Package ‘JaBbA’

February 14, 2018

Title Infer cancer graph genomes from read depth and junctions

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 (>= 3.3.0),
Matrix (>= 1.2),
gUtils (>= 0.2),
gGnome (>= 0.1),
gplots (>= 3.0),
GenomicRanges (>= 1.3),
optparse (>= 1.4.4),
gTrack (>= 0.1),
Ppurple(>= 0.2.0)

Imports VariantAnnotation (>= 1.24),
DNAcopy (>= 1.52),
data.table (>= 1.9),
igraph (>= 1.1),
methods,
parallel,
graphics,
grDevices,
stats,
utils

Suggests gurobi(>= 7.0),
testthat (>= 2.0)

License GPL-2

LazyData true

RoxygenNote 6.0.1

R topics documented:

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(junctions, coverage, seg = NULL, outdir = "./JaBbA", cfield = NULL,
  tfield = NULL, nudge.balanced = FALSE, thresh.balanced = 500,
  nseg = NULL, hets = NULL, name = "tumor", purity = NA, ploidy = NA,
  field = "ratio", subsample = NULL, tilim = 1200, mem = 16,
  reiterate = 0, rescue.window = 10000, init = NULL, edgenudge = 0.1,
  use.gurobi = FALSE, slack.penalty = 100, overwrite = FALSE,
  mc.cores = 1, strict = FALSE, max.threads = Inf, max.mem = 16,
  verbose = TRUE)
```

Arguments

`junctions`	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 GRanges-List
`coverage`	GRanges of coverage OR path to tsv of cov file w GRanges style columns, rds of GRanges or .wig / .bed file of (+/- normalized, GC corrected) fragment density
`seg`	optional path to existing segmentation, if missing then will segment coverage using DNACopy with standard settings
`outdir`	out directory to dump into, default ./
`cfield`	character, junction confidence meta data field in ra
`tfield`	character, tier confidence meta data field in ra

`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
`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 fraction with which 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.
`use.gurobi`	logical flag specifying whether to use gurobi (if TRUE) instead of CPLEX (if FALSE) .. up to user to make sure the respective package is already installed
`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.
`mc.cores`	integer how many cores to use to fork subgraphs generation (default = 1)
`strict`	logical flag specifying whether to only include junctions that exactly overlap segs
`cores`	number of cores to use (default 1)
`nseg`	path to data.frame or GRanges rds of normal seg file with coordinates and $cn data field specifying germline integer copy number

Value

gGraph (gGnome package) of balanced rearrangement graph

Examples

```
library(JaBbA)
junctions = system.file("extdata", "junctions.vcf", package = 'JaBbA')
coverage = system.file("extdata", "coverage.txt", package = 'JaBbA')
hets = system.file("extdata", "hets.txt", package = 'JaBbA')

## run analysis without hets
jab = JaBbA(junctions = junctions, coverage = coverage)

## run analysis with hets in different subdir (default ./JaBbA)
jab = JaBbA(junctions = junctions, coverage = coverage, hets = hets, outdir = './mydir')

## run analysis with "tiered" junctions, these have metadata field $tier (in this case ir
jun = read.junctions(junctions)

## these have metadata field tier, tier 1 junctions are forced to be included, tier 2 = a
## tier 3 are only used in later iterations to rescue loose ends
values(jun)$tier

## here we will iterate JaBbA max 4 times, or until we run out of tier three junctions ne
jab = JaBbA(junctions = jun, coverage = coverage, hets = hets, reiterate = 4, outdir = '.

## can increase the window to rescue more junctions within 100kb of loose ends,
## will overwrite original ./JaBbA outdir with overwrite = TRUE
jab = JaBbA(junctions = jun, coverage = coverage, hets = hets, reiterate = 4, rescue.winc
```

`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 karyograph

run ramip

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

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

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

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

Dumps JaBbA graph into json

read.junctions

takes in either file or data frame from various formats including BND VCF, bedpe, and others, and returns GRangesList of junctions and returns junctions in VCF format.

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)

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

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

wrapper around variantAnnotation reads VCF into granges or data.table format

Applies FUN locally to levels of x and returns vector of length() (eg can do a "local" order within levels)

Modification of Rcplex which takes in mipcontrol parameters.

Usage

```
jabba_stub(junctions, coverage, seg = NULL, cfield = NULL, tfield = NULL,
  nudge.balanced = FALSE, thresh.balanced = 500, outdir = "./",
  nseg = NULL, hets = NULL, name = "tumor", mc.cores = 1,
  max.threads = Inf, max.mem = 16, purity = NA, ploidy = NA,
  strict = FALSE, mipstart = NULL, field = "ratio", subsample = NULL,
  tilim = 1200, mem = 16, init = NULL, edgenudge = 0.1,
  slack.penalty = 100, use.gurobi = FALSE, overwrite = F,
  verbose = TRUE)

karyograph_stub(seg.file, cov.file, nseg.file = NULL, het.file = NULL,
  ra = NULL, junction.file = NULL, out.file, use.ppurple = TRUE,
  ra.file = NULL, verbose = FALSE, force.seqlengths = NULL, purity = NA,
  ploidy = NA, field = "ratio", mc.cores = 1, max.chunk = 1e+08,
  subsample = NULL)

.plot_ppfit(kag, xlim = c(-Inf, Inf))

ramip_stub(kag.file, out.file, mc.cores = 1, max.threads = Inf, 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, mipstart = NULL, use.gurobi = FALSE,
  verbose = FALSE, edge.nudge = 0, ab.force = NULL, ab.exclude = NULL)

jmessage(..., pre = "JaBbA")

.correct.slack(ra.sol)

vaggregate(...)

write.tab(x, ..., sep = "\t", quote = F, row.names = F)

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

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

anno.hop(walks)

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

read.junctions(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)

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

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

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

read_vcf(fn, gr = NULL, hg = "hg19", geno = NULL, swap.header = NULL,
  verbose = FALSE, add.path = FALSE, tmp.dir = "~/temp/.tmpvcf", ...)

levapply(x, by, FUN = "order")

Rcplex2(cvec, Amat, bvec, Qmat = NULL, lb = 0, ub = Inf,
  control = list(), objsense = c("min", "max"), sense = "L",
  vtype = NULL, n = 1)
```

Arguments

`junctions`	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 GRanges-List
`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 we will segment coverage 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
`mc.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)
`use.gurobi`	logical flag whether to use gurobi vs CPLEX
`overwrite`	flag whether to overwrite existing output directory contents or just continue with existing files.
`kag`	output of karyograph
`...`	arguments to aggregate
`x`	input vector of data
`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
`walks`	walks to annotate
`jab`	input jab object
`file`	output json file
`paths`	GRanges or GRangesList
`by`	length(x) vector of categorical labels
`FUN`	function that takes a length k vector and outputs a length k vector, used for processing each "level" of by
`nseg`	path to data.frame or GRanges rds of normal seg file with coordinates and $cn data field specifying germline integer copy number
`jab`	JaBbA object #

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

named vector indexed by levels of "by"

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

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#'

length(x) vector of outputs, the results of applying FUN to each "by" defined level of x

Author(s)

Marcin Imielinski

Marcin Imielinski

Marcin Imielