Package 'ADaCGH2'

April 14, 2017

Version 2.14.0

Date 2016-04-29

Title Analysis of big data from aCGH experiments using parallel computing and ff objects

Author Ramon Diaz-Uriarte <rdiaz02@gmail.com> and Oscar M. Rueda <rueda.om@gmail.com>. Wavelet-based aCGH smoothing code from Li Hsu <lih@fhcrc.org> and Douglas Grove <dgrove@fhcrc.org>. Imagemap code from Barry Rowlingson <B.Rowlingson@lancaster.ac.uk>. HaarSeg code from Erez Ben-Yaacov; downloaded from <http://www.ee.technion.ac.il/people/YoninaEldar/Info/software/HaarSeg.htm>.

Maintainer Ramon Diaz-Uriarte <rdiaz02@gmail.com>

Depends R (>= 3.2.0), parallel, ff, GLAD

Imports bit, ffbase, DNAcopy, tilingArray, waveslim, cluster, aCGH, snapCGH

Suggests CGHregions, Cairo, limma

Enhances Rmpi

Description Analysis and plotting of array CGH data. Allows usage of Circular Binary Segementation, wavelet-based smoothing (both as in Liu et al., and HaarSeg as in Ben-Yaacov and Eldar), HMM, BioHMM, GLAD, CGHseg. Most computations are parallelized (either via forking or with clusters, including MPI and sockets clusters) and use ff for storing data.

biocViews Microarray, CopyNumberVariants

LazyLoad Yes

License GPL (>= 3)

URL https://github.com/rdiaz02/adacgh2

NeedsCompilation yes

R topics documented:

`cutFile` *Cut a file and return individual columns*

Description

A text file is split by individual columns, creating as many files as individual columns. Splitting is used using multiple cores, if available. Specified columns are renamed as ID.txt, Chrom.txt, Pos.txt, and other columns can be deleted. The individual files can then all be read from a given directory with `inputToADaCGH`.

Usage

```
cutFile(filename,
        id.col,
        chrom.col,
        pos.col,
        sep = "\t",
        cols = NULL,
        mc.cores = detectCores(),
        delete.columns = NULL,
        fork = FALSE)
```

Arguments

`filename`	Name of the text file with the data. The file contains at least a column for probe ID, a column for Chromosome, and a column for position, and one or more for data. The column is expected to have a first row with an identifier. All columns are to be separated by a single separating ccharacter `colsep`.
`id.col`	The number of the column (starting from 1) that should be used as ID. This is the name before any columns are possibly deleted (see argument `delete.columns`).
`chrom.col`	The number of the column (starting from 1) that should be used as the identifier for Chromosome. This is the name before any columns are possibly deleted (see argument `delete.columns`).
`pos.col`	The number of the column (starting from 1) that should be used as the position (the coordinates). This is the name before any columns are possibly deleted (see argument `delete.columns`).
`cols`	The number of columns of the file. If not specified, we try to guess it. But guessing can be dangerous.
`sep`	The field or column separator, similar to `sep` in `read.table`, etc. The default is a tab. A space can be specified as 'sep = " "'. If you use the default in, say, `read.table`, which is 'sep = ""', then multiple consecutive tab or space field separators are taken as one —this is also the behavior in `read.table` or awk, which is what we use.
`mc.cores`	The number of processes to launch simultaneously.
`delete.columns`	The number of the columns (starting from 1) that you do not want to preserve. You probably do not want to have too many of these, and if you do you should note that the file is cut into pieces BEFORE dealing with the unwanted columns so many unwanted columns will mean that we are doing many unwanted calls to cut.

fork — Should we fork R processes, via mclapply, or just send several system commands from this R process. The default (FALSE) is probably the most reasonable option for large files.

Details

This function is unlikely to work under Windows unless MinGW or similar are installed (and even then it might not work). This function should work under Mac OS, and it does in the machines we've tried it, but it seems not to work on the BioC testing machine.

This function basically calls "head" and "awk" using `system`, and trying to divide all the jobs into as many cores as you specify (argument `cores`).

Value

This function is used for it main effect: cutting a file into individual one-column files. These files are names "col_1.txt", "col_2.txt", etc, and there are three called "ID.txt", "Chrom.txt", "Pos.txt". The files are created in the current working directory.

As we move the files corresponding to "ID", "Chrom", and "Position", the stdout output is shown to the user (to check that things worked).

After calling this function, you can call `inputToADaCGH`.

Author(s)

Ramon Diaz-Uriarte <rdiaz02@gmail.com>

Examples

```
## Read a tab separated file, and assign the first,
## second, and third positions to ID, Chrom, and Position

if( (.Platform$OS.type == "unix") && (Sys.info()['sysname'] != "Darwin") ) {
## This will not work in Windows, and might, or might not, work under Mac

fnametxt <- list.files(path = system.file("data", package = "ADaCGH2"),
                       full.names = TRUE, pattern = "inputEx.txt")
cutFile(fnametxt, 1, 2, 3, sep = "\t")

## Verify we have ID, Chrom, Pos
c("ID.txt", "Chrom.txt", "Pos.txt") %in% list.files(getwd())

## verify some other column

c("col_5.txt") %in% list.files(getwd())

## Read a white space separated file, and assign the first, second, and
## third positions to ID, Chrom, and Position, but remove the fifth
## column

fnametxt2 <- list.files(path = system.file("data", package = "ADaCGH2"),
                        full.names = TRUE, pattern = "inputEx-sp.txt")
cutFile(fnametxt2, 1, 2, 3, sep = " ", delete.columns = 5)

}
```

inputEx *A fictitious aCGH data set*

Description

A fictitious aCGH data set.

Usage

```
inputEx
```

Format

A data frame with 500 rows and 6 columns; the last three correspond to the aCGH data for three samples. There are data for five chromosomes. The same file is available in three formats: an RData file, a tab separated text file, and a space-separate text file.

Source

Simulated data

inputToADaCGH *Convert CGH data to ff or RAM objects for use with ADaCGH2*

Description

Input data with CGH data are converted to several ff files and data checked for potential errors and location duplications.

Usage

```
inputToADaCGH(ff.or.RAM = "RAM",
                      robjnames = c("cgh.dat", "chrom.dat",
                                    "pos.dat", "probenames.dat"),
                      ffpattern = paste(getwd(), "/", sep = ""),
                      MAList = NULL,
                      cloneinfo = NULL,
                      RDatafilename = NULL,
                      textfilename = NULL,
                      dataframe = NULL,
                      path = NULL,
                      excludefiles = NULL,
                      cloneinfosep = "\t",
                      cloneinfoquote = "\"",
                      minNumPerChrom = 10,
                      verbose = FALSE,
                      mc.cores = floor(detectCores()/2))
```

Arguments

ff.or.RAM	Whether you want to store the output as ff or RAM objects ("usual" R objects, such as data frames and vectors).
robjnames	Name of the objects that will be created in you use ff.or.RAM = "RAM". If the names existing in the environment from where the function is called, they will not be overwritten, and the function will abort.
ffpattern	See argument pattern in ff. The default is to create the "ff" files in the current working directory.
MAList	The name of an object of class MAList (as.MAList) or SegList (e.g., dim.SegList). See vignnettes for these packages for details about these objects. You have to specify one, and only one, of MAList, RDatafilename, textfilename, path.
cloneinfo	A character vector with the full path to a file that conforms to the characteristics of file in function read.clonesinfo (see details in the vignette) or the name of a data frame with at least a column named "Chr" (with chromosomal informtaion) and "Position". This is only needed if you use MAList and your MAList object does not have Position and Chr columns.
RDatafilename	Name of data RData file that contains the data frame with original, non-ff, data. Note: this is the name of the RData file (possibly including path), NOT the name of the data frame. (For that, look at dataframe). The first three columns of the data frame are the IDs of the probes, the chromosome number, and the position, and all remaining columns contain the data for the arrays, one column per array. The names of the first three column do not matter, but the order does. Names of the remaining columns will be used if existing; otherwise, fake array names will be created. You have to specify one, and only one, of MAList, RDatafilename, textfilename, path.
textfilename	The name of a text file with the data. It should be a tab separated file, with a header. The first three columns of the data frame are the IDs of the probes, the chromosome number, and the position, and all remaining columns contain the data for the arrays, one column per array. The names of the first three column do not matter, but the order does. Names of the remaining columns will be used if existing; otherwise, fake array names will be created. You have to specify one, and only one, of MAList, RDatafilename, textfilename, path.
dataframe	The name of a data frame with the data. The first three columns of the data frame are the IDs of the probes, the chromosome number, and the position, and all remaining columns contain the data for the arrays, one column per array. The names of the first three column do not matter, but the order does. Names of the remaining columns will be used if existing; otherwise, fake array names will be created.
path	The name of the directory (the full path) to where each of the individual, one-column, files are. We will read ALL of the files in this directory, except for those listed under excludefiles. One file has to be named "ID.txt", another "Chrom.txt", and a third "Pos.txt". The rest of the files can be named any way you want and those are the files that contain the CGH data. All of the files are expected to be one-column text files, with a first row with a header. The header will not be used for "ID.txt", "Chrom.txt", or "Pos.txt", but the header will be used as the name of the array/subject for the CGH data files.

	You have to specify one, and only one, of `MAList`, `RDatafilename`, `textfilename`, `path`.
`excludefiles`	If you have specified `path`, names of files not to be read. A vector of strings. These should be the names of the files, without path information (as all of the files are in the same directory, as specified by `path`).
`cloneinfosep`	Argument to `read.table` if reading a `cloneinfo` file. Note: this is NOT used if reading a text file given in `textfilename`.
`cloneinfoquote`	Argument to `read.table` if reading a `cloneinfo` file. Note: this is NOT used if reading a text file given in `textfilename`.
`minNumPerChrom`	If any chromosome has fewer observations than minNumPerChrom the function will fail. This can help detect upstream pre-processing errors.
`verbose`	If TRUE, provide additional information that can be useful to debug problems. Right now it provides the list of files that will be read if using a directory. The default is FALSE.
`mc.cores`	The number of cores to use when reading files. This is always 1 in Windows. See details about the number of cores in `mclapply` and `detectCores`. Contention problems in I/O might be minimized by making this number smaller or much smaller than what is returned by `detectCores`. For larte jobs, please do some initial benchmarking. See comments and discussion in file "benchmark.pdf". In general, if you have a single SATA disk make this a small number (say, 2 or 3 or 6); in contrast, if you have many fast SAS disks in a RAID0 or RAID10 array, you can increase the number quite a bit (but generally always well below what detectCores gives).

Details

If there are identical positions (in the same chromosome) a small random uniform variate is added to get unique locations.

We carry out several checks (e.g., no duplicated positions), but note that we DO NOT check for extremely large or small values, and this includes NOT CHECKING for infinite values.

Missing values are allowed in the data columns. However, we do not check for missing values in the ID, chromosome, or position columns, except if you are using as input an RData file or MA list. You better not have any missing values there; otherwise, things will break in strange ways. Why this inconsistency? Checking for missing values can consume a lot of resources (CPU and memory). If your are really huge, they will probably be stored as text files, and you are expected to use the appropriate tools there to filter (e.g., sed, awk, whatever). If they exist as an MA list or an RData file, they once fitted in RAM, so checking for these NAs is probably reasonable.

If you provide a text file as input (`textfilename`), the reading operation is carried out using `read.table.ffdf`, to allow for reading very large files. Using this option, however, does not force you to produce as output `ff` objects.

Commented examples of reading objects from **limma** and **snapCGH** are provided in the vignnette.

Value

This function is used mainly for its side effects: writing either several `ff` files to the current working directory, or several RAM objects (the usual, in memory, local, R objects). The actual names are printed out.

Author(s)

Ramon Diaz-Uriarte <rdiaz02@gmail.com>

See Also

cutFile for obtaining files in the format needed if you read from a directory.

Examples

```
## Create a temp dir for storing output.
## (Not needed, but cleaner).
dir.create("ADaCGH2_example_input_dir")
originalDir <- getwd()
setwd("ADaCGH2_example_input_dir")
## Sys.sleep(1)

## Get location (and full filename) of example data file
fnameRData <- list.files(path = system.file("data", package = "ADaCGH2"),
                         full.names = TRUE, pattern = "inputEx.RData")

fnametxt <- list.files(path = system.file("data", package = "ADaCGH2"),
                         full.names = TRUE, pattern = "inputEx.txt")

namepath <- system.file("example-datadir", package = "ADaCGH2")

## Read from RData and write to ff
inputToADaCGH(ff.or.RAM = "ff",
                      RDatafilename = fnameRData)

## Read from text file and write to ff
inputToADaCGH(ff.or.RAM = "ff",
                      textfilename = fnametxt)


## Read from text file and write to RAM
inputToADaCGH(ff.or.RAM = "RAM",
                      textfilename = fnametxt)

## Read from a directory and write to ff
inputToADaCGH(ff.or.RAM = "ff",
                      path = namepath)

### Clean up (DO NOT do this with objects you want to keep!!!)
load("chromData.RData")
load("posData.RData")
load("cghData.RData")

delete(cghData); rm(cghData)
delete(posData); rm(posData)
delete(chromData); rm(chromData)
unlink("chromData.RData")
unlink("posData.RData")
unlink("cghData.RData")
unlink("probeNames.RData")


### Running in a separate process. Only makes sense
### if returning ff objects (ff.or.RAM = "ff")
```

```
### This example will not work on Windows

## Not run:
mcparallel(inputToADaCGH(ff.or.RAM = "ff",
                                 RDatafilename = fnameRData),
                                 silent = FALSE)
  tableChromArray <- mccollect()
  if(inherits(tableChromArray, "try-error")) {
    stop("ERROR in input data conversion")
  }
### Clean up (DO NOT do this with objects you want to keep!!!)
load("chromData.RData")
load("posData.RData")
load("cghData.RData")

delete(cghData); rm(cghData)
delete(posData); rm(posData)
delete(chromData); rm(chromData)
unlink("chromData.RData")
unlink("posData.RData")
unlink("cghData.RData")
unlink("probeNames.RData")

## End(Not run)

### Try to prevent problems in R CMD check
## Sys.sleep(2)

### Delete temp dir
setwd(originalDir)
## Sys.sleep(2)
unlink("ADaCGH2_example_input_dir", recursive = TRUE)
## Sys.sleep(2)
```

`outputToCGHregions` *ADaCGH2 output as input to CGHregions*

Description

Convert ADaCGH2 output to a data frame that can be used as input for `CGHregions`. This function takes as input the two possible types of input produced by the `pSegment` functions: either an ff object (and its associated directory) or the names of the RAM objects (the usual, in memory R objects) with the output, and chromosome, position, and probe name information.

Usage

```
outputToCGHregions(ffoutput = NULL, directory = getwd(),
                                 output.dat = NULL,
                                 chrom.dat = NULL,
                                 pos.dat = NULL,
                                 probenames.dat = NULL)
```

Arguments

ffoutput	The name of the ff object with the output from a call to a pSegment function. You must provide either this argument or all of the arguments output.dat, chrom.dat, pos.dat and probenames.dat.
directory	The directory where the initial data transformation and the analysis have been carried out. It is a lot better if you just work on a single directory for a set of files. Otherwise, unless you keep very carefull track of where you do what, you will run into trouble. This is only relevant if you use an ff object (i.e., if ffoutput is not NULL.)
output.dat	The name of the RAM object with the output from a call to a pSegment function. You must provide this argument (as well as chrom.dat, pos.dat and probenames.dat) OR the name of an ffobject to argument ffoutput.
chrom.dat	The name of the RAM object with the chromosome data information. See the help for inputToADaCGH.
pos.dat	The name of the RAM object with the position data information. See the help for inputToADaCGH.
probenames.dat	The name of the RAM object with the probe names. See the help for inputToADaCGH.

Value

A data frame of 4 + k columns that can be used as input to the CGHregions function. The first four columns are the probe name, the chromosome, the position and the position. The last k columns are the calls for the k samples.

Note

This function does NOT check if the calls are meaningfull. In particular, you probably do NOT want to use this function when pSegment has been called using 'merging = "none"'.

Moreover, we do not check if there are missing values, and CGHregions is likely to fail when there are NAs. Finally, we do not try to use ff objects, so using this function with very large objects will probably fail.

Author(s)

Ramon Diaz-Uriarte <rdiaz02@gmail.com>

See Also

pSegment

Examples

```
## Get location (and full filename) of example data file
## We will read from a text file

fnametxt <- list.files(path = system.file("data", package = "ADaCGH2"),
                       full.names = TRUE, pattern = "inputEx.txt")
```

```
##################################
#####
##### Using RAM objects
#####
##################################

## Read data into RAM objects



inputToADaCGH(ff.or.RAM = "RAM",
                      textfilename = fnametxt)

## Run segmentation (e.g., HaarSeg)

haar.RAM.fork <- pSegmentHaarSeg(cgh.dat, chrom.dat,
                                  merging = "MAD")

forcghr <- outputToCGHregions(output.dat = haar.RAM.fork,
                              chrom.dat = chrom.dat,
                              pos.dat = pos.dat,
                              probenames.dat = probenames.dat)

## Run CGHregions
if(require(CGHregions)) {
  regions1 <- CGHregions(na.omit(forcghr))
  regions1
}

##################################
#####
##### Using ff objects
#####
##################################

if(.Platform$OS.type != "windows") {

## We do not want this to run in Windows the automated tests since
## issues with I/O. It should work, though, in interactive usage


## Create a temp dir for storing output.
## (Not needed, but cleaner).
dir.create("ADaCGH2_cghreg_example_tmp_dir")
originalDir <- getwd()
setwd("ADaCGH2_cghreg_example_tmp_dir")
## Sys.sleep(1)


inputToADaCGH(ff.or.RAM = "ff",
                      textfilename = fnametxt)



haar.ff.fork <- pSegmentHaarSeg("cghData.RData",
                                "chromData.RData",
```

```
                                merging = "MAD")

forcghr.ff <- outputToCGHregions(ffoutput = haar.ff.fork)

if(require(CGHregions)) {
  regions1 <- CGHregions(na.omit(forcghr.ff))
  regions1
}

### Clean up (DO NOT do this with objects you want to keep!!!)
load("chromData.RData")
load("posData.RData")
load("cghData.RData")

delete(cghData); rm(cghData)
delete(posData); rm(posData)
delete(chromData); rm(chromData)
unlink("chromData.RData")
unlink("posData.RData")
unlink("cghData.RData")
unlink("probeNames.RData")

lapply(haar.ff.fork, delete)
rm(haar.ff.fork)

### Delete all files and temp dir
setwd(originalDir)
## Sys.sleep(2)
unlink("ADaCGH2_cghreg_example_tmp_dir", recursive = TRUE)
## Sys.sleep(2)
}
```

