Magolego SNA - Information Propagation

Independent Cascade model

In the Independent Cascade model, we have an initial set of active nodes. On every step an active node v can activate connected neighbor w with a probability $p_{v,w}$. Each node in the network has only one chance to activate its neighbor, and the process stops when there are no more activations possible.

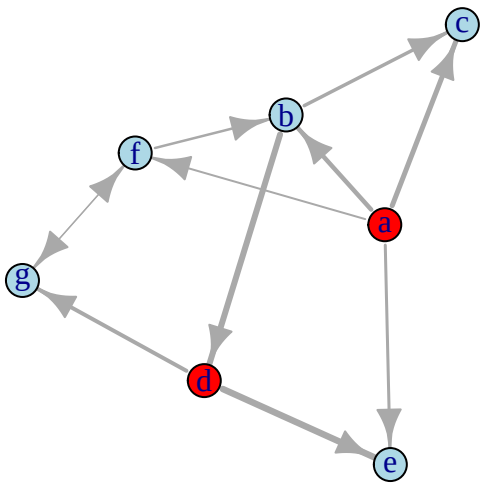
Since there is no package implementing this model in R, let is implement the simulation from scratch. The function below has a graph object representing the network as its first parameter, and a list of names of initially active nodes as its second parameter. The code visualizes the network at each step of the simulation, marking activated nodes with red color. Note also that edges with higher probability to activate the target node are drawn in a more thick way.

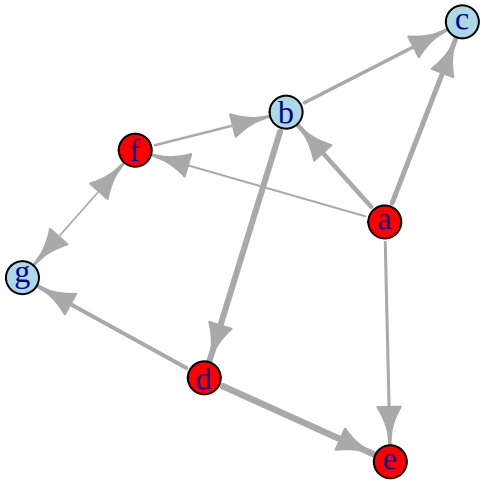
```
library('igraph')

# Independent Cascade model
#   - graph: igraph object
#   - activated: list of initially activated nodes
IC <- function (g, activated) {
  # Defining the graph layout to preserve it
  # the same throughout the visualization
  l <- layout.fruchterman.reingold(g)
  # Setting the activated nodes
  V(g)$activated <- F
  for (v in activated) {
    V(g)[v]$activated <- T
  }
  # Marking all activations (edges) as "not yet tried"
  E(g)$tried <- F
  possible.activations = ecount(g)
  # The process goes on until there are possible activations
  while(possible.activations > 0) {
    # Network visualization (at each simulation step)
    V(g)$color <- ifelse(V(g)$activated, "red", "lightblue")
    plot(g, layout=l, edge.width=E(g)$weight*5)
    # Iterating through activated nodes
    for(v in V(g)) {
      if(V(g)[v]$activated) {
        # Finding activations for each note that have not been tried yet
        for(w in neighbors(g, V(g)[v]$name, mode="out")) {
          e <- E(g)[from(V(g)[v]$name) & to(V(g)[w]$name)]
          if (! e$tried) {
            # Activation attempt
            if (runif(1, 0, 1) <= e$weight) {
              V(g)[w]$activated <- T
            }
            e$tried <- T
            possible.activations <- possible.activations - 1
          }
        }
      }
```

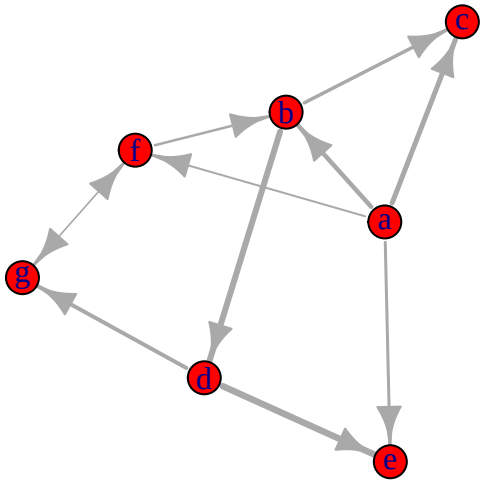
```
    }
  }
  # Network visualization after the process has terminated
  V(g)$color <- ifelse(V(g)$activated, "red", "lightblue")
  plot(g, layout=l, edge.width=E(g)$weight*5)
}
```

