Package 'JaBbA'

January 25, 2018

Title Resconstruct Rearrangment Graphs from Cancer Whole Genome Sequencing

Version 0.0.0.9000

Description JaBbA simultaneously infers integer copy numbers for genomic segments and their connections based on the read depth and the graph structure defined by the structural variations identified from WGS data. Two essential inputs are 1) a high-density read depth ratio between tumor and normal samples (we also have guidelines for the cases lacking paired normal sequencing) and 2) the putative somatic structural variations. The output is a somatic rearrangement graph representing the integer copy numbers of DNA segments and their connections.

Depends R (>= 3.1.1),
Matrix,
gTrack,
gUtils

Imports data.table (>= 1.9),
GenomicRanges (>= 1.18),
igraph (>= 0.7),
gUtils,
gTrack

License What license is it under?

LazyData true

RoxygenNote 6.0.1

R topics documented:

`abs2rel` *abs2rel*

Description

rescales CN values from relative to "absolute" (i.e. per cancer cell copy) scale given purity and ploidy By default, output is normalized to 1 (i.e. assumes that the total relative copy number signal mass over the genome is 1)

Usage

```
abs2rel(gr, purity = NA, ploidy = NA, gamma = NA, beta = NA,
  field = "cn", field.ncn = "ncn", total = 1)
```

Arguments

gr	GRanges input with meta data field corresponding to mean relative copy "mean" in that interval
purity	purity of sample
ploidy	ploidy of sample
gamma	gamma fit of solution (over-rides purity and ploidy)
beta	beta fit of solution (over-rides purity and ploidy)
field	meta character specifying meta data field in "gr" variable from which to extract signal, default "mean"
field.ncn	character specifying meta data field in "gr" variable from which to extract germline integer copy number, default "ncn", if doesn't exist, germline copy number is assumed to be zero

Details

takes in gr with signal field "field"

Value

numeric vector of integer copy numbers

---

adj2inc *adj2inc*

Description

converts adjacency matrix (of directed graph) into incidence matrix - ie an nodes x edges matrix, for each edge i connecting node j to k, column i will have -1 at position j and +1 at position k

Usage

```
adj2inc(A)
```

---

all.paths *all.paths*

Description

Low level function to enumerate all elementary paths and cycles through graph

Usage

```
## S3 method for class 'paths'
all(A, all = F, ALL = F, sources = c(), sinks = c(),
  source.vertices = sources, sink.vertices = sinks, exclude = NULL,
  verbose = FALSE, ...)
```

Arguments

A	nxn adjacency matrix
all	logical flag, if all = T, will include all sources (parentless vertices) and sinks (childless vertices) in path computati
ALL	logical flag, if ALL = T, will also include vertices without outgoing and incoming edges in paths
sources	graph indices to treat as sources (by default is empty)
sinks	graph indices to treat as sinks (by default is empty)
verbose	logical flag

Details

takes directed graph represented by n x n binary adjacency matrix A and outputs all cycles and paths between source.vertices, sink.vertices

Value

list of integer vectors corresponding to indices in A (i.e. vertices) $paths = paths indices $cycles = cycle indices

---

anno.hops *anno.hops*

Description

Adds simple annotations to GRangesList of walks including distance along each reference fragment and distance between "hops"

Usage

```
anno.hop(walks)
```

Arguments

walks	walks to annotate

Author(s)

Marcin Imielinski

`annotate.walks` *annotate.walks*

Description

Low level function to annotate walks (GRanges list) with cds / promoter annotations

Usage

```
annotate.walks(walks, cds, promoters = NULL, filter.splice = T,
  verbose = F, prom.window = 1000, max.chunk = 1e+09, mc.cores = 1,
  exhaustive = FALSE)
```

Arguments

`walks`	GRangesList of walks to query (eg from traversal of JaBbA object)
`cds`	GRangesList of CDS annotation, one per transcript (each a coding region of an exon)
`promoters`	GRanges of promoters (same length as transcript)
`filter.splice`	flag whether to filter out splice variants of a given gene
`verbose`	flag
`prom.window`	window to use around each transcript to identify putative promoter if promoter is NULL
`mc.cores`	number of cores to use

Details

given: walks: input grl of walks on the genome tx: gr annotating transcript boundaries or grl annotating exon level transcripts e.g. refgene or grl with grl-level meta data fields $s1, $s2, $e1, $e2 annotating start and end positions of transcipt and cds respectively, $gene_sym representing gene label, $chr - chromosome, $str strand and gr level features (exon_frame) annotating the frame (0,1,2) of the first exon position in a + transcript and last exon position in a - transcript. Assumes that exons are ordered in each grl item in the order of transcription (i.e. right most exon for negative strand transcripts) (e.g. output of read_refGene(grl = T))

Value

a grl of putative fusions with annotations in values field: $label $type e.g. promoter-fusion, 5-UTR fusion, In-frame fusion, 3' truncated fusion, 5' truncated fusion, in-frame poly-fusion, 3' truncated poly-fusion, 5' truncated poly-fusion $genes genes involved in fusion $transcripts transcripts involved in fusion

annotates every possible altered transcript in region including - transcripts with truncated

`cbs_stub` *run cbs*

Description

run cbs

Usage

```
cbs_stub(cov.file, out.file, field = "ratio", alpha = 1e-05)
```

`chromoplexy` *chromoplexy*

Description

Determines chromoplexy paths from standard JaBbA output

Usage

```
chromoplexy(kag = NULL, jab = NULL, sol = NULL, all = F, ref.only = F,
  filt.jab = T, reciprocal = TRUE, hijacked = TRUE, paths = F,
  dist = 1000, cn.dist = dist, verbose = F, interval = 400,
  mc.cores = 1, chunksize = 5000)
```

Arguments

`jab`	JaBbA object
`all`	logical flag whteher to enumerate all possible cycles, otherwise will return (an arbitrary) minimal decompositoin into the shortest "chains" of balanced rearrangements
`paths`	logical flag if paths = T, will also try to compute paths (in addition to cycles), default = FALSE
`dist`	maximum distance at which to cluster junctions, i.e. in which to consider a deletion or amplification bridge
`cn.dist`	minimum distance at which to enforce copy concordance for deletion and amplification bridges

Details

Outputs all chromoplexy paths and cycles (i.e. paths and cycles in breakpoint graph) allowing quasi-reciprocal rearrangements with amplification / deletion bridge distance threshold "dist"

Value

paths and cycles of as list of vectors of aberrant edge index sequences, aberrant edges refer to edges described in kag$ab.edges matrix

`collapse.paths` *collapse.paths*

Description

collapse simple paths in a graph G (adjacency matrix or igraph object) returns m x m new adjacency matrix and map of old vertex id's to new ones $adj = m x m matrix #map = length n with indices 1 .. m

Usage

```
collapse.paths(G, verbose = T)
```

`convex.basis` *convex.basis*

Description

Outputs a matrix K of the convex basis of matrix A

Usage

```
convex.basis(A, interval = 80, chunksize = 100, exclude.basis = NULL,
  exclude.range = NULL, maxchunks = Inf, verbose = F)
```

Details

i.e. each column x = K[,i] is a minimal solution (with respect to sparsity) to Ax = 0, x>=0

exclude.basis = 0, 1 matrix of dimension k x ncol(A) specifying k sparsity patterns that we would like to exclude from the convex.basis. This can speed up computation since any non-negative combinations of vectors that satisfy an exclusion property will also be excludable, and thus we can remove such vectors as soon as we detect them..

exclude.range = 9, 1 matrix of dimension k x nrow(A) specifying k sparsity patterns that we would like exclude, but these are specified in terms of the range of abs(A) .. i.e. we want to exclude all basis vectors v such that nz(exclude.ranges[i,]) C nz(abs(A)*v)) for some pattern i. Again any non-neg linear comb of any intermediate-basis vector that satisfies this property will satisfy it, as a result we can exclude these vectors when we see them.

fusions *fusions*

Description

annotates all gene fusions given an n x n adjacency matrix A of n genomic segments seg and grl of transcripts (eg output of read_RefGene) seg must be (1) a tiling of the genome and (2) have copies of both + and - intervals for each genomic range (eg output of karyograph)

Usage

```
fusions(junctions = NULL, jab = NULL, cds = NULL, promoters = NULL,
  query = NULL, prom.window = 1000, verbose = T, max.chunk = 1e+10,
  cb.interval = 10000, cb.chunksize = 10000, cb.maxchunks = 1e+10,
  exhaustive = FALSE, debug = NULL, mc.cores = 1)
```

Arguments

junctions	GRangesList of junctions (each a length 2 GRanges)
jab	JaBbA object (overrides junctions input)
cds	CDS annotations (GrangesList of transcript composed of coordinates coding regions of exons)
promoters	GRanges of promoters (same length as transcript)
query	optional query limiting walks to specific regions of interest
prom.window	window to use around each transcript to identify putative promoter if promoter is NULL

Details

alternate input is a pile of junctions of ranges with strand orientation pointing AWAY from breakend

cds = gencode cds GRanges gff3 / gtf input

"gene_name" GRangesList meta data field is used in annotation and in not creating "splice" fusions that arise from different transcripts of the same gene.

Value

GRangesList of walks corresponding to transcript boundaires

`gr.tile.map` *gr.tile.map*

Description

Given two tilings of the genome (eg at different resolution) query and subject outputs a length(query) list whose items are integer vectors of indices in subject overlapping that overlap that query (strand non-specific)

Usage

```
gr.tile.map(query, subject, mc.cores = 1, verbose = FALSE)
```

Arguments

`query`	Query
`subject`	Subject
`mc.cores`	number of cores
`verbose`	Default FALSE

Note

Assumes that input query and subject have no gaps (including at end) or overlaps, i.e. ignores end() coordinates and only uses "starts"

`jab2json` *jab2json*

Description

Dumps JaBbA graph into json

Usage

```
jab2json(jab, file, maxcn = 100, maxweight = 100)
```

Arguments

`jab`	input jab object
`file`	output json file

Author(s)

Marcin Imielinski ++ = RL +- = RR -+ = LL

JaBbA *JaBbA*

Description

Module to run jbaMIP + preprocessing from text file or rds input and dump files out to text.

Generates the following files in the output directory:

karyograph.rds — file of unpopulated karyograph as an RDS file of a list object storing the output of karyograph

jabba.rds — file storing JaBbA object

jabba.simple.rds — file storing JaBbA object simplified so that segments containing all unpopulated aberrant junctions are merged

jabba.raw.rds — storing raw jbaMIP solution, this may be useful for debugging and QC

jabba.png, jabba.simple.png — gTrack images of the above reconstructions

jabba.seg.txt — tsv file with jabba.simple solution segments

jabba.seg.rds — GRanges rds with jabba.simple solution segments

jabba.adj.txt — tsv file with edges (i.e. node pairs) of adjacency matrix populated with inferred copy numbers and node ids indexing segments in jabba.seg.txt

jabba.vcf, jabba.simple.vcf — BND-style vcf output of junctions in JaBbA output populated with rearrangement and interval copy numbers

jabba.cnv.vcf, jabba.simple.cnv.vcf — cfopy number style VCF showing jabba copy number output

Usage

```
JaBbA(ra, coverage, seg = NULL, cfield = NULL, tfield = NULL,
  nudge.balanced = FALSE, thresh.balanced = 500, outdir = "./",
  nseg = NULL, hets = NULL, name = "tumor", cores = 4, purity = NA,
  ploidy = NA, field = "ratio", subsample = NULL, tilim = 1200,
  mem = 16, reiterate = 0, rescue.window = 10000, init = NULL,
  edgenudge = 0.1, slack.penalty = 1000, overwrite = F)
```

Arguments

ra	GRangesList of junctions (i.e. bp pairs with strands oriented AWAY from break) OR path to junction VCF file (BND format), dRanger txt file or rds of GRangesList
coverage	GRanges of coverage OR path to cov file, rds of GRanges or .wig / .bed file of (normalized, GC corrected) fragment density
seg	optional path to existing segmentation, if missing then will segment abu using DNACopy with standard settings
cfield	character, junction confidence meta data field in ra
tfield	character, tier confidence meta data field in ra
outdir	out directory to dump into, default ./
nseg	optional path to normal seg file with $cn meta data field
hets	optional path to hets.file which is tab delimited text file with fields seqnames, start, end, alt.count.t, ref.count.t, alt.count.n, ref.count.n

name	prefix for sample name to be output to seg file
cores	number of cores to use (default 1)
field	field of coverage GRanges to use as fragment density signal (only relevant if coverage is GRanges rds file)
subsample	numeric between 0 and 1 specifying how much to sub-sample high confidence coverage data
tilim	integer scalar timeout in seconds for jbaMIP computation (default 1200 seconds)
mem	numeric scalar max memory in GB for MIP portion of algorithm (default 16)
reiterate	integer scalar specifying how many (re-)iterations of jabba to do, rescuing lower tier junctions that are near loose ends (requires junctions to be tiered via a grangeslist or VCF metadata field $tfield), tiers are 1 = must use, 2 = may use, 3 = use only in iteration>1 if near loose end
rescue.window	integer scalar bp window around which to rescue lower tier junctions
init	jabba object (list) or path to .rds file containing previous jabba object which to use to initialize solution, this object needs to have the identical aberrant junctions as the current jabba object (but may have different segments and loose ends, i.e. is from a previous iteration)
edgenudge	numeric hyper-parameter of how much to nudge or reward aberrant junction incorporation, default 0.1 (should be several orders of magnitude lower than average 1/sd on individual segments), a nonzero value encourages incorporation of perfectly balanced rearrangements which would be equivalently optimal with 0 copies or more copies.
slack.penalty	penalty to put on every loose.end copy, should be calibrated with respect to 1/(k*sd)^2 for each segment, i.e. that we are comfortable with junction balance constraints introducing k copy number deviation from a segments MLE copy number assignment (the assignment in the absence of junction balance constraints)
overwrite	logical flag whether to overwrite existing output directory contents or just continue with existing files.
nseg	path to data.frame or GRanges rds of normal seg file with coordinates and $cn data field specifying germline integer copy number

---

`jabba.alleles` *jabba.alleles*

Description

Populates allelic value s for JaBbA object. This does not explicitly impose junction balance constraints on alleles, but rather just computes the maximum likelihood estimate given allelic counts and the inferred total copy number on a given segment according to JaBbA

Usage

```
jabba.alleles(jab, het.sites, alt.count.field = "alt.count.t",
  ref.count.field = "ref.count.t", baf.field = "baf.t", split.ab = F,
  uncoupled = FALSE, conservative = FALSE, verbose = F)
```

Arguments

jab	JaBbA object
het.sites	GRanges with meta data fields (see below) for alt and rref count
alt.count.field	character specifying alt.count meta data field in input het.sites (default alt.count.t)
ref.count.field	character specifying ref.count meta data field in input het.sites (default ref.count.t)
split.ab	logical flag whether to split aberrant segmetns (segmentss with ab edge entering or leaving prior to computing allelic states (default FALSE)
uncoupled	logical flag whether to not collapse segments after inferring MLE estimate (default FALSE), if FALSE will try to merge adjacent segments and populate allele-specific junctions with copy numbers on the basis of the MLE fit on individual allelic segments
conservative	if TRUE then will leave certain allelic segments "unphased" if one cannot sync the high / low interval state with the incoming and / or outgoing junction state
verbose	logical flag

Value

list with following fields: $segstats = GRanges of input segments with $cn.high and $cn.low segments populated $asegstats = GRanges of allelic segments (length is 2*length(segstats)) with high and low segments each having $cn, this is a "melted" segstats GRAnges $atd = gTrack of allelic segments and supporting input het.sites $aadj = allelic adjacency matrix of allele specific junctions $ab.ix = indices of aberrant edges in $aadj $ref.ix = indices of reference edges in $aadj

---

JaBbA.digest *JaBbA.digest*

Description

JaBbA.digest

Usage

```
JaBbA.digest(jab, kag = NULL, verbose = T, keep.all = T)
```

Arguments

jab	JaBbA object "undigested"
kag	karyograph (original karyograph input to JaBbA), if NULL then will "redigest" JaBbA object
verbose	logical flag
keep.all	keep.all (default TRUE) whether to keep 0 copy junctions or collapse segments across these as well

Details

processes JaBbA object (1) collapsing segments with same copy number that lack loose ends (2) (optional) +/- adds segments correponding to loose ends (3) outputting edges data frame with colors, and other formatting information (4) outputting junctions GRangesList with copy number, color, lty and other plotting components

TODO: replace with proper object instantiator

`jabba.dist` *jabba.dist*

Description

Computes distance between pairs of intervals on JaBbA graph

Usage

```
jabba.dist(jab, gr1, gr2, matrix = T, directed = FALSE, max.dist = Inf,
  include.internal = TRUE, verbose = FALSE, EPS = 1e-09)
```

Arguments

`jab`	JaBbA object
`gr1`	interval set 1 GRanges
`gr2`	interval set 2 GRanges
`matrix`	flag whteher to output a matrix
`max.dist`	numeric (default = Inf), if non-infinity then output will be a sparse matrix with all entries that are greater than max.dist set to zero

Details

Given "jabba" object and input granges gr1 and gr2 of (signed) intervals

Value

a length(gr1) x length(gr2) matrix whose entries ij store the distance between the 3' end of gr1[i] and 5' end of gr2[j]

`jabba.gwalk` *jabba.gwalk*

Description

Computes greedy collection (i.e. assembly) of genome-wide walks (graphs and cycles) by finding shortest paths in JaBbA graph.

Usage

```
jabba.gwalk(jab, verbose = FALSE, return.grl = TRUE)
```

Arguments

`jab`	JaBbA object #

Value

GRangesList of walks with copy number as field $cn, cyclic walks denoted as field $is.cycle == TRUE, and $wid (width) and $len (segment length) of walks as additional metadata

Author(s)

Marcin Imielinski

Xiaotong Yao first peel off "simple" paths i.e. zero degree ends with >0 copy number so now we want to subtract that cn units of that path from the graph so we want to update the current adjacency matrix to remove that path while keeping track of of the paths on the stack then find paths that begin at a node and end at (one of its) immediate upstream neighbors this will be a path for whom col index is = parent(row) for one of the rows so now we want to subtract that cn units of that path from the graph so we want to update the current adjacency matrix to remove that path while keeping track of of the cycles on the stack

---

`jabba.hood` *jabba.hood*

Description

Given JaBbA object and seed window "win", outputs a reduced set of windows within neighborhoof of n coordinate (ork nodes) within the seed region(s) on the graph (only includes edges with weight !=0)

Usage

```
jabba.hood(jab, win, d = 0, k = NULL, pad = 0, ignore.strand = TRUE,
  bagel = FALSE, verbose = FALSE)
```

Arguments

jab	JaBbA object
win	GRanges of window of interest
d	= distance in coordinates on graph
k	Neighborhood on graph around window of interest to query
pad	pad level at which to collapse nearly reference adjacent intervals

Value

a reduced set of windows within neighborhood k of seed on the graph (only includes edges with weight !=0)

---

jabba.kid *jabba.kid*

Description

Given a set of gwalks (grangeslist outputs of jabba.gwalk) identifies strings "kidnapped" fragments i.e. strings of tempaled insertions, which are outputted as a grangeslist

Usage

```
jabba.kid(gwalks, pad = 5e+05, min.ab = 5e+05, min.run = 2)
```

Arguments

gwalks	grangeslist of walks (e.g.outputted from jabba.gwalks)
pad	how much bp neighboring material around each kidnapped string to include in outputs
min.ab	minimal bp distance a junction needs to be considered aberrant (e.g. to exclude very local deletions)
min.run	how many aberrant junctions to require in the outputted kidnapped fragments

Author(s)

Marcin Imielinski

jabba.melt *jabba.melt*

Description

jabba.melt

Usage

```
jabba.melt(jab, anti = FALSE, verbose = FALSE, mc.cores = 1,
  max.del = 10)
```

Arguments

jab	JaBbA object "undigested"
verbose	logical flag
kag	karyograph (original karyograph input to JaBbA), if NULL then will "redigest" JaBbA object
keep.all	keep.all (default TRUE) whether to keep 0 copy junctions or collapse segments across these as well