---

pChromPlot *Segment plots for aCGH as PNG*

Description

Produce PNG figures of segment plots (by chromosome) for aCGH segmentation results. Internal calls are parallelized for increased speed and we use ff objets to allow the handling of very large objects. The output can include files for creating HTML with imagemaps.

Usage

```
pChromPlot(outRDataName, cghRDataName, chromRDataName,
           probenamesRDataName = NULL,
           posRDataName = NULL,
           imgheight = 500,
           pixels.point = 3,
           pch = 20,
           colors = c("orange", "red", "green", "blue", "black"),
           imagemap = FALSE,
           typeParall = "fork",
           mc.cores = detectCores(),
```

```
            typedev = "default",
            certain_noNA = FALSE,
            loadBalance = TRUE,
            ...)
```

Arguments

outRDataName	The RAM object or the RData file name that contains the results from the segmentation (as an ffdf object), as carried out by any of the pSegment functions. Note that the type of object in outRDataName, cghRDataName, chromRDataName, posRDataName, should all be of the same type: all ff objects, or all RAM objects. As well, note that if you use RAM objects, you must use typeParall = "fork"; with ff objects you can use both typeParall = "cluster" and typeParall = "fork". Further details are provided in the vignette.
cghRDataName	The Rdata file name that contains the ffdf with the aCGH data or the name of the RAM object with the data. If this is an ffdf object, it can be created using as.ffdf with a data frame with genes (probes) in rows and subjects or arrays in columns. You can also use inputToADaCGH to produce these type of files.
chromRDataName	The RData file name with the ff (short integer) vector with the chromosome indicator, or the name of the RAM object with the data. Function inputToADaCGH produces these type of files.
posRDataName	The RData file name with the ff double vector with the location (e.g., position in kbases) of each probe in the chromosome, or the name of the RAM object with the data. Function inputToADaCGH produces these type of files. This argument is used for the spacing in the plots. If NULL, the x-axis goes from 1:number of probes in that chromosome.
probenamesRDataName	The RData file name with the vector with the probe names or the RAM object. Function inputToADaCGH produces these type of files. (Note even if this is an RData file stored on disk, this is not an ff file.) This won't be needed unless you set imagemap = TRUE.
imgheight	Height of png image. See png.
pixels.point	Approximate number of pixels that each point takes; this determines also final figure size. With many probes per chromosome, you will want to make this a small value.
pch	The type of plotting symbol. See par.
colors	A five-element character vector with the colors for: probes without change, probes that have a "gained" status, probes that have a "lost" status, the line that connects (smoothed values of) probes, the horizontal line at the 0 level.
imagemap	If FALSE only the png figure is produced. If TRUE, for each array * chromosome, two additional files are produced: "pngCoord_ChrNN@MM" and "geneNames_ChrNN@MM", where "NN" is the chromosome number and "MM" is the array name. The first file contains the coordinates of the png and radius and the second the gene or probe names, so that you can easily produce an HTML imagemap. (Former versions of ADaCGH did this automatically with Python. In this version we include the Python files under "imagemap-example".)

typeParall	One of "fork" or "cluster". "fork" is unavailable in Windows, and will lead to sequential execution. "cluster" requires having set up a cluster before, with appropriate calls to makeCluster, in which case the cluster can be one of the available types (e.g., sockets, MPI, etc). Using "fork" and "cluster" will lead to different schemes for parallelization. See the vignette. If you use ff objects, you can use different options for typeParall for segmentation and plotting.
mc.cores	The number of cores used if typeParall = "fork". See details in mclapply.
typedev	The device type. One of "cairo", "cairo-png", "Cairo", or "default". "Cairo" requires the Cairo package to be available, but might work with headless Linux server without png support, and might be a better choice with Mac OS. "default" chooses "Cairo" for Mac, and "cairo" otherwise.
certain_noNA	Are you certain, absolutely sure, your data contain no missing values? (Default is FALSE). If you are, you can achieve considerable speed ups by setting it to TRUE. See the help for this option in pSegment. Of course, if you are setting it to true, the object you pass with the output, outRDataName, must have been generated using certain_noNA.
loadBalance	If TRUE (the default) use load balancing with MPI (use clusterApplyLB instead of clusterApply) and a similar approach for forking (set mc.preschedule = FALSE in the call to mclapply).
...	Additional arguments; not used.

Value

Used only for its side effects of producing PNG plots, stored in the current working directory (getwd().)

Author(s)

Ramon Diaz-Uriarte <rdiaz02@gmail.com>

See Also

pSegment

Examples

```
######################################################
###
### Using forking with RAM objects
###
######################################################

### Note to windows users: under Windows, this will
### result in sequential execution, as forking is not
### available.


## Get example input data and create data objects
```

```
data(inputEx)

## (this is not necessary, but is convenient;
##  you could do the subsetting in the call themselves)
cgh.dat <- inputEx[, -c(1, 2, 3)]
chrom.dat <- as.integer(inputEx[, 2])
pos.dat <- inputEx[, 3]



## Segment with HaarSeg
haar.RAM.fork <- pSegmentHaarSeg(cgh.dat, chrom.dat,
                                    merging = "MAD")


pChromPlot(haar.RAM.fork,
           cghRDataName = cgh.dat,
           chromRDataName = chrom.dat,
           posRDataName = pos.dat,
           imgheight = 350)


## Not run:

#####################################################
###
### Using a cluster with ff objects and create imagemaps
###
#####################################################



## Create a temp dir for storing output
dir.create("ADaCGH2_plot_tmp_dir")
originalDir <- getwd()
setwd("ADaCGH2_plot_tmp_dir")


## Start a socket cluster. Change the appropriate number of CPUs
## for your hardware and use other types of clusters (e.g., MPI)
## if you want.

cl2 <- makeCluster(4,"PSOCK")
clusterSetRNGStream(cl2)
setDefaultCluster(cl2)
clusterEvalQ(NULL, library("ADaCGH2"))
## The following is not really needed if you create the cluster AFTER
## changing directories. But better to be explicit.
wdir <- getwd()
clusterExport(NULL, "wdir")
clusterEvalQ(NULL, setwd(wdir))


## Get input data in ff format
## (we loaded the RData above, but we need to find the full path
## to use it in the call to inputToADaCGH)
```

```
fname <- list.files(path = system.file("data", package = "ADaCGH2"),
                    full.names = TRUE, pattern = "inputEx.RData")

inputToADaCGH(ff.or.RAM = "ff",
              RDatafilename = fname)



## Segment with HaarSeg

haar.ff.cluster <- pSegmentHaarSeg("cghData.RData",
                                   "chromData.RData",
                                   merging = "MAD",
                                   typeParall= "cluster")

## Save the output (an ff object) and plot
save(haar.ff.cluster, file = "haar.ff.cluster.out.RData",
     compress = FALSE)


pChromPlot(outRDataName = "haar.ff.cluster.out.RData",
           cghRDataName = "cghData.RData",
           chromRDataName = "chromData.RData",
           posRDataName = "posData.RData",
           probenamesRDataName = "probeNames.RData",
           imgheight = 350,
           imagemap = TRUE,
           typeParall= "cluster")

### Explicitly stop cluster
stopCluster(NULL)

### Clean up (DO NOT do this with objects you want to keep!!!)
load("chromData.RData")
load("posData.RData")
load("cghData.RData")

delete(cghData); rm(cghData)
delete(posData); rm(posData)
delete(chromData); rm(chromData)
unlink("chromData.RData")
unlink("posData.RData")
unlink("cghData.RData")
unlink("probeNames.RData")

lapply(haar.ff.cluster, delete)
rm(haar.ff.cluster)
unlink("haar.ff.cluster.out.RData")

### Try to prevent problems in R CMD check
## Sys.sleep(2)


### Delete all png files and temp dir
setwd(originalDir)
## Sys.sleep(2)
unlink("ADaCGH2_plot_tmp_dir", recursive = TRUE)
```

```
## Sys.sleep(2)


## End(Not run)

### PNGs are in this directory
getwd()
```

---

pSegment *Parallelized/"unified" versions of several aCGH segementation algorithms/methods*

Description

These functions parallelize several segmentation algorithms and make their calling use the same conventions as for other methods.

Usage

```
pSegmentDNAcopy(cghRDataName, chromRDataName, merging = "MAD",
                mad.threshold = 3, smooth = TRUE,
                alpha=0.01, nperm=10000,
                p.method = "hybrid",
                min.width = 2,
                kmax=25, nmin=200,
                eta = 0.05,trim = 0.025,
                undo.splits = "none",
                undo.prune=0.05, undo.SD=3,
                typeParall = "fork",
                mc.cores = detectCores(),
                certain_noNA = FALSE,
                loadBalance = TRUE,
                ...)

pSegmentHaarSeg(cghRDataName, chromRDataName,
                merging = "MAD", mad.threshold = 3,
                W = vector(),
                rawI = vector(),
                breaksFdrQ = 0.001,
                haarStartLevel = 1,
                haarEndLevel = 5,
                typeParall = "fork",
                mc.cores = detectCores(),
                certain_noNA = FALSE,
                loadBalance = FALSE,
                ...)

pSegmentHMM(cghRDataName, chromRDataName,
            merging = "mergeLevels", mad.threshold = 3,
            aic.or.bic = "AIC",
```

```
              typeParall = "fork",
              mc.cores = detectCores(),
              certain_noNA = FALSE,
              loadBalance = TRUE,
              ...)


pSegmentBioHMM(cghRDataName, chromRDataName, posRDataName,
                merging = "mergeLevels", mad.threshold = 3,
                aic.or.bic = "AIC",
                typeParall = "fork",
                mc.cores = detectCores(),
                certain_noNA = FALSE,
                loadBalance = TRUE,
                ...)

pSegmentCGHseg(cghRDataName, chromRDataName, CGHseg.thres = -0.05,
                merging = "MAD", mad.threshold = 3,
                typeParall = "fork",
                mc.cores = detectCores(),
                certain_noNA = FALSE,
                loadBalance = TRUE,
                ...)


pSegmentGLAD(cghRDataName, chromRDataName,
              deltaN = 0.10,
              forceGL = c(-0.15, 0.15),
              deletion = -5,
              amplicon = 1,
              typeParall = "fork",
              mc.cores = detectCores(),
              certain_noNA = FALSE,
              GLADdetails = FALSE,
              loadBalance = TRUE,
              ...)


pSegmentWavelets(cghRDataName, chromRDataName, merging = "MAD",
                  mad.threshold = 3,
                  minDiff = 0.25,
                  minMergeDiff = 0.05,
                  thrLvl = 3, initClusterLevels = 10,
                  typeParall = "fork",
                  mc.cores = detectCores(),
                  certain_noNA = FALSE,
                  loadBalance = TRUE,
                  ...)
```

Arguments

cghRDataName	The Rdata file name that contains the ffdf with the aCGH data or the name of the in-memory, RAM, R object with the data (a data frame). If this is an ffdf object, it can be created using as.ffdf with a data frame with genes (probes) in rows and subjects or arrays in columns. You can also use inputToADaCGH to produce these type of files. Note that the type of object in cghRDataName, chromRDataName, posRDataName, should all be of the same type: all ff objects, or all RAM objects, the usual R objects. Moreover, the type of input determines the type of output: if you use ff objects as input, you will get the output as an ff object.
chromRDataName	The RData file name with the ff (short integer) vector with the chromosome indicator, or the name of the in-memory RAM R object with the data. Function inputToADaCGH produces these type of files.
posRDataName	The RData file name with the ff double vector with the location (e.g., position in kbases) of each probe in the chromosome, or the name of the in-memory RAM R object with the data. Function inputToADaCGH produces these type of files.
merging	Merging method; for most methods one of "MAD" or "mergeLevels". For CBS (pSegmentDNAcopy), GGHseg (pSegmentCGHseg), HaarSeg (pSegmentHaarSeg), and Wavelets (as in Hsu et al. —pSegmentWavelets) also "none". This option does not apply to GLAD (which has its own merging-like approach). See details.
mad.threshold	If using merging = "MAD" the value such that all segments where abs(smoothed value) > m*MAD will be declared aberrant —see p. i141 of Ben-Yaacov and Eldar. No effect if merging = "mergeLevels" (or "none").
typeParall	One of "fork" or "cluster". "fork" is unavailable in Windows, and will lead to sequential execution. "cluster" requires having set up a cluster before, with appropriate calls to makeCluster, in which case the cluster can be one of the available types (e.g., sockets, MPI, etc). Using "fork" and "cluster" will lead to different schemes for parallelization. See details and the vignette.
mc.cores	The number of cores used if typeParall = "fork". See details in mclapply.
certain_noNA	Are you certain, absolutely sure, your data contain no missing values? (Default is FALSE). If you are, you can achieve considerable speed ups by setting it to TRUE. But if you set it to TRUE and you are wrong, some methods will fail (some with harder to understand error messages) and, even worse, other methods might appear to work (but give incorrect results). You've been warned.
loadBalance	If TRUE (the default for all methods except HaarSeg) use load balancing with MPI (use clusterApplyLB instead of clusterApply) and a similar approach for forking (set mc.preschedule = FALSE in the call to mclapply).
smooth	For DNAcopy only. If TRUE (default) carry out smoothing as explained in smooth.CNA.
alpha	For DNAcopy only. See segment.
nperm	For DNAcopy only. See segment.
p.method	For DNAcopy only. See segment.

`min.width` For DNAcopy only. See segment.

`kmax` For DNAcopy only. See segment.

`nmin` For DNAcopy only. See segment.

`eta` For DNAcopy only. See segment.

`trim` For DNAcopy only. See segment.

`undo.splits` For DNAcopy only. See segment.

`undo.prune` For DNAcopy only. See segment.

`undo.SD` For DNAcopy only. See segment.

`W` For HaarSeg: Weight matrix, corresponding to quality of measurment. Insert 1/(sigma**2) as weights if your platform output sigma as the quality of measurment. W must have the same size as I.

`rawI` For HaarSeg. Mininum of the raw red and raw green measurment, before the log. rawI is used for the non-stationary variance compensation. rawI must have the same size as I.

`breaksFdrQ` For HaarSeg. The FDR q parameter. Common used values are 0.05, 0.01, 0.001. Default value is 0.001.

`haarStartLevel` For HaarSeg. The detail subband from which we start to detect peaks. The higher this value is, the less sensitive we are to short segments. The default is value is 1, corresponding to segments of 2 probes.

`haarEndLevel` For HaarSeg. The detail subband until which we use to detect peaks. The higher this value is, the more sensitive we re to large trends in the data. This value DOES NOT indicate the largest possible segment that can be detected. The default is value is 5, corresponding to step of 32 probes in each direction.

`aic.or.bic` For HMM and BioHMM. One of "AIC" or "BIC". See 'criteria' in runBioHMM.

`CGHseg.thres` The threshold for the adaptive penalization in Picard et al.'s CGHseg. See p. 13 of the original paper. Must be a negative number. The default value used in the original reference is -0.5. However, our experience with the simulated data in Willenbrock and Fridlyand (2005) indicates that for those data values around -0.005 are more appropriate. We use here -0.05 as default.

`deltaN` Only for GLAD. See 'deltaN' in daglad.

`forceGL` Only for GLAD. See 'forceGL' in daglad.

`deletion` Only for GLAD. See 'deletion' in daglad.

`amplicon` Only for GLAD. See 'amplicon' in daglad.

`GLADdetails` Only for GLAD. If set to TRUE the function returns verbose output about where it is along the execution. This option (setting it to FALSE) is likely to become hard-coded in the future.

`minMergeDiff` Used only when doing merging in the wavelet method of Hsu et al.. The finall call as to which segments go together is done by a `mergeLevels` approach, but an initial collapsing of very close values is performed (otherwise, we could end up passing to mergeLevels as many initial levels as there are points).

`minDiff` For Wavelets (Hsu et al.). Minimum (absolute) difference between the medians of two adjacent clusters for them to be considered truly different. Clusters "closer" together than this are collapsed together to form a single cluster.

`thrLvl` The level used for the wavelet thresholding in Hsu et al.

`initClusterLevels`
For Wavelets (Hsu et al.). The initial number of clusters to form.

`...` Additional arguments; not used.

Details

In most cases, these are wrappers to the original code, with modifications for parallelization and for using `ff` objects, if appropriate.

Using option `typeParall = "fork"` will, as it says, use the forking mechanism available in package `parallel`. The objects used can be either `ff` objects or regular R objects. Using `typeParall = "cluster"` will use a pre-existing cluster, and the objects used must be `ff` ones, since we only pass pointers to the objects, not the objects themselves, to try to minimize communication and memory usage. To put it the other way around: if you use RAM objects, you must use `typeParall = "fork"`; with `ff` objects you can use both `typeParall = "cluster"` and `typeParall = "fork"`. Further details are provided in the vignette.

For HMM, BioHMM, CGHseg, and Wavelets, the first part of the analysis is conducted parallelizing over array by chromosome (because the methods are slow and/or very memory consuming). The final step (merging), however, is carried out over array (it is a step that must be carried array-wise). For all other methods, we have parallelized over arrays: the extra communication overheads of the much finer-grained parallelization of array by chromosome are rarely justified with these methods and, in the case of GLAD, would require modifying the original C code.