Let us try this simulation on a simple directed network, setting just two nodes as initially infected ones:

```
g <- read.table("graph.txt", header=T)
g <- graph.data.frame(g)
IC(g, c("a", "d"))
```

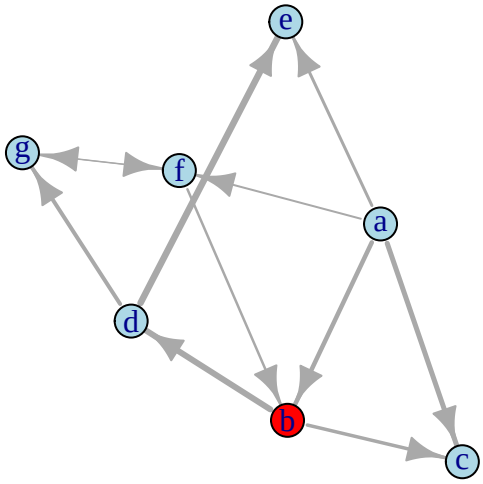


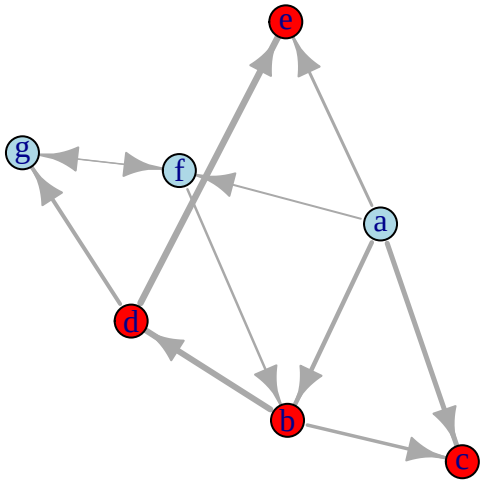

c
b
f
a
g
d
e

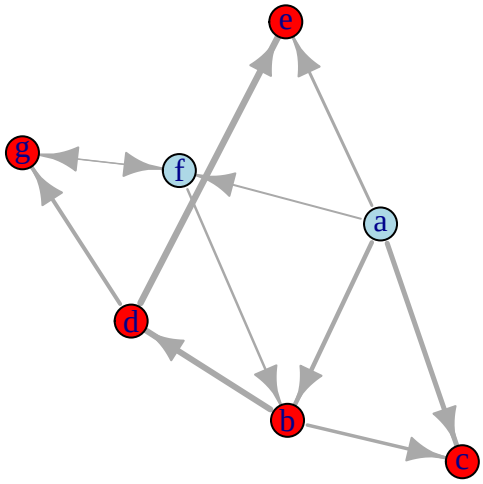


Let us see what happens if we set only one node activated before the process starts:

```
IC(g, c("b"))
```


e
g
f
a
d
b
c


e
g
f
a
d
b
c



Linear threshold model

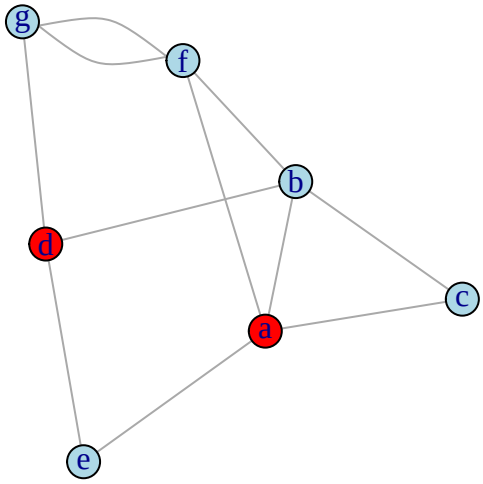
Let us now implement the Linear Threshold model. For simplicity, we will implement its unweighted version where the threshold of acceptance of the new behavior ("activation") depends only on the fraction on the nearest neighbors being activated.

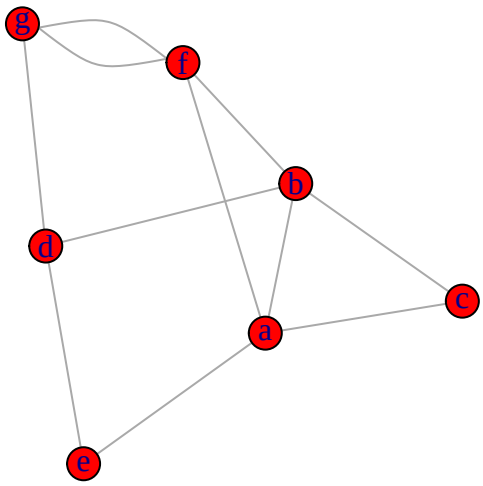
```
library('igraph')

# Linear Threshold model (unweighted version)
#   - graph: igraph object
#   - activated: list of initially activated nodes
#   - a: payoff for activation
#   - b: payoff for being not activated
LT <- function (g, activated, a, b) {
  # Defining the graph layout to preserve it
  # the same throughout the visualization
  l <- layout.fruchterman.reingold(g)
  # calculating the threshold
  threshold <- b * 1.0 / (a + b)
  # Setting the activated nodes
  V(g)$activated <- F
  for (v in activated) {
    V(g)[v]$activated <- T
  }
```

```
  # Indicator of whether simulation should stop
  any.changes <- T
  while(any.changes) {
    # Network visualization (at each simulation step)
    V(g)$color <- ifelse(V(g)$activated, "red", "lightblue")
    plot(g, layout=l)
    any.changes <- F
    # Iterating through non-activated nodes
    for(v in V(g)) {
      if(! V(g)[v]$activated) {
        # Calculating the fraction of activated neighbors
        neighborhood <- neighbors(g, V(g)[v]$name)
        activated.neighbors <- 0
        total.neighbors <- length(neighborhood)
        for(w in neighborhood) {
          if(V(g)[w]$activated) {
            activated.neighbors <- activated.neighbors + 1
          }
        }
        if (total.neighbors > 0 &&
            activated.neighbors * 1.0 / total.neighbors >= threshold) {
          V(g)[v]$activated <- T
          any.changes <- T
        }
      }
    }
  }
  # Network visualization after the process has terminated
  V(g)$color <- ifelse(V(g)$activated, "red", "lightblue")
  plot(g, layout=l)
}
```

Below we try this simulation for the same graph, but this time in its undirected version (we set $a = 3$, $b = 2$, so that the new behavior is "better"):

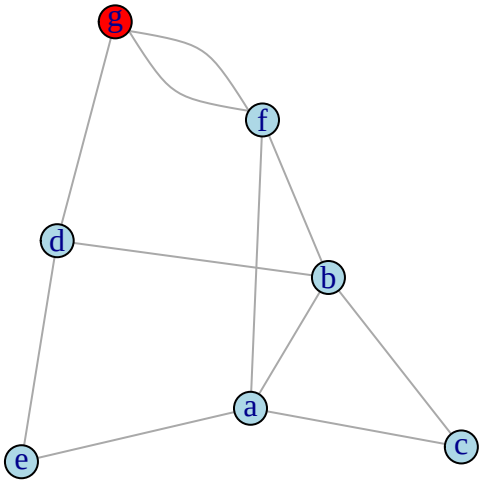
```
g <- read.table("graph.txt", header=T)
g <- graph.data.frame(g, directed=F)
LT(g, c("a", "d"), a=3, b=2)
```

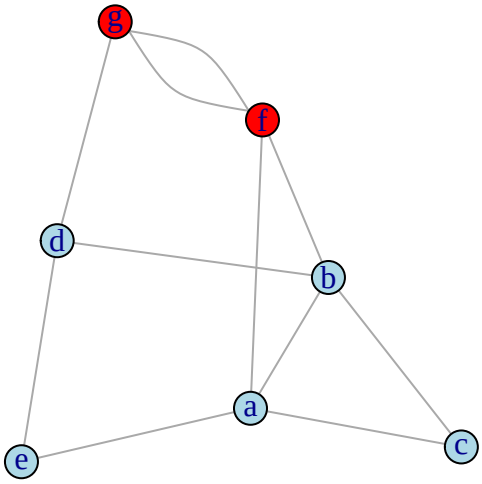

g
f
b
d
c
a
e



Other choices of initially activated nodes make the result completely different:

```
g <- read.table("graph.txt", header=T)
g <- graph.data.frame(g, directed=F)
LT(g, c("g"), a=3, b=2)
```


g
f
d
b
a
e
c


g
f
d
b
a
e
c