Details

melt JaBbA graph into "events" that decompose the total ploidy into amplicons (or deleticons, if anti = TRUE) Each amplicons / deleticon is flanked by either (1) junctions (2) loose ends or (3) chromosome ends / telomeres

jabba.walk *jabba.walk*

Description

Computes walks around all aberrant edges in JABbA object

Usage

```
jabba.walk(sol, kag = NULL, digested = T, outdir = "temp.walk",
  junction.ix = NULL, loci = NULL, clustersize = 100, trim = FALSE,
  trim.w = 1e+06, prune = FALSE, prune.d1 = 1e+05, prune.d2 = 1e+05,
  maxiterations = Inf, mc.cores = 1,
  genes = read.delim("~/DB/COSMIC/cancer_gene_census.tsv", strings =
  F)$Symbol, verbose = T, max.threads = 4, customparams = T, mem = 6,
  all.paths = FALSE, nomip = F, tilim = 100, nsolutions = 100,
  cb.interval = 10000, cb.chunksize = 10000, cb.maxchunks = 1e+10)
```

Arguments

sol	JaBbA object
outdir	output directory
junction.ix	junction indices around which to build walks (default is all junctions)
loci	loci around which to build walks (over-rides junction.ix), alternatively can be a list of "all.paths" objects (i.e. each a list utput of initial all.paths = TRUE run +/- field $prior for walk to re-eval a given all.paths combo
clustersize	size of the cluster to output around the locus or junction of interest
trim	logical flag whether trim in neighborhood of junction (only applicable if loci = NULL, default = TRUE)
trim.w	integer width to which to trim to
prune	flag whether to prune trivial walks for whom a path can be drawn from first to last interval in a graph linking intervals with pairwise distance < d1 on the walk or distance < d2 on the reference
prune.d1	local distance threshold for walk pruning
prune.d2	referenc distance threshold for walk pruning
mc.cores	number of cores to use, default 1
genes	character vector of gene symbols with which to annotate walk (eg cancer genes)
verbose	logical flag

Details

Takes in JaBbA solution and computes local reconstructions around all aberrant edges (default). Reconstructions (i.e. Huts) consists of collections of walks, each walk associated with a copy number, and a given region (collection of genomic windows). The interval sum of walks in a given region, weighted by copy numbers will recapitulate the marginal copy profile (as estimated by JaBbA). The reconstruction is chosen to maximize parsimony.

Optional flags allow making huts around specific junctions or specified loci (GRangesList)

Walks are reconstructed locally within "clustersize" nodes of each aberrant edge, where clustersize is measured by the number of total edges. Larger cluster sizes may fail to be computationally tractable, i.e. with a highly rearranged genome in an area of dense interconnectivity.

Value

list of walk set around each locus or junction that is inputted to analysis, each list item is a list with the following fields $win = input locus of interest, $grl = GRangesList of walks, $grs is a collapsed footprint of all walks in the walk list for this locu $td gTrack of of the output, additional outputs for debugging: $sol, $K, $Bc, $eix, $vix, $h

jabba2vcf *jabba2vcf*

Description

Converts jabba output to vcf file according to 4.2 "BND" syntax

Usage

```
jabba2vcf(jab, fn = NULL, sampleid = "sample", hg = skidb::read_hg(fft =
  T), cnv = FALSE)
```

Arguments

jab	JaBbA object
fn	output file name
sampleid	sample id
hg	human genome as BSgenome or ffTrack
cnv	flag whether to dump in CNV format

Value

returns character string or writes to file if specified

jabba_stub *jabba_stub*

Description

Internal function to run single iteration of JaBbA

Generates the following files in the output directory:

karyograph.rds — file of unpopulated karyograph as an RDS file of a list object storing the output of karyograph

jabba.rds — file storing JaBbA object

jabba.simple.rds — file storing JaBbA object simplified so that segments containing all unpopulated aberrant junctions are merged

jabba.raw.rds — storing raw jbaMIP solution, this may be useful for debugging and QC

jabba.png, jabba.simple.png — gTrack images of the above reconstructions

jabba.seg.txt — tsv file with jabba.simple solution segments

jabba.seg.rds — GRanges rds with jabba.simple solution segments

jabba.adj.txt — tsv file with edges (i.e. node pairs) of adjacency matrix populated with inferred copy numbers and node ids indexing segments in jabba.seg.txt

jabba.vcf, jabba.simple.vcf — BND-style vcf output of junctions in JaBbA output populated with rearrangement and interval copy numbers

jabba.cnv.vcf, jabba.simple.cnv.vcf — cfopy number style VCF showing jabba copy number output

Usage

```
jabba_stub(ra, coverage, seg = NULL, cfield = NULL, tfield = NULL,
  nudge.balanced = FALSE, thresh.balanced = 500, outdir = "./",
  nseg = NULL, hets = NULL, name = "tumor", cores = 4, purity = NA,
  ploidy = NA, field = "ratio", subsample = NULL, tilim = 1200,
  mem = 16, init = NULL, edgenudge = 0.1, slack.penalty = 1000,
  overwrite = F)
```

Arguments

ra	GRangesList of junctions (i.e. bp pairs with strands oriented AWAY from break) OR path to junction VCF file (BND format), dRanger txt file or rds of GRangesList
coverage	GRanges of coverage OR path to cov file, rds of GRanges or .wig / .bed file of (normalized, GC corrected) fragment density
seg	optional path to existing segmentation, if missing then will segment abu using DNACopy with standard settings
cfield	character, junction confidence meta data field in ra
tfield	character, tier confidence meta data field in ra
outdir	out directory to dump into, default ./
nseg	optional path to normal seg file with $cn meta data field
hets	optional path to hets.file which is tab delimited text file with fields seqnames, start, end, alt.count.t, ref.count.t, alt.count.n, ref.count.n
name	prefix for sample name to be output to seg file
cores	number of cores to use (default 1)
field	field of coverage GRanges to use as fragment density signal (only relevant if coverage is GRanges rds file)
subsample	numeric between 0 and 1 specifying how much to sub-sample high confidence coverage data
tilim	timeout for jbaMIP computation (default 1200 seconds)
init	jabba object (list) or path to .rds file containing previous jabba object which to use to initialize solution, this object needs to have the identical aberrant junctions as the current jabba object (but may have different segments and loose ends, i.e. is from a previous iteration)
edgenudge	numeric hyper-parameter of how much to nudge or reward aberrant junction incorporation, default 0.1 (should be several orders of magnitude lower than average 1/sd on individual segments), a nonzero value encourages incorporation of perfectly balanced rearrangements which would be equivalently optimal with 0 copies or more copies.
slack.penalty	penalty to put on every loose.end copy, should be calibrated with respect to 1/(k*sd)^2 for each segment, i.e. that we are comfortable with junction balance constraints introducing k copy number deviation from a segments MLE copy number assignment (the assignment in the absence of junction balance constraints)
overwrite	flag whether to overwrite existing output directory contents or just continue with existing files.
nseg	path to data.frame or GRanges rds of normal seg file with coordinates and $cn data field specifying germline integer copy number