CGHseg has been implemented here following the original authors description. Note that several publications incorrectly claim that they use the CGHseg approach when, actually, they are only using the "segment" function in the "tilingArray" package, but they are missing the key step of choosing the optimal number of segments (see p. 13 in Picard et al, 2005). We implement the author's method in our (internal, so use "ADaCGH2:::piccardsKO" to see it) function "piccardsKO".

When using GLAD, we use the HaarSeg approach. This is the same as using the `daglad` function with argument `smoothfunc = "haarseg"`.

For BioHMM and HMM the smoothed results are merged, by default by the mergeLevels algorithm, as recommended in *Willenbrock and Fridlyand, 2005*. For DNAcopy the default used to be mergeLevels, following the above recommendations, but we are now using MAD by default, as it is much faster and it is unclear that mergeLevels is the right approach with the type of data available today. Your mileage might vary and you probably will want to try both on some test data and check which makes more sense.

Merging is also done in GLAD (with GLAD's own merging algorithm). For HaarSeg, calling/merging is carried out using MAD, following page i141 of Ben-Yaacov and Eldar, section 2.3, "Determining aberrant intervals": a MAD (per their definition) is computed and any segment with absolute value larger than mad.threshold * MAD is considered aberrant. Merging is also performed for CGHseg (the default, however, is MAD, not mergeLevels). Merging (using either of "mergeLevels" or "MAD") can also be used with the wavelet-based method of Hsu et al.; please note that the later is an experimental feature implemented by us, and there is no study of its performance.

In summary, for all segmentation methods (except GLAD) merging is available as either "mergeLevels" or "MAD". For DNAcopy, CGHseg, HaarSeg, and wavelets as in Hsu et al., you can also choose no merging, though this will rarely be what you want (we offer this option to allow using the original authors' choices in their first descriptions of methods).

When using mergeLevels, we map the results to states of "Alteration", so that we categorize each probe as taking one, and only one, of three possible values, -1 (loss of genomic DNA), 0 (no change in DNA content), +1 (gain of genomic DNA). We have made the assumption, in this mapping, that the "no change" class is the one that has the absolute value closest to zero, and any other classes are either gains or losses. When the data are normalized, the "no change" class should be the most common one. When using MAD this step is implicit in the procedure (any segment with absolute value larger than mad.threshold * MAD is considered aberrant).

Note that "mergeLevels", in addition to being used for calling gains and losses, results in a decrease in the number of distinct smoothed values, since it can merge two or more adjacent smoothed levels. "MAD", in contrast, performs no merging as such, but only calling.

Value

A list of two components (the components will be either ff or regular, in-memory R objects, depending on the input):

outSmoothed	The smoothed values, as either a ffdf object or a data frame object. Each column is an array or sample, and each row a probe.
outState	The calls for each probe, as either an ffdf object or a data frame object. Each column is an array or sample, and each row a probe. For methods that accept "none" as an argument to 'merging', the states cannot be interpreted directly as gain or loss; they are simply discrete codes for distinct segments.

If the output uses ffdf, rows and columns of each element can be accessed in the usual way for ffdf objects, but accept also most of the usual R operations for data frames.

Author(s)

The code for DNAcopy, HMM, BioHMM, and GLAD are basically wrappers around the original functions by their corresponding authors, with some modiffications for parallelization and usage of ff objects. The original packages are: DNAcopy, aCGH, snapCGH, cgh, GLAD, respectively. The CGHseg method uses package tilingArray.

HaarSeg has been turned into an R package, available from https://r-forge.r-project.org/projects/haarseg/. That package uses, at its core, the same R and C code as we do, from Ben-Yaacov and Eldar. We have not used the available R package for historical reasons (we used Eldar and Ben-Yaacov's C and R code in the former ADaCGH package, before a proper R package was available).

For the wavelet-based method we have only wrapped the code that was kindly provided by L. Hsu and D. Grove, and parallelized a few calls. Their original code is included in the sources of the package.

Parallelization and modifications for using ff and additions are by Ramon Diaz-Uriarte <rdiaz02@gmail.com>

References

Diaz-Uriarte, R. (2014). ADaCGH2: parallelized analysis of (big) CNA data. *Bioinformatics*, **30**: 1759–1761.

Carro A, Rico D, Rueda O M, Diaz-Uriarte R, and Pisano DG. (2010). waviCGH: a web application for the analysis and visualization of genomic copy number alterations. *Nucleic Acids Research*, **38 Suppl**:W182–187.

Fridlyand, Jane and Snijders, Antoine M. and Pinkel, Dan and Albertson, Donna G. (2004). Hidden Markov models approach to the analysis of array CGH data. *Journal of Multivariate Analysis*, **90**: 132–153.

Hsu L, Self SG, Grove D, Randolph T, Wang K, Delrow JJ, Loo L, Porter P. (2005) Denoising array-based comparative genomic hybridization data using wavelets. *Biostatistics*, **6**:211-26.

Hupe, P. and Stransky, N. and Thiery, J. P. and Radvanyi, F. and Barillot, E. (2004). Analysis of array CGH data: from signal ratio to gain and loss of DNA regions. *Bioinformatics*, **20**: 3413–3422.

Lingjaerde OC, Baumbusch LO, Liestol K, Glad I, Borresen-Dale AL. (2005). CGH-Explorer: a program for analysis of CGH-data. *Bioinformatics*, **21**: 821–822.

Marioni, J. C. and Thorne, N. P. and Tavare, S. (2006). BioHMM: a heterogeneous hidden Markov model for segmenting array CGH data. *Bioinformatics*, **22**: 1144–1146.

Olshen, A. B. and Venkatraman, E. S. and Lucito, R. and Wigler, M. (2004) Circular binary segmentation for the analysis of array-based DNA copy number data. *Biostatistics*, **4**, 557–572. http://www.mskcc.org/biostat/~olshena/research.

Picard, F. and Robin, S. and Lavielle, M. and Vaisse, C. and Daudin, J. J. (2005). A statistical approach for array CGH data analysis. *BMC Bioinformatics*, **6**, 27. http://dx.doi.org/10.1186/1471-2105-6-27.

