TP 3 : Igraph/R - Identication de communautés locales

Rushed Kanawati

6 décembre 2016

Résumé

Dans Ce TP nous étudions différentes approches d'identification de communautés locales dans des réseaux complexes..

Exercices

1 Télécharger sur le site `http://lipn.fr/~kanawati/ars` les graphes suivants `dolphins.gml`, `polblogs.gml`, `football.gml`, `karate.gml` et `polbooks.gml`. Pour ces graphes, les communautés détectés par les experts sont renseignées dans l'attribut `value`

```
g1<- read.graph("karate.gml", format="gml")
...
```

2 Développer une fonction `mod_R` qui calcule la modularité locale $R = \frac{B_{in}}{B_{in}+B_{out}}$

```
mod_R <- function(g,C,B,S){
  # R = Bin /(Bin+Bout)
  # C,B,S are three disjoint subsets of V(g), g is a graph
  bin <- length(E(g)[B %--% B])
  bout <- length(E(g)[B %--% S])
  return (bin/(bin+bout))
}
```

3 Développer une fonction `mod_M` qui calcule la modularité locale $M = \frac{D_{in}}{D_{out}}$

```
mod_M <- function(g,C,B,S){
  # D = B union C
  # M = Din / Dout
  D <- union(C,B)
  din <- length(E(g)[D %--% D ])
  dout <- length(E(g)[B %--% S])
  return(din/dout)
}
```

4 Développer une fonction `mod_L` qui calcule la modularité locale $L = \frac{L_{in}}{L_{out}}$

```
neighbors_in <- function(n,g,E){
  # returns the number of neighbors of node n in set E in graph g
  return(length(intersect(neighbors(g,n),E)))
}

mod_L <- function(g,C,B,S){
  D <- union(C,B)
  lin <- sum(sapply(D,neighbors_in,g,D))/length(D)
  lout <- sum(sapply(B,neighbors_in,g,S))/length(B)
  return(lin/lout)
}
```

5 Développer une fonction `conductance` qui calcule la modularité locale $f(S) = \frac{C_S}{2m_s+C_S}$

```
conductance <- function(g,C,B,S){
  D <- union(C,B)
  cs <- length(E(g)[B%--%S])
  ms <- length(E(g)[D%--%D])
  return(cs/(cs+2*ms))
}
```

6 Développer une fonction `local_com` qui calcule la communauté locale d'un nœud selon la stratégie d'optimisation gloutonne d'une des fonctions de modularité locales développées ci-avant.

```
update <- function(n,g,C,B,S){
  # move n in S to D
  S <- S[S!=n]
  D <- union(C,B)
  if (all(neighbors(g,n) %in% D)){
    # add n to C
    C <- union(c,n)
  }else{
    # add n to B
    B <- union(B,n)
    new_s=setdiff(neighbors(g,n),union(D,S))
    if (length(new_s) >0){
      S <- union(S,new_s)
    }
    for (b in B){
      if (all(neighbors(g,b) %in% D)){
        B<-B[B!=b]
        C<-union(C,b)
      }
    }
  }
  return(list(C=C,B=B,S=S))
}
```

```r

compute_quality<-function(n,g,C,B,S,mod){
  # computes the quality of a community if node n joins
  # n is a node in S
  res <- update(n,g,C,B,S)
  C<-res$C
  B<-res$B
  S<-res$S
  return (mod(g,C,B,S))
}


local_com <- function(target,g,mod){
  # initilizations
  if (is.igraph(g) && target %in% V(g)){
      C <- c()
      B <- c(target)
      S <- c(V(g)[neighbors(g,target)]$id)
      Q <- 0
      new_Q <- 0
      while ((length(S)>0) && (new_Q >= Q)){
        QS <- sapply(S,compute_quality,g,C,B,S,mod)
        new_Q <- max(QS)
        if (new_Q>=Q){
          s_node <- S[which.max(QS)]
          res <- update(s_node,g,C,B,S)
          C<-res$C
          B<-res$B
          S<-res$S
          Q <-new_Q
        }
      }
      return(union(C,B))
  }else{
    stop("invalid arguments")
  }
}
```

7 Ecrire une fonction qui calcule une partition d'un graphe en fonction de l'appartenance des nœuds a une communauté locale d'un nœud cible.

```r
compute_ego_partition <- function(target,g,mod){
  res$com <- local_com(target,g,mod)
  res$not_com <-V(g)[!(id %in% com)]$id
  return(res)
}
```

8 Ecrire une fonction qui calcule la qualité d'une communauté d'un nœud en fonction de la similarité avec une décomposition de vérité terrain.

```r
groundtruth_localcom_quality <- function(g,bipartition,method="nmi"){
  V(g)[id %in% bipartition$com]$egocom <-0
  V(g)[id %in% bipartition$not_com]$egocom <-1
```

```
  return(compare(V(g)$value,V(g)$egocom,method))
}

localcom_quality <-function(target,g,mod,method){
  bipartition <-compute_ego_partition(target,g,mod)
  return(groundtruth_localcom_quality(g,bipartition,method))
}
```