vertices, named, from 7598a10:
#> [1] 1 2 3 4 5 6 7
plot(g2)                   # plot graph with random layout
```



2.2 Subgraphs

- A graph $H = (V_H, E_H)$ is a subgraph of $G = (V_G, E_G)$ if $V_H \subseteq V_G$ and $E_H \subseteq E_G$.
- An *induced subgraph* of graph G is a subgraph $G' = (V', E')$ where $V' \subseteq V$ is a prespecified subset of vertices and $E' \subseteq E$ is the collection of edges to be found in G among that subset of vertices.

```
g3 = induced_subgraph(g, v=1:3)    # only select vertices 1:3
plot(g, layout=g.layout, main='full graph')
plot(g3, main='subgraph')
```



2.3 Representations for Graphs

2.3.1 Edge List

An edge list is usually represented as a two-column matrix (or data.frame)

```
get.edgelist(g)
#>      [,1] [,2]
#> [1,] "1"  "2"
#> [2,] "1"  "3"
#> [3,] "2"  "3"
#> [4,] "2"  "4"
#> [5,] "3"  "5"
```

```
#>  [6,] "4"  "5"
#>  [7,] "4"  "6"
#>  [8,] "4"  "7"
#>  [9,] "5"  "6"
#> [10,] "6"  "7"
get.edgelist(g2)     # notice the additional entries
#>       [,1] [,2]
#>  [1,] "1"  "2"
#>  [2,] "1"  "3"
#>  [3,] "2"  "3"
#>  [4,] "2"  "4"
#>  [5,] "3"  "5"
#>  [6,] "4"  "5"
#>  [7,] "4"  "6"
#>  [8,] "4"  "7"
#>  [9,] "5"  "6"
#> [10,] "6"  "4"
#> [11,] "6"  "5"
#> [12,] "6"  "7"
```

2.3.2 Adjacency Matrix

An adjacency matrix is the $|V| \times |V|$ matrix, $\mathbf{A}$ such that

$$A_{ij} = \begin{cases} 1, & \text{if } \{i,j\} \in E, \\ 0, & \text{otherwise} \end{cases}$$

For undirected graphs, the adjacency matrix will by symmetric.

```
get.adjacency(g)     # binary and symmetric
#> 7 x 7 sparse Matrix of class "dgCMatrix"
#>   1 2 3 4 5 6 7
#> 1 . 1 1 . . . .
#> 2 1 . 1 1 . . .
#> 3 1 1 . . 1 . .
#> 4 . 1 . . 1 1 1
#> 5 . . 1 1 . 1 .
#> 6 . . . 1 1 . 1
#> 7 . . . 1 . 1 .
get.adjacency(g2)    # binary and not-symmetric
#> 7 x 7 sparse Matrix of class "dgCMatrix"
#>   1 2 3 4 5 6 7
#> 1 . 1 1 . . . .
#> 2 . . 1 1 . . .
#> 3 . . . . 1 . .
#> 4 . . . . 1 1 1
#> 5 . . . . . 1 .
#> 6 . . . 1 1 . 1
#> 7 . . . . . . .
```

2.3.3 Adjacency List

The adjacency list is an array (in R, a list) of size $|V|$, where the elements of the list indicate the set of vertices that are adjacent. It is essentially the sparse representation of the adjacency matrix.

```
get.adjlist(g)
#> $`1`
#> + 2/7 vertices, named, from 75722d0:
#> [1] 2 3
#>
#> $`2`
#> + 3/7 vertices, named, from 75722d0:
#> [1] 1 3 4
#>
#> $`3`
#> + 3/7 vertices, named, from 75722d0:
#> [1] 1 2 5
#>
#> $`4`
#> + 4/7 vertices, named, from 75722d0:
#> [1] 2 5 6 7
#>
#> $`5`
#> + 3/7 vertices, named, from 75722d0:
#> [1] 3 4 6
#>
#> $`6`
#> + 3/7 vertices, named, from 75722d0:
#> [1] 4 5 7
#>
#> $`7`
#> + 2/7 vertices, named, from 75722d0:
#> [1] 4 6
```

2.4 Movement about a graph

- A *walk* on a graph G describes a sequence of adjacent vertices $(v_0, v_1, ..., v_n)$, where each v_i is connected to v_{i+1} by an edge.
- *Trails* are walks without repeated edges
- *Paths* are walks without repeated vertices or edges
- A *cycle* is a trail with the same starting and ending vertices
  - An *acyclic* graph is one with no cycles (usually for directed graphs)
- A *connected* graph is one where a walk exists between every pair of vertices
- *Geodesic distance* (also called *number of hops*) is the length of the shortest path between two vertices

```
distances(g)
#>   1 2 3 4 5 6 7
#> 1 0 1 1 2 2 3 3
#> 2 1 0 1 1 2 2 2
#> 3 1 1 0 2 1 2 3
#> 4 2 1 2 0 1 1 1
```

```
#> 5 2 2 1 1 0 1 2
#> 6 3 2 2 1 1 0 1
#> 7 3 2 3 1 2 1 0
```

2.4.1 Weighted Edges

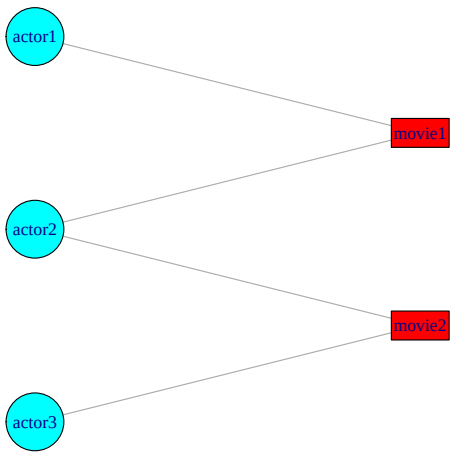
- Edges can have attributes that describe the nature of the connection between two vertices
- An example is to assign edge weights $\{w_{ij} : e_{ij} \in E\}$
  - Weights can be measurements of things like: flow rate, number of transactions, call time, travel speed, etc.
- More generally, consider the weight matrix W, which is the $|V| \times |V|$ matrix containing the edge weights. The weights will be $W_{ij} = 0$ if $A_{ij} = 0$.
  - The adjacency matrix is a special case of weight matrix with binary weights

2.5 Bipartite graphs

A bipartite graph (also called *two-mode*) is a graph $G = (V, E)$ such that the vertex set V may be partitioned into two disjoint sets V_1 and V_2, and each edge in E has one endpoint in V_1 and the other in V_2.

```
g.bip <- graph.formula(actor1:actor2:actor3,
   movie1:movie2, actor1:actor2 - movie1,
   actor2:actor3 - movie2)
V(g.bip)$type <- grepl("^movie", V(g.bip)$name)
plot(g.bip, layout=-layout.bipartite(g.bip)[,2:1],
   vertex.size=30, vertex.shape=ifelse(V(g.bip)$type,
     "rectangle", "circle"),
   vertex.color=ifelse(V(g.bip)$type, "red", "cyan"))

get.incidence(g.bip)        # get the incidence matrix
#>        movie1 movie2
#> actor1      1      0
#> actor2      1      1
#> actor3      0      1
```

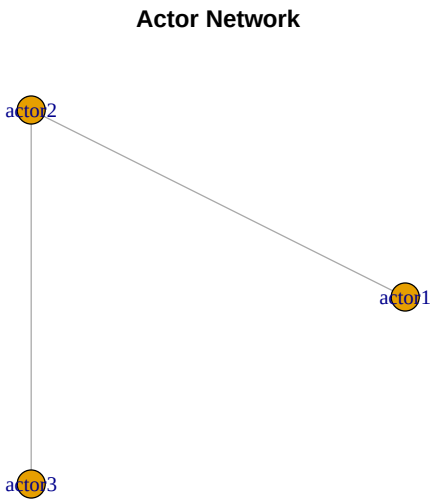


Some examples:
- Membership networks: V_1 are the members and V_2 the organizations

- Recommender data: V_1 are the movies and V_2 the reviewers
- Market basket data: V_1 are the shoppers and V_2 are the items in the store
- Travel: V_1 are the people and V_2 are the places they visit

A bipartite graph can be accompanied by the induced subgraph formed by connecting the vertices, say V_1, by assigning an edge to vertices that edges in E to at least one common vertex in V_2