Price TS, Regan R, Mott R, Hedman A, Honey B, Daniels RJ, Smith L, Greenfield A, Tiganescu A, Buckle V, Ventress N, Ayyub H, Salhan A, Pedraza-Diaz S, Broxholme J, Ragoussis J, Higgs DR, Flint J, Knight SJ. (2005) SW-ARRAY: a dynamic programming solution for the identification of copy-number changes in genomic DNA using array comparative genome hybridization data. *Nucleic Acids Res.*, **33**:3455-64.

Willenbrock, H. and Fridlyand, J. (2005). A comparison study: applying segmentation to array CGH data for downstream analyses. *Bioinformatics*, **21**, 4084–4091.

Diaz-Uriarte, R. and Rueda, O.M. (2007). ADaCGH: A parallelized web-based application and R package for the analysis of aCGH data, *PLoS ONE*, **2**: e737.

Ben-Yaacov, E. and Eldar, Y.C. (2008). A Fast and Flexible Method for the Segmentation of aCGH Data, *Bioinformatics*, **24**: i139-i145.

See Also

pChromPlot, inputToADaCGH

Examples

```
#####################################################
###
### Using forking with RAM objects
###
#####################################################

### Note to windows users: under Windows, this will
### result in sequential execution, as forking is not
### available.


## Get example input data and create data objects

data(inputEx)

## (this is not necessary, but is convenient;
##  you could do the subsetting in the call themselves)
cgh.dat <- inputEx[, -c(1, 2, 3)]
chrom.dat <- as.integer(inputEx[, 2])
pos.dat <- inputEx[, 3]



## Segment with HaarSeg
haar.RAM.fork <- pSegmentHaarSeg(cgh.dat, chrom.dat,
                                 merging = "MAD")

## What does the output look like?
```

```
lapply(haar.RAM.fork, head)

## Where and what length are segments in first sample?
rle(haar.RAM.fork$outSmoothed[, 1])


## Repeat, without load-balancing
haar.RAM.fork.nlb <- pSegmentHaarSeg(cgh.dat, chrom.dat,
                                     merging = "MAD",
                                     loadBalance = FALSE)


if(.Platform$OS.type != "windows") {

## We do not want this to run in Windows the automated tests since
## issues with I/O. It should work, though, in interactive usage


#####################################################
###
### Using forking with ff objects
###
#####################################################

### Note to windows users: under Windows, this will
### result in sequential execution, as forking is not
### available.



## Create a temp dir for storing output and ff objects.
## (Not needed, but cleaner).

dir.create("ADaCGH2_example_tmp_dir")
originalDir <- getwd()
setwd("ADaCGH2_example_tmp_dir")


## Get input data in ff format
## (we loaded the RData above, but we need to find the full path
##  to use it in the call to inputToADaCGH)

fname <- list.files(path = system.file("data", package = "ADaCGH2"),
                    full.names = TRUE, pattern = "inputEx.RData")

inputToADaCGH(ff.or.RAM = "ff",
                      RDatafilename = fname)


## Segment with HaarSeg

haar.ff.fork <- pSegmentHaarSeg("cghData.RData",
                                "chromData.RData",
                                 merging = "MAD")

## What does the output look like?
haar.ff.fork
```

```
## Note the warnings; we will be gentler in next example.


#####################################################
###
### Using a cluster with ff objects
###
#####################################################


## Start a socket cluster. Change the appropriate number of CPUs
## for your hardware and use other types of clusters (e.g., MPI)
## if you want.

cl2 <- parallel::makeCluster(4,"PSOCK")
parallel::clusterSetRNGStream(cl2)
parallel::setDefaultCluster(cl2)
parallel::clusterEvalQ(NULL, library("ADaCGH2"))
## The following is not really needed if you create the cluster AFTER
## changing directories. But better to be explicit.
wdir <- getwd()
parallel::clusterExport(NULL, "wdir")
parallel::clusterEvalQ(NULL, setwd(wdir))



## Segment with HaarSeg

haar.ff.cluster <- pSegmentHaarSeg("cghData.RData",
                                   "chromData.RData",
                                   merging = "MAD",
                                   typeParall= "cluster")


## Avoid warnings by opening the objects
names(haar.ff.cluster)
open(haar.ff.cluster$outSmoothed)
open(haar.ff.cluster$outState)

## Alternatively, we can open the two ffdfs with lapply
## lapply(haar.ff.cluster, open)



#########################################
###
### Compare output (should be identical)
###
#########################################

all.equal(haar.ff.cluster$outSmoothed[ , ],
          haar.ff.fork$outSmoothed[ , ])

all.equal(haar.ff.cluster$outSmoothed[ , ],
          haar.RAM.fork$outSmoothed[ , ])

identical(haar.ff.cluster$outState[ , ],
```

```
          haar.ff.fork$outState[ , ])

identical(haar.ff.cluster$outState[ , ],
          haar.RAM.fork$outState[ , ])


######################################################################
####
####                    Clean up actions
####
####  (These are not needed. They are convenient here, to prevent
####   leaving garbage in your hard drive. In "real life" you will
####   have to decide what to delete and what to store).
######################################################################


### Explicitly stop cluster
parallel::stopCluster(cl2)

### All objects (RData and ff) are left in this directory
getwd()

### We will clean it up, and do it step-by-step
### BEWARE: DO NOT do this with objects you want to keep!!!

## Remove ff and RData for the data

load("chromData.RData")
load("posData.RData")
load("cghData.RData")

delete(cghData); rm(cghData)
delete(posData); rm(posData)
delete(chromData); rm(chromData)
unlink("chromData.RData")
unlink("posData.RData")
unlink("cghData.RData")
unlink("probeNames.RData")


## Remove ff and R objects with segmentation results

lapply(haar.ff.fork, delete)
rm(haar.ff.fork)

lapply(haar.ff.cluster, delete)
rm(haar.ff.cluster)


### Try to prevent problems in R CMD check
## Sys.sleep(2)

### Delete temp dir
setwd(originalDir)
## Sys.sleep(2)
unlink("ADaCGH2_example_tmp_dir", recursive = TRUE)
## Sys.sleep(2)
```

```
}
```

Index