```
## FESTA algorithm
# Load required packages and utility functions
require(amap)
require(plyr)

##### Splicing detection
##### Cluster genes according to reciprocal correlations, then iteratively cut tree (bottom
up) until one cluster is the most expressed or tied for expression across all samples. Uses
hcluster from amap package for clustering

## Required input data:
## data.frame with one row per exon and one column per sample, plus
## "geneID" column with unique gene identifier
## "exonID" column with unique exon identifier

# Example data
# geneID <- paste("gene",100:500, sep = "")
# exonID <- paste(merge(geneID,c(1:10))[,1], merge(geneID,c(1:10))[,2],sep = "exon")
# exprData <- matrix(log2(rbinom(n = 4010*10, size = 1000, prob = .3)), ncol = 10)
# exampleData <- data.frame(geneID = geneID,
#                           exonID = exonID,
#                           Evalue = exprData)

## Parameter description
## exceptions:
## signDigits: number of digits rounded from expression scores for ranking calculations
## distMethod: distance metric used by the clustering algorithm (default: correlation), see
function hcluster from package amap for more information
## link: agglomeration method used by the clustering algorithm (default: complete), see
function hcluster from package amap for more information
## nbproc: number of subprocess for parallelization (default: 1), see function hcluster from
package amap for more information

ClusterExons <- function(data = NULL, exceptions = ceiling(x = (ncol(data)-2)*.1),
signDigits = 3, distMethod = "correlation", link = "complete", nbproc = 1) {
  require(amap)
  require(plyr)
  ExonAssTable <- list()
  exceptions <- exceptions/(ncol(data)-2)
  row.names(data) <- data$exonID
  for (gID in unique(data$geneID)){
    Evalues<-data[which(data$geneID%in%gID),-grep(pattern = "ID",x = names(data))]
    if (nrow(Evalues)<2) {
      ExonAssTable[[gID]]<-as.data.frame(matrix(row.names(Evalues), ncol = 1))
      names(ExonAssTable[[gID]])<-"exonID"
      ExonAssTable[[gID]]$splicing_category<-"single_expressed_exon"
      ExonAssTable[[gID]]$clusters<-0
      ExonAssTable[[gID]]$clusterranks<-1
      ExonAssTable[[gID]]<-ExonAssTable[[gID]][c("clusters", "exonID", "clusterranks",
      "splicing_category")]
    } else {
      tree<-hcluster(Evalues, method = distMethod, link = link, nbproc = nbproc)
      for (trans in nrow(Evalues):1){
        # evaluate relative times it is ranked as first
        Evalues$clusters<-cutree(tree, k = trans)
        clusterranks<-ddply(Evalues, .(clusters), colwise(median))
        if (nrow(clusterranks)==1){ # there are no subranking isoforms
          ExonAssTable[[gID]]<-as.data.frame(matrix(row.names(Evalues), ncol = 1))
          names(ExonAssTable[[gID]])<-"exonID"
          ExonAssTable[[gID]]$clusters<-0
          ExonAssTable[[gID]]$clusterranks<-1
          ExonAssTable[[gID]]$splicing_category<-"unspliced"
          ExonAssTable[[gID]]<-ExonAssTable[[gID]][c("clusters", "exonID", "clusterranks",
          "splicing_category")]
          break} else {
            clusterranks<-apply(clusterranks[,-1], 2, function(x){rank(-round(x, digits =
            signDigits), ties.method = "min", na.last = T)})
            row.names(clusterranks)<-unique(Evalues$cluster)
            clusterranks<-apply(clusterranks, 1, function(x){sum(x==1)/ncol(clusterranks)})
            if (sum((clusterranks)>=(1-exceptions))==1) { # control that there is only one
            exon group which consistently ranks 1st or tied allowing for exceptions
```

```
            matchmaker<-Evalues[,"clusters", drop=F] # create table with exon subcluster
            assignments
            clusters<-as.data.frame(clusterranks) # store subcluster ranks
            clusters$clusters<-c(1:nrow(clusters)) #annotate subcluster ranks with
            subcluster IDs
            matchmaker<-join(matchmaker, clusters, by="clusters", type="left") # merge
            exon with subcluster ID and ranks
            row.names(matchmaker)<-row.names(Evalues) # add exon names
            matchmaker$exonID<-row.names(matchmaker)
            ExonAssTable[[gID]]<-matchmaker
            ExonAssTable[[gID]]$splicing_category<-"spliced"
            ExonAssTable[[gID]]<-ExonAssTable[[gID]][c("clusters", "exonID",
            "clusterranks", "splicing_category")]
            break} else {next}}}
    }
  }

  # this step causes problem in R below version 2
  ExonAssTable<-ldply(ExonAssTable,rbind)
  names(ExonAssTable)[1]<-"geneID"
  # assign constitutive/specific status

  ExonAssTable$constitutive<-ifelse((ExonAssTable$clusterranks>=(1-exceptions)),"constitutive"
  ,"facultative")
  # merge cluster assignments into unique IDs with designation of constitutiveness
  ExonAssTable$transcriptID<-as.factor(paste0(ExonAssTable$geneID, "_t",
  ExonAssTable$clusters, ifelse(ExonAssTable$constitutive=="constitutive", "_con","_fac"),
  sep=""))
  # code ID variables as factors
  ExonAssTable$geneID <- as.factor(ExonAssTable$geneID)
  ExonAssTable$splicing_category <- as.factor(ExonAssTable$splicing_category)
  ExonAssTable$constitutive <- as.factor(ExonAssTable$constitutive)
  ExonAssTable
}



##### Average expression values based on unique eigenexon IDs

## Required input data:
## data.frame with one row per exon and one column per sample, plus
## "geneID" column with unique gene identifier
## "transcriptID" column with unique exon identifier as per assigned via ClusterExons
## "constitutive" column identifying which transcripts are from constitutive nodes as per
assigned via ClusterExons

## Parameter description:
## splicingRatios: Logical. If FALSE, expression from all entries is reported on the same
scale. If TRUE, expression from splicing entries is normalized by their gene's constitutive
expression score, generating splicing ratios
## NAcorrection: Logical. Applicable only if splicingRatios is TRUE. If TRUE, splicing
ratios higher than 1 are set to 1 and NA/NaN/infinity values to 0. This accounts for
experimental error in measurements.

AverageExons <- function(data = NULL, splicingRatios = F, NAcorrection = F){
  if (splicingRatios == F) {
    out <-ddply(.data = data, .variables = .(transcriptID), numcolwise(median), na.rm = T)
    out[order(out$transcriptID),]
  } else {
    ## split into constitutives and facultatives
    ConTranscripts <- data[which(data$constitutive=="constitutive"),]
    FacTranscripts <- data[which(data$constitutive!="constitutive"),]
    ## average exon values within transcripts
    ConTranscripts <- ddply(ConTranscripts, .variables = .(geneID, transcriptID),
    numcolwise(median), na.rm = T)
    FacTranscripts <- ddply(FacTranscripts, .variables = .(geneID, transcriptID),
    numcolwise(median), na.rm = T)
    FacSplicing <- apply(FacTranscripts, 1, function(Fac){
      Spl <- as.numeric(Fac[-grep(pattern = "ID", x = names(FacTranscripts))])
      Con <- ConTranscripts[which(Fac["geneID"]==ConTranscripts$geneID),-grep(pattern =
      "ID", x = names(ConTranscripts))]
```

```
      Spl/Con
    })
    FacSplicing <- cbind(FacTranscripts[,c("geneID","transcriptID")],ldply(FacSplicing))
    if (NAcorr == T) {
      # set NA/NaN and infinity scores to 0, set scores greater than 1 to 1
      FacSplicing[,sapply(FacSplicing, is.numeric)]<-apply(FacSplicing[,sapply(FacSplicing,
      is.numeric)],c(1,2),function(x){as.numeric(ifelse(is.na(x), 0, x))})
      FacSplicing[,sapply(FacSplicing, is.numeric)]<-apply(FacSplicing[,sapply(FacSplicing,
      is.numeric)],c(1,2),function(x){as.numeric(ifelse(x>1, 1, x))})
    }
    out <- rbind(ConTranscripts,FacSplicing)
    out[order(out$transcriptID),]
  }
}
```