```
plot(bipartite.projection(g.bip)$proj1,main="Actor Network",
     layout=layout_in_circle)
```



2.6 Graph Layout

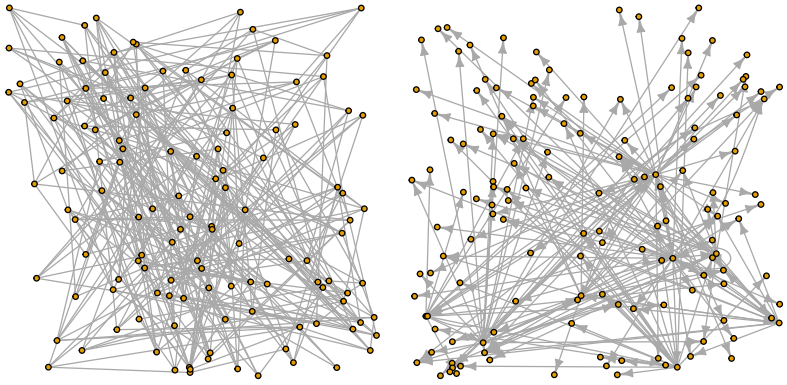
Graph layouts are projections of the vertices and edges into some space. Different layouts reveal different aspects of a graph.
- Note: 2D layouts can be very misleading; don't trust your eyes
- Choose the layout to help reveal the structure. In `igraph`,
- `layout.fructerman.reingold` is a spring-embedder method
- `layout.kamada.kawai` is based on multidimensional scaling (MDS)
- These will be a function of the *distance* between vertices

```
#- Aids blog network
library(sand)
data(aidsblog)
aidsblog = upgrade_graph(aidsblog)

#- lattice data
g.l <- graph.lattice(c(5, 5, 5))
```

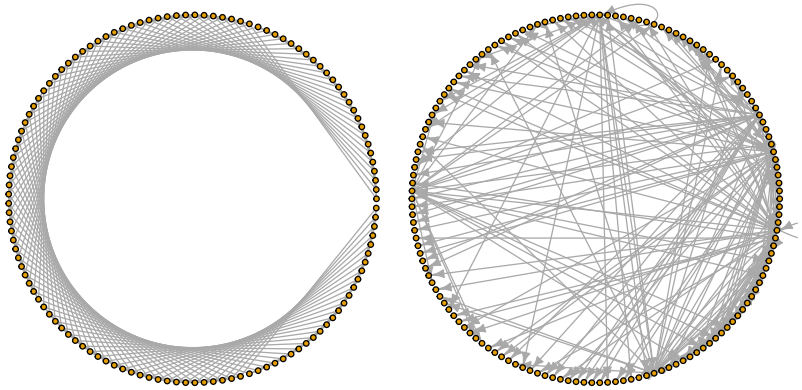
Random Layout

5x5x5 Lattice **Blog Network**



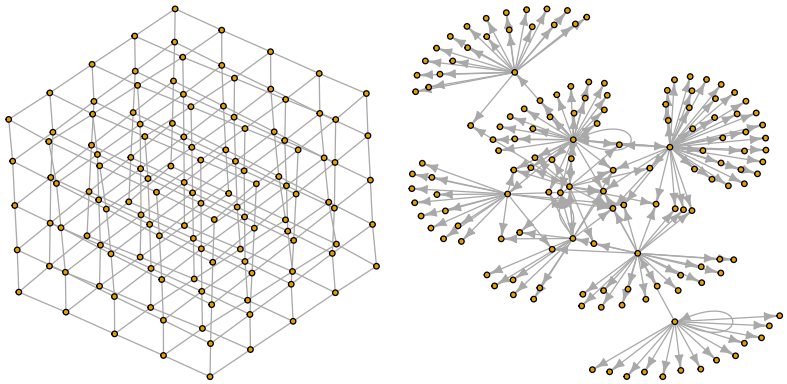
Circle Layout

5x5x5 Lattice **Blog Network**



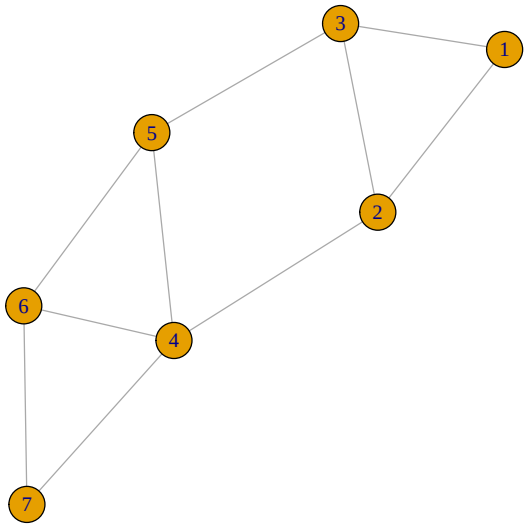
MDS Layout

5x5x5 Lattice **Blog Network**



3 Graphs and Matrix Algebra

We will be using our example graph



- Adjacency matrix

$$A_{ij} = \begin{cases} 1, & \text{if } \{i,j\} \in E, \\ 0, & \text{otherwise} \end{cases}$$

- The row sums give the vertex *degree*,

$$d_i = \sum_j A_{ij}$$

  which is the number of edges vertex i is connected to

```
A = get.adjacency(g)
A = as.matrix(A)      # get 'un-sparse' matrix
A
#>   1 2 3 4 5 6 7
#> 1 0 1 1 0 0 0 0
#> 2 1 0 1 1 0 0 0
#> 3 1 1 0 0 1 0 0
#> 4 0 1 0 0 1 1 1
#> 5 0 0 1 1 0 1 0
#> 6 0 0 0 1 1 0 1
#> 7 0 0 0 1 0 1 0
rowSums(A)            # degree from adjacency matrix
#> 1 2 3 4 5 6 7
#> 2 3 3 4 3 3 2
degree(g)             # using igraph::degree() function
#> 1 2 3 4 5 6 7
#> 2 3 3 4 3 3 2
```

- For directed graphs (digraphs), $d_i^{out} = \sum_j A_{ij}$ and $d_i^{in} = \sum_j A_{ji}$
  - rowsums or colsums
- The matrix power, A^r gives the number of walks of length r between vertices

```
#- Direct method
Ar = diag(nrow(A))
```

```
for(i in 1:2) {Ar <- Ar %*% A}   # r = 2
Ar
#>      1 2 3 4 5 6 7
#> [1,] 2 1 1 1 1 0 0
#> [2,] 1 3 1 0 2 1 1
#> [3,] 1 1 3 2 0 1 0
#> [4,] 1 0 2 4 1 2 1
#> [5,] 1 2 0 1 3 1 2
#> [6,] 0 1 1 2 1 3 1
#> [7,] 0 1 0 1 2 1 2

#- eigen method
r = 2
eig = eigen(A)
Ar2 = eig$vectors %*% diag(eig$values^r) %*% solve(eig$vectors)
round(Ar2)
#>      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
#> [1,]    2    1    1    1    1    0    0
#> [2,]    1    3    1    0    2    1    1
#> [3,]    1    1    3    2    0    1    0
#> [4,]    1    0    2    4    1    2    1
#> [5,]    1    2    0    1    3    1    2
#> [6,]    0    1    1    2    1    3    1
#> [7,]    0    1    0    1    2    1    2
```

- *Graph Laplacian* is the $|V| \times |V|$ matrix $L = D - A$, where $D = \text{diag}[d_i : i \in V]$ is the diagonal matrix with degree along the diagonal. It is useful for calculating:

$$\mathbf{x}^\mathsf{T} L \mathbf{x} = \sum_{\{i,j\} \in E} (x_i - x_j)^2$$

  for $\mathbf{x} \in \mathbb{R}^{|V|}$.

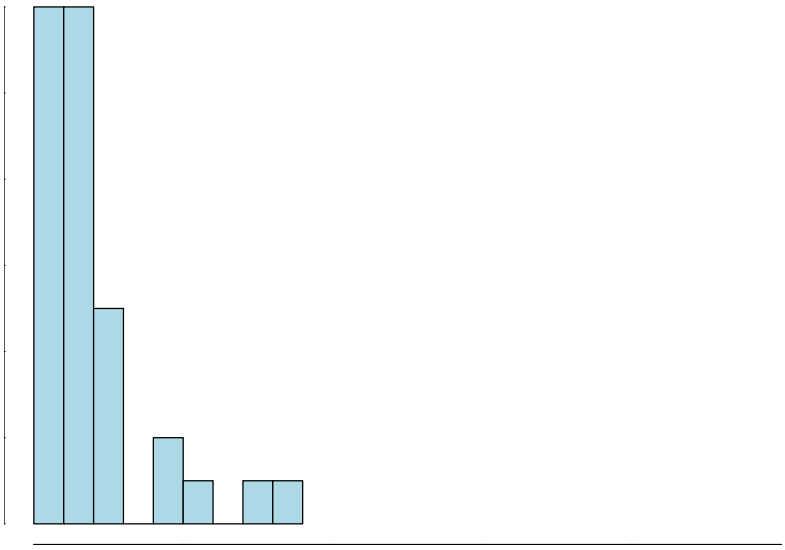
4 Descriptive Analysis of Network Graph Characteristics

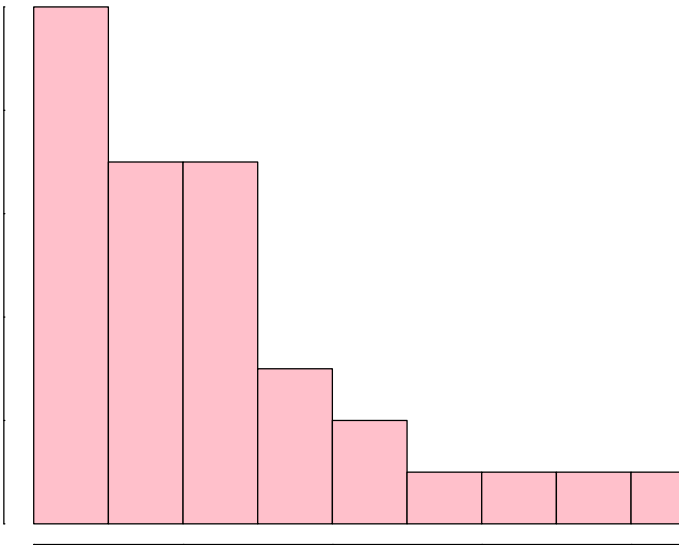
4.1 Vertex Degree Distribution

```
library(sand)
data(karate)

#- degree
hist(degree(karate), col="lightblue", xlim=c(0,50),
   xlab="Vertex Degree", ylab="Frequency", main="")

#- strength (i.e., weighted degree)
hist(graph.strength(karate), col="pink",
   xlab="Vertex Strength", ylab="Frequency", main="")
```





4.2 Density

The density of a graph (or subgraph) is the ratio of number of edges to the total possible number of edges.

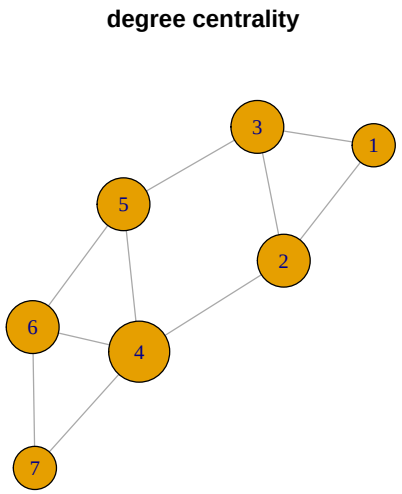
$$\text{den}(G) = \frac{|E|}{|V|(|V|-1)/2}$$

4.3 Vertex Centrality

Centrality tries to assess how "important" a vertex is.
- Which actors in a social network seem to hold the 'reins of power'?
- How authoritative does a webpage seem to be considered?
- How critical is a router in the internet network?

1. Degree centrality - (the number of edges a vertex has) is the most basic definition of importance

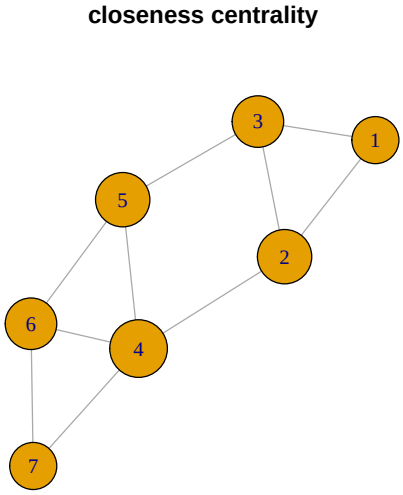
```
deg = degree(g)
cent.deg = deg/sum(deg)
plot(g,layout=g.layout,vertex.size=80*sqrt(cent.deg))
title("degree centrality")
```



2. Closeness centrality - measures the importance in terms of how 'close' a vertex is to the other vertices in the graph. The standard approach is to let the centrality vary inversely with a measure of the total distance of a vertex to all the others:

$$c(v) = \frac{1}{\sum_{u \in V} \text{dist}(v, u)}$$