jbaMIP *jbaMIP*

Description

jbaMIP

Usage

```
jbaMIP(adj, segstats, beta = NA, gamma = NA, field.ncn = "ncn",
  tilim = 20, mipemphasis = 0, epgap = 0.01, ploidy.min = 0.1,
  ploidy.max = 20, beta.guess = beta, beta.min = beta.guess,
  beta.max = beta.guess, gamma.guess = NA, gamma.min = gamma.guess,
  gamma.max = gamma.guess, cn.sd = 1, cn.prior = cn.sd, partition = T,
  purity.prior.mean = NA, purity.prior.sd = 0.3,
  purity.prior.strength = 1, purity.prior = c(purity.prior.mean,
  purity.prior.sd), cn.fix = rep(NA, length(segstats)), cn.lb = cn.fix,
  cn.ub = cn.fix, loose.ends = c(), adj.lb = 0 * adj, adj.nudge = 0 *
  adj, na.node.nudge = TRUE, ecn.out.ub = rep(NA, length(segstats)),
  ecn.in.ub = rep(NA, length(segstats)), gurobi = F, nsolutions = 1,
  verbose = F, debug = F, mc.cores = 1, ignore.edge = FALSE,
  ignore.cons = TRUE, edge.slack = TRUE, slack.prior = 1, ...)
```

Arguments

adj	n x n adjacency matrix interpreted as binary (this is the $adj output of karyograph)
segstats	n x 1 GRanges object with "mean" and "sd" value fields
beta	numeric guess for beta (i.e. from ppgrid)
gamma	numeric guess for gamma (i.e. from ppgrid)
field.ncn	this field takes into account normal copy number in relative to absolute conversion
adj.lb	nxn matrix of lower bounds on particular copy numbers - this is used to force certain junctions into the graph
adj.nudge	nxn adjacency matrix of "nudge" rewards on individual junctions, NOTE: maximum value in this matrix
slack.prior	1/slack.prior = penalty for each additional copy number of each slack edge, the higher slack.prior the more slack we allow in the reconstruction, should be intuitively calibrated to the expected "incompleteness" of the reconstruction, 1/slack.prior should be calibrated with respect to 1/(k*sd)^2 for each segment, so that we are comfortable with junction balance constraints introducing k copy number deviation from a segments MLE copy number assignment (the assignment in the absence of junction balance constraints)

Details

primary "heavy" lifting task of JaBbA. Sets up optimization problem given an input graph and segstats input and sends to CPLEX via RCplex

combines edge-conservation constraints from karyograph (n x n adjacency matrix connecting n genomic intervals) with segment abundance data (segstats - length n GRanges object with mean and sd fields corresponding to posterior means and sd's on the the relative "concentration" of each interval) to infer

(1) interval and edge absolute copy numbers on the karyograph (2) purity and ploidy (3) slack edges (if any needed)

basically solves ABSOLUTE problem (fitting integer grid to continuous segment intensities) while enforcing edge-conservation constraints.

Most important optional parameters include (1) cn.sd (expected deviation of absolute copy number from ploidy) (2) ploidy.min and ploidy.max –> useful for probing alternate solutions, but can be generously set (3) adj.lb –> enforces minimal edge absolute copy number, eg to force aberrant adjacency use (4) edge.slack - logical variable to determine whether or not to allow penalized relaxation of edge conservation constraints (5) nsolutions - number of alternate solutions

Value

output is a Rcplex solution or list of Rcplex solution with additional fields, each Rcplex solution is a list and the additional fields added by jbaMIP are $adj input n x n adjacency matrix populated with integer copy numbers $segstats input segstats vector populated with meta data fields $cn, $ecn.in, $ecn.out, $edges.out, $eslack.in, $eslack.out $purity purity value associated with relative-absolute affine copy number conversion for this solution $ploidy purity value associated with relative-absolute affine copy number conversion for this solution $

Additional fields for qc / technical debugging: $nll.cn negative log likelihood corresponding to the CN fit in this solution $nll.opt negative log likelihood correpsonding to the MLE CN without junction constraints $residual = value of residual between copy solution and MLE fit without junction constraints $beta beta value associated with relative-absolute affine copy number conversion for this solution $gamma gamma value associated with relative-absolute affine copy number conversion for this solution $gap.cn total gap between MLE fit without junction constraints and JaBbA fit $ploidy.constraints input ploidy constraints $beta.constraints input beta constrinats $cn.prior input cn.prior $slack.prior input slack.prior mimielinski Sunday, Dec 31, 2017 03:58:45 PM

`jbaMIP.allelic` *jbaMIP.allelic*

Description

Takes adj and segstats from output from jbaMIP and a granges of het.sites with $ref.count and $alt.count

Usage

```
jbaMIP.allelic(adj, segstats, purity, gamma, partition = T,
  slack.prior = 0.001)
```

Arguments

`adj`	adjacency matrix populated with total copy counts on junctions
`segstats`	granges tiling genome populated with total copy counts on interval, mu_high, sd_high, mu_low, sd_low variables on alleles

purity	purity from solution
gamma	gamma param from jbaMIP
slack.prior	1/slack.prior = penalty for each slack i.e. loose end copy in solution

Details

assumes segstats has fields $cn populated and adj has copy states

Value

list with fields $segstats = input segstats annotated with fitted cn.high, cn.low columns $asegstats = output "allelic" segstats, with $cn, $parent.node, $eslack.in, $eslack.out, $phased fields filled in $adj = output length(segstats) x length(segstats) x 2 "allelic" adjacency matrix with inferred allelic copy numbers on edges $aadj = flattened output 2*length(segstats) x 2* length(segstats) "allelic" adjacency matrix with inferred allelic copy numbers on edges

---

jbaMIP.mipstart *jbaMIP*

Description

jbaMIP

Usage

```
jbaMIP.mipstart(adj, segstats, beta = NA, gamma = NA, field.ncn = "ncn",
  tilim = 20, mipemphasis = 0, epgap = 0.01, ploidy.min = 0.1,
  ploidy.max = 20, beta.guess = beta, beta.min = beta.guess,
  beta.max = beta.guess, gamma.guess = NA, gamma.min = gamma.guess,
  gamma.max = gamma.guess, cn.sd = 1, cn.prior = cn.sd, partition = T,
  purity.prior.mean = NA, purity.prior.sd = 0.3,
  purity.prior.strength = 1, purity.prior = c(purity.prior.mean,
  purity.prior.sd), cn.fix = rep(NA, length(segstats)), cn.lb = cn.fix,
  cn.ub = cn.fix, loose.ends = c(), adj.lb = 0 * adj, adj.nudge = 0 *
  adj, na.node.nudge = TRUE, ecn.out.ub = rep(NA, length(segstats)),
  ecn.in.ub = rep(NA, length(segstats)), gurobi = F, nsolutions = 1,
  verbose = F, debug = F, mc.cores = 1, ignore.edge = FALSE,
  ignore.cons = TRUE, edge.slack = TRUE, slack.prior = 1, ...)
```

Arguments

adj	n x n adjacency matrix interpreted as binary (this is the $adj output of karyograph)
segstats	n x 1 GRanges object with "mean" and "sd" value fields
beta	numeric guess for beta (i.e. from ppgrid)
gamma	numeric guess for gamma (i.e. from ppgrid)
field.ncn	this field takes into account normal copy number in relative to absolute conversion
adj.lb	nxn matrix of lower bounds on particular copy numbers - this is used to force certain junctions into the graph

`adj.nudge` nxn adjacency matrix of "nudge" rewards on individual junctions, NOTE: maximum value in this matrix

`slack.prior` 1/slack.prior = penalty for each additional copy number of each slack edge, the higher slack.prior the more slack we allow in the reconstruction, should be intuitively calibrated to the expected "incompleteness" of the reconstruction, 1/slack.prior should be calibrated with respect to 1/(k*sd)^2 for each segment, so that we are comfortable with junction balance constraints introducing k copy number deviation from a segments MLE copy number assignment (the assignment in the absence of junction balance constraints)

Details

primary "heavy" lifting task of JaBbA. Sets up optimization problem given an input graph and segstats input and sends to CPLEX via RCplex

combines edge-conservation constraints from karyograph (n x n adjacency matrix connecting n genomic intervals) with segment abundance data (segstats - length n GRanges object with mean and sd fields corresponding to posterior means and sd's on the the relative "concentration" of each interval) to infer

(1) interval and edge absolute copy numbers on the karyograph (2) purity and ploidy (3) slack edges (if any needed)

basically solves ABSOLUTE problem (fitting integer grid to continuous segment intensities) while enforcing edge-conservation constraints.

Most important optional parameters include (1) cn.sd (expected deviation of absolute copy number from ploidy) (2) ploidy.min and ploidy.max –> useful for probing alternate solutions, but can be generously set (3) adj.lb –> enforces minimal edge absolute copy number, eg to force aberrant adjacency use (4) edge.slack - logical variable to determine whether or not to allow penalized relaxation of edge conservation constraints (5) nsolutions - number of alternate solutions

Value

output is a Rcplex solution or list of Rcplex solution with additional fields, each Rcplex solution is a list and the additional fields added by jbaMIP are $adj input n x n adjacency matrix populated with integer copy numbers $segstats input segstats vector populated with meta data fields $cn, $ecn.in, $ecn.out, $edges.out, $eslack.in, $eslack.out $purity purity value associated with relative-absolute affine copy number conversion for this solution $ploidy purity value associated with relative-absolute affine copy number conversion for this solution $

Additional fields for qc / technical debugging: $nll.cn negative log likelihood corresponding to the CN fit in this solution $nll.opt negative log likelihood correpsonding to the MLE CN without junction constraints $residual = value of residual between copy solution and MLE fit without junction constraints $beta beta value associated with relative-absolute affine copy number conversion for this solution $gamma gamma value associated with relative-absolute affine copy number conversion for this solution $gap.cn total gap between MLE fit without junction constraints and JaBbA fit $ploidy.constraints input ploidy constraints $beta.constraints input beta constrinats $cn.prior input cn.prior $slack.prior input slack.prior

`jbaMIP.process` *jbaMIP.process*

Description

process jbaMIP solution "sol" given original graph "g" (karyograph() list output) into JaBbA object

Usage

```
jbaMIP.process(sol, allelic = F)
```

Arguments

`sol`	JaBbA object
`allelic`	logical flag specifying whether object is allelic

Details

output is

Value

list with items: $B incidence matrix of augmented graph (including slack vertices) (vertices x edges) rownames of $B are vertex names of $G and colnames of B are named with character version of their $G indices (i.e. column order of B respects the original edge order in the solution)

$e edge constraints for downstream karyoMIP, i.e the copy numbers at the edges $e.ij numedges x 2 vertex pair matrix denoting what are the vertex pairs corresponding to the cols of $B and entries of $e, $eclass, $etype etc $eclass id for each unique edge / anti-edge equivalence class $etype specifies whether edge is slack or nonslack

`jbaMIP.summarize` *jbaMIP.summarize*

Description

summarizes miqp result (i.e. multiple solutions) outputting data frame with summary info

Usage

```
jbaMIP.summarize(sol)
```

jgraph *jgraph*

Description

takes in a jabba object and threshold for clusters and "quasi-reciprocal" junctions

Usage

```
jgraph(jab, thresh_cl = 1e+06, all = FALSE, thresh_r = 1000,
  clusters = FALSE)
```

junction.paths *junction.paths*

Description

Applies "pigeonhole principle" to enumerate all junction paths in a karyograph that can be proven to have copy number greater than 0

Usage

```
junction.paths(cn, adj)
```

Arguments

cn	length n vector of integer copy numbers
adj	nxn matrix of junction copy numbers

Details

Takes as input adjacency matrix specifying junction copy numbers and numeric vector specifying node copy numbers.

Value

list with fields $paths list of n paths, each path i consisting of an n_i x 2 matrix specifying sequences of n_i junctions (each an ij node pair) $mcn minimum copy number associated with path i

karyograph *karyograph*

Description

karyograph

Usage

```
karyograph(junctions, tile = NULL, label.edges = FALSE)
```

Arguments

junctions	GRangesList of junctions, where each item is a length GRanges of signed locations
tile	GRanges optional existing tiling of the genome (eg a copy number segmentation) from which additional segments will be created

Details

builds graph from rearrangement breakpoints +/- copy number endpoints used for downstream jbaMIP and karyoMIP functions

Input bpp is a GRangesList of signed locus pairs describing aberrant adjacencies. The convention is as follows: Each locus in the input breakpoint pair points to the direction that is being joined by the adjacencies i.e. (-) bp points to "left" or preceding segment (+) bp points to the "right" or the following segment

eg imagine a|bp1|b c|bp2|d "+" bp point to the right (eg b or d), "-" bp point to the left (a or c)

Input "tile" is a set of intervals whose endpoints are also used to partition the genome prior to the building of the karyograph.

Output karyograph connects signed genomic intervals (in a signed tiling of the reference genome) with "aberrant" and "reference" edges. Reference edges connect intervals that are adjacent in the reference genome, and aberrant edges are inferred (upstream of this) through cancer genome paired end analysis. Note that every node, edge, and path in this karyograph has a "reciprocal path"

Value

a list with the following fields $tile = GRanges of length 2*n tiling of the genome corresponding to union of rearrangement breakpoints and copy number endpoints $G = igraph object representing karyograph, here are the edge and vertex features vertex features: $chrom, $start, $end, $width, $strand, $size, $shape, $border.width, $label, $chrom.ord, $y, $col, $weight edge features:$ $bp.id, $weight, $from, $to, $col, $type, $line.style, $arrow.shape, $width, important: $type specifies which edges are "aberrant" and "reference", $bp.id specifies which input rearrangement (item in junctions) a given aberrant edge came from (and is NA for reference edges) $adj = 2n x 2n adjacency matrix whose nonzero entries ij show the edge.id in $G $ab.adj = 2n x 2n binary matrix specifying aberrant edges $ab.edges = length(junctions) x 'from', 'to' x '+', '-' mapping junction id's (indices into input junctions lists) to source and sink vertices, in both orientations mimielinski Sunday, Aug 06, 2017 06:46:15 PM fix added to handle strange [0 0] junctions outputted by Snowman ... which failed to match to any tile todo: may want to also handle junctions that fall off the other side of the chromosome

karyograph_stub *karyograph*

Description

karyograph

Usage

```
karyograph_stub(seg.file, cov.file, nseg.file = NULL, het.file = NULL,
  ra = NULL, dranger.file = NULL, out.file, ra.file = NULL,
  force.seqlengths = NULL, purity = NA, ploidy = NA, field = "ratio",
  mc.cores = 1, max.chunk = 1e+08, subsample = NULL)
```

karyoMIP *karyoMIP*

Description

MIP to locally compute walks in an existing JaBbA reconstruction, note: usually many optimal solutions to a given run. Used by jabba.walk.

Usage

```
karyoMIP(K, e, eclass = 1:length(e), kclass = NULL, prior = rep(0,
  ncol(K)), mprior = NULL, cpenalty = 1, tilim = 100, epgap = 1,
  nsolutions = 50, objsense = "max", ...)
```

Arguments

K	|E| x k binary matrix of k "extreme" contigs across |E| edges
e	edge copy numbers across |E| edges
eclass	edge equivalence classes, used to constrain strand flipped contigs to appear in solutions together, each class can have at most 2 members
prior	prior log likelihood of a given contig being in the karyotype
cpenalty	karyotype complexity penalty - log likelihood penalty given to having a novel contig in the karyotype, should be calibrated to prior, i.e. higher than the contig-contig variance in the prior, otherwise complex karyotypes may be favored
tilim	time limit to optimizatoin
nsolutions	how many equivalent solutions to report

Details

TODO: Make user friendly, still pretty raw

takes |E| x k matrix of k extreme paths (i.e. contigs) across e edges of the karyograph and length |E| vector of edge copy numbers (eclass), length |E| vector of edge equivalence classes (both outputs of jbaMIP.process) and computes most likely karyotypes that fit the edge copy number profile subject to some prior likelihood over the k extreme paths

Value

Rcplex solution list object with additional field $kcn for path copy number, $kclass for k class id, $mval for mval

---

`karyoMIP.to.path` *karyoMIP.to.path*

Description

for a karyoMIP solution and associated K matrix of n x e elementary paths (input to karyoMIP), and v x e edge signed incidence matrix

Usage

```
karyoMIP.to.path(sol, K, e, gr = NULL, mc.cores = 1, verbose = T)
```

Arguments

`sol`	solution to karyoMIP
`K`	matrix of elementary paths (input to karyoMIP)
`e`	nrow(K) x 2 edge matrix representing vertex pairs (i.e. edges to which K is referring to)
`gr`	optional GRanges whose names are indexed by rownames of B
`mc.cores`	integer number of cores
`verbose`	flag

Value

A list with following items: $path length k list of paths, cycles (each item i is vector denoting sequence of vertices in G) $is.cycle length k logical vector whose component i denotes whether path i is cyclic $cn length k integer vector whose component i denotes copy number of contig i $path.grl if path.grl == T

`karyotrack` *karyo track*

Usage

```
karyotrack(kag, paths = NULL, col = "red", pad = 0)
```

Arguments

`kag`	output of karyograph
`paths`	GRanges or GRangesList

Details

Takes karyograph and outputs gTrack +/- highlighting of one or more paths defined as GRanges or GRangesList (for multiple paths) Edges will only be highlighted when the exact interval pair corresponding to the edge is included in the graph

Value

gTrack of karyograph with particular nodes / edges colored with specified colors

---

loose.ends *loose.ends*

Description

takes jbaMIP output and outputs a vector of ranges on the right or left end of the intervals that have type 1-4 labels where

Usage

```
loose.ends(sol, kag)
```

Arguments

sol	JaBbA object
kag	karyograph object

Details

type1 = cn drop, no junction on this side, slack type2 = cn drop, no junction on this side slack type3 = no cn diff, used junction on other side, slack type4 = no cn diff, unused junction on other side, no slack

Value

vector of ranges on the right or left end of the intervals that have type 1-4 labels where

---

mmatch *mmatch*

Description

Low level utility function to match rows of matrix A to matrix B

Usage

```
mmatch(A, B, dir = 1)
```

`munlist` *munlist*

Description

unlists a list of vectors, matrices, data frames into a n x k matrix whose first column specifies the list item index of the entry and second column specifies the sublist item index of the entry and the remaining columns specifies the value(s) of the vector or matrices.

Usage

```
munlist(x, force.rbind = F, force.cbind = F, force.list = F)
```

Details

force.cbind = T will force concatenation via 'cbind' force.rbind = T will force concatenation via 'rbind'

`pp.nll` *pp.nll*

Description

computes neg log likelihood ($nll) for purity, ploidy combo and mle abs copy numbers ($v), returns as list

Usage

```
pp.nll(segstats, purity = NA, ploidy = NA, gamma = NA, beta = NA,
  field = "mean", field.ncn = "ncn", v = NULL)
```

Arguments

`segstats`	granges of segstats with field $mean and $sd corresponding to mean and standard deviation on estimates of fragment density
`purity`	numeric purity value
`ploidy`	numeric ploidy value
`gamma`	numeric gamma value (over-rides purity, ploidy)
`beta`	numeric beta value (over-rides purity, ploidy)
`field.ncn`	character specifying meta data field specifying germline copy number, default value is "ncn", if absent then will assume 2
`solution`	over-ride if not interested in MLE / weighted least squares fit, this should be an integer vector of length(segstats)

Details

v can be over-ridden to compute NLL for other (eg non MLE) values

Value

negative log likelihood of MLE i.e. least squares model (or supplied solution) fit

`ppgrid` *ppgrid*

Description

least squares grid search for purity and ploidy modes

Usage

```
ppgrid(segstats, allelic = FALSE, purity.min = 0.01, purity.max = 1,
  ploidy.step = 0.01, purity.step = 0.01, ploidy.min = 1.2,
  ploidy.max = 6, plot = F, verbose = F, mc.cores = 10)
```

