```
#Backbone extraction
#Not finished

backboneNetwork<-function(g,alpha,evalFunc){
        #Returns a backbone network based on LANS.
        #g is a weighted perhaps directed graph
        #alpha is the significance level for links
        #First, convert graph to adjancy matrix

        A<-get.adjacency(g,attr="weight")
        A<-as.matrix(A)
        #Now, convert this matrix to a probability matrix,p-matrix.
The function rowSums(A) returns a vector with the sum of allthe
entries in a row
        p<-A/rowSums(A)
        #Apparently, R interprets this division in the following
way: Divide each row, i, in A with the corresponding i'th entry in
the vector.
        #Nonparametric sparsification (Foti et.al, 2011 in PLOS
ONE)

        #This is the evaluation function. It takes a vector of
probabilities, Q, and compares each entry with the other entries in
the vector.
        #It returns the number of entries that are less than or
equal to the i'th entry.
        F_hat<-function(Q){
          x<-vector()
          for(j in 1:length(Q)){
            x[j]<-length(which(Q!=0 & Q<=Q[j]))/length(which(Q>0))
          }
          return(x)
        }
        #The following produces a matrix, sigMatrix, with values 1
for the links that are to be kept and 0 for the links that we throw
away.
        sigMatrix<-matrix(nrow = length(V(g)), ncol=length(V(g)))
        for(i in 1:length(V(g))){
          sigMatrix[i,]<-F_hat(p[i,])
        }
        sigMatrix2<-sigMatrix >= 1 - alpha

        mode(sigMatrix2)<-"numeric"
        sigMatrix2[is.na(sigMatrix2)] <- 0
        #Now multiply the original adjacency matrix with sigMatrix
to get rid of the insignificant links
        B<-sigMatrix2*A


        if(evalFunc==1){
          #directed
          h<-graph.adjacency(B,mode=c("directed"),weighted=TRUE)
          V(h)$id<-V(g)$id
```

```
        }
                else{

                        #soft
                        h<-
graph.adjacency(B,mode=c("max"),weighted=TRUE)
                        V(h)$id<-V(g)$id
                        }


#h<-as.undirected(h, mode = c("collapse"),edge.attr.comb = "min")


return(h)
}


#Alpha<-function(Q){
#  x<-vector()
#  for(j in 1:length(Q)){
#    x[j]<-(1-Q[j])**(length(which(Q>0))-1)
#  }
#  return(x)
#}

#sigMatrix<-matrix(nrow = length(V(g)), ncol=length(V(g)))
#for(i in 1:length(V(g))){
#  sigMatrix[i,]<-Alpha(p[i,])

#}
#sigMatrix2<-sigMatrix < alpha
#mode(sigMatrix2)<-"numeric"
#sigMatrix2[is.na(sigMatrix2)] <- 0
#Now multiply the original adjacency matrix with sigMatrix to get
rid of the insignificant links
#B<-sigMatrix2*A
#Now create a graph from the new matrix.
#h<-graph.adjacency(B,mode=c("lower"),weighted=TRUE)
#V(h)$id<-V(g)$id
```