```
close = closeness(g)
cent.close = close/sum(close)
plot(g,layout=g.layout,vertex.size=80*sqrt(cent.close))
title("closeness centrality")
```

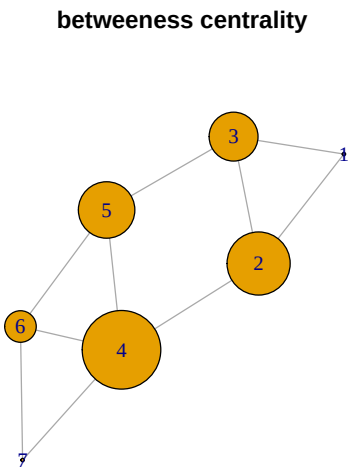


3. Betweenness centrality - measures how many paths cross through a vertex. An important vertex is one in which lots of information flows.

$$c(v) = \sum_{s \neq t \neq v \in V} \frac{\sigma(s, t|v)}{\sigma(s, t)}$$

where $\sigma(s, t|v)$ is the total number of *shortest paths* between s and t that pass through v, and $\sigma(s, t)$ is the total number of shortest paths between s and t (regardless of whether or note they pass through v).

```
between = betweenness(g)
cent.between = between/sum(between)
plot(g,layout=g.layout,vertex.size=80*sqrt(cent.between))
title("betweeness centrality")
```

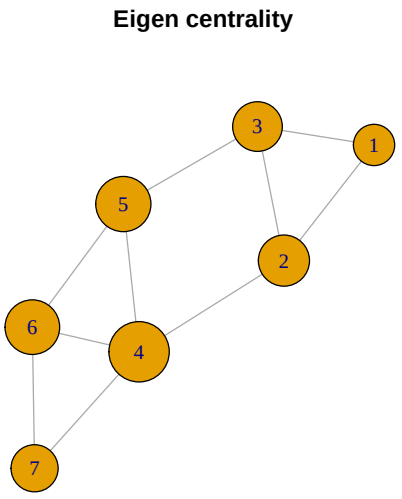


4. Eigenvector centrality - based on the notion that an important vertex will be connected to other importance vertices.

$$c(v) = \alpha \sum_{\{u,v\} \in E} c(u)$$

- Notice that the centrality for vertex v is the sum of the centrality of the vertices that it is connected to.
- This is the in form $A\mathbf{c} = \mathbf{c}$
- Or more generally, $A\mathbf{c} = \lambda\mathbf{c}$ (the eigen equations where $\mathbf{c}$ are the eigenvectors and λ the eigenvalues).
  - If A is not a stochastic matrix (rows sum to one, non-negative), then use the eigenvector corresponding to the largest magnitude eigenvalue
  - Standardize c to either have a maximum value of 1 or norm (sum of squares) of 1.

```
eigen = eigen_centrality(g)$vector  # first eigenvalue (max of 1)
cent.eigen = eigen/sum(eigen)
plot(g,layout=g.layout,vertex.size=80*sqrt(cent.eigen))
title("Eigen centrality")
```



- We can use the *power method* to solve $\mathbf{c} = A\mathbf{c}$

$$c^{\text{new}} = Ac^{\text{old}}/||Ac^{\text{old}}||$$

```
n = nrow(A)
y = matrix(1/n, n, 1) # initialize
for (i in 1:50){      # run until converges
  y = A%*%y
  y = y/sum(y)
}
data.frame(cent.eigen, y)
#>   cent.eigen          y
#> 1 0.09121398 0.09121398
#> 2 0.14012363 0.14012363
#> 3 0.13213865 0.13213865
#> 4 0.19489883 0.19489882
#> 5 0.16307969 0.16307969
#> 6 0.15973495 0.15973495
#> 7 0.11881028 0.11881028
```

5 PageRank

5.1 Random Surfer

The pagerank algorithm is based on the idea of a random (internet) surfer who randomly clicks on links from the current page.

- Consider transforming the adjacency matrix A into the appropriate transition matrix (markov chain)
  - Let $P_{ij} = A_{ij}/d_i^{out}$ be the row-standardized transition probability

$$P_{ij} = \begin{cases} \frac{1}{d_i^{out}}, & \text{if } \{i,j\} \in E, \\ 0, & \text{otherwise} \end{cases}$$

  - P_{ij} is the probability of a move from $i \to j$ if all edges are equally likely (random walk)
  - $P = D^{-1}A$ where $D = diag(d^{out})$