Arguments

`segstats`	GRanges object of intervals with meta data fields "mean" and "sd" (i.e. output of segstats function)
`allelic`	logical flag, if TRUE will also look for mean_high, sd_high, mean_low, sd_low variables and choose among top solutions from top copy number according to the best allelic fit
`mc.cores`	integer number of cores to use (default 1)

Value

data.frame with top purity and ploidy solutions and associated gamma and beta values, for use in downstream jbaMI

`proximity` *proximity*

Description

Takes a set of n "query" elements (GRanges object, e.g. genes) and determines their proximity to m "subject" elements (GRanges object, e.g. regulatory elements) subject to set of rearrangement adjacencies (GRangesList with width 1 range pairs)

Usage

```
proximity(query, subject, ra = GRangesList(), jab = NULL, verbose = F,
  mc.cores = 1, max.dist = 1e+06)
```

Arguments

`query`	GRanges of "intervals of interest" eg regulatory elements
`subject`	GRanges of "intervals of interest" eg genes
`ra`	GRangesList of junctions (each a length 2 GRanges, similar to input to karyograph)
`jab`	existing JaBbA object (overrides ra input)

verbose	logical flag
mc.cores	how many cores (default 1)
max.dist	maximum genomic distance to store and compute (1MB by default) should the maximum distance at which biological interactions may occur

Details

This analysis makes the (pretty liberal) assumption that all pairs of adjacencies that can be linked on a karyograph path are in cis (i.e. share a chromosome) in the tumor genome.

Value

list of n x m sparse distance matrices: $ra = subject-query distance in the rearranged genome for all loci < max.dist in tumor genome $wt = subject-query distance in the reference genome for all loci < max.dist in tumor genome $rel = subject-query distance in ra relative to wild type for above loci NOTE: values x_ij in these matrices should be interpreted with a 1e-9 offset to yield the actual value y_ij i.e. y_ij = x_ij-1e-9, x_ij>0, y_ij = NA otherwise (allows for sparse encoding of giant matrices)

---

ramip_stub *run ramip*

Description

run ramip

Usage

```
ramip_stub(kag.file, out.file, mc.cores = 1, mem = 16, tilim = 1200,
  slack.prior = 0.001, gamma = NA, beta = NA, customparams = T,
  purity.min = NA, purity.max = NA, ploidy.min = NA, ploidy.max = NA,
  init = NULL, edge.nudge = 0, ab.force = NULL)
```

---

ra_breaks *ra_breaks*

Description

takes in either file or data frame from dranger or snowman or path to BND / SV type vcf file and returns junctions in VCF format.

Usage

```
ra_breaks(rafile, keep.features = T, seqlengths = hg_seqlengths(),
  chr.convert = T, snowman = FALSE, swap.header = NULL,
  breakpointer = FALSE, seqlevels = NULL, force.bnd = FALSE, skip = NA,
  get.loose = FALSE)
```

Details

The default output is GRangesList each with a length two GRanges whose strands point AWAY from the break. If get.loose = TRUE (only relevant for VCF)

`ra_tier` *ra_tier (*

Description

Classify full set of dRanger rearrangements into "tiers" of confidence

(1) Tier 1 BPresult>0 and somatic_score>min.score1 (2) Tier 2 BPresult=0 and somatic_score>min.score1 (3) Tier 3 min.score2<=somatic_score<=min.score2 & tumreads>min.reads

Usage

```
ra_tier(dra, min.score1 = 10, min.score2 = 4, min.treads1 = 10,
  min.treads2 = 3, max.nreads = Inf)
```

`rel2abs` *rel2abs*

Description

rescales CN values from relative to "absolute" (i.e. per cancer cell copy) scale given purity and ploidy

Usage

```
rel2abs(gr, purity = NA, ploidy = NA, gamma = NA, beta = NA,
  field = "ratio", field.ncn = "ncn")
```

Arguments

`gr`	GRanges input with meta data field corresponding to mean relative copy "mean" in that interval
`purity`	purity of sample
`ploidy`	ploidy of sample
`gamma`	gamma fit of solution (over-rides purity and ploidy)
`beta`	beta fit of solution (over-rides purity and ploidy)
`field`	meta data field in "gr" variable from which to extract signal, default "mean"
`field.ncn`	meta data field in "gr" variable from which to extract germline integer copy number, default "ncn", if doesn't exist, germline copy number is assumed to be zero

Details

takes in gr with signal field "field"

Value

numeric vector of integer copy numbers

`segstats` *segstats is a step in the JaBbA pipeline*

Description

segstats is a step in the JaBbA pipeline

Usage

```
segstats(target, signal = NULL, field = "signal", asignal = NULL,
  afields = c("ref.count", "alt.count"), prior_weight = 1,
  prior_mean = NA, prior_alpha = NA, prior_beta = NA, max.chunk = 1e+08,
  max.slice = 20000, na.thresh = 0.2, subsample = NULL, mc.cores = 1,
  nsamp_prior = 1000, ksamp_prior = 100)
```

Arguments

`target`	GRanges of segments on which segstats will be computed
`signal`	GRanges of coverage from which samples will be taken
`field`	field of "signal" GRanges from which coverage signal will be pulled
`asignal`	optional GRanges corresponding width 1 bialellic snp allele counts across the genome,
`afields`	length 2 character vector meta data fields of asignal GRanges that will be used to get allele counts (default is ref.count, alt.count)
`subsample`	number between 0 and 1 with which to subsample per segment for coverage (useful for superdense coverage eg 50 bases to avoid correlations between samples due to read overlap)
`mc.cores`	number of cores to run on (default 1)

Details

computes posterior mean's and sd's for a target tiling GRanges of segments (target) target must be a non-overlapping gapless strandless or two stranded tiling of the genome (eg output of gr.tile) if two stranded, then every stranded interval must have a mirror image interval included

given a GRanges of signals using value field "field" of signal GRanges assuming that the signals inside each interval in "target" are independent samples from a normal distribution of unknown mean and variance

will also compute means ands std deviations on a gamma posterior of the the "high" and "low" alleles for a granges "asignal" representing allelic signal ref count and tot count across a set of locations (i.e. modeling the counts as a poisson random variable) Fields are specified by afields

outputs target GRanges with "$mean" and "$sd" fields populated

`sparse_subset` *sparse_subset*

Description

given k1 x n matrix A and k2 x n matrix B returns k1 x k2 matrix C whose entries ij = 1 if the set of nonzero components of row i of A is a (+/- strict) subset of the nonzero components of row j of B

Usage

```
sparse_subset(A, B, strict = FALSE, chunksize = 100, quiet = FALSE)
```

`vaggregate` *vaggregate*

Description

same as aggregate except returns named vector with names as first column of output and values as second

Note: there is no need to ever use aggregate or vaggregate, just switch to data.table

Usage

```
vaggregate(...)
```

Arguments

`...`	arguments to aggregate

Value

named vector indexed by levels of "by"

Author(s)

Marcin Imielinski

Index