```
P = sweep(A, 1, rowSums(A), '/')
round(P, 2)
#>      1    2    3    4    5    6    7
#> 1 0.00 0.50 0.50 0.00 0.00 0.00 0.00
#> 2 0.33 0.00 0.33 0.33 0.00 0.00 0.00
#> 3 0.33 0.33 0.00 0.00 0.33 0.00 0.00
#> 4 0.00 0.25 0.00 0.00 0.25 0.25 0.25
#> 5 0.00 0.00 0.33 0.33 0.00 0.33 0.00
#> 6 0.00 0.00 0.00 0.33 0.33 0.00 0.33
#> 7 0.00 0.00 0.00 0.50 0.00 0.50 0.00
```

5.2 PageRank Details

Consider the directed graph representation of the www: $G = (V, E)$, where $n = |V|$ are the number of webpages

- Webpages link (hyperlink) to other webpages with directed edges
- An important webpage is one that many (important) pages link to it
- The PageRank score of page i is

$$\begin{aligned} r_i &= \sum_{\{j,i\} \in E} \frac{r_j}{d_j^{out}} \\ &= \sum_j P_{ji} r_j \end{aligned}$$

- This gives the system of equations

$$\mathbf{r} = P^\mathsf{T} \mathbf{r}$$

But we have a problem. What if some pages cannot be reached by other pages?

The approach taken in PageRank is to add a dampening factor. Or alternatively the idea that the websurfer will randomly click links, but occasionally will pick another webpage (from the full set of vertices) at random and starts again. This can be modeled by

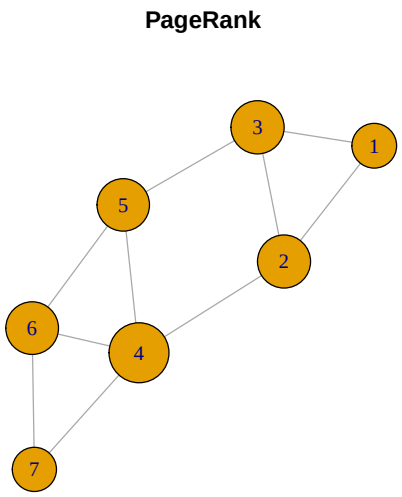
$$\mathbf{r} = \frac{1-d}{n}\mathbf{e} + dP^\mathsf{T}\mathbf{r}$$

- $0 \leq d \leq 1$ is the *dampening factor* or the probability the surfer keeps clicking on the links (and thus $1 - d$ the probability of random selection)
  - Original method used $d = 0.85$
- $\mathbf{e}$ is a column vector of ones

- Equivalently,

$$\begin{aligned} r_i &= \frac{1-d}{n} + d \sum_{\{j,i\} \in E} \frac{r_j}{d_j^{out}} \\ &= \frac{1-d}{n} + d \sum_{j=1}^{n} P_{ji} r_j \end{aligned}$$

```
pr = page_rank(g)$vector
cent.pr = pr/sum(pr)
plot(g,layout=g.layout,vertex.size=80*sqrt(cent.pr))
title("PageRank")
```



- The power iteration method can also be used to solve this equation, which finds the eigenvector with eigenvalue of 1. This is a very fast approach which can be parallel processed.

```
P = sweep(A, 1, rowSums(A), '/')
d = 0.85

y = matrix(1/n, n, 1) # initialize
for (i in 1:50){     # run until converges
  y =   (1-d)/n + d*crossprod(P,y)
  y = y/sum(y)         # should sum to 1, but roundoff error
}
data.frame(cent.pr, y)
#>     cent.pr         y
#> 1 0.1068964 0.1068964
#> 2 0.1504901 0.1504901
#> 3 0.1511611 0.1511611
#> 4 0.1920074 0.1920074
#> 5 0.1470447 0.1470447
#> 6 0.1481845 0.1481845
#> 7 0.1042158 0.1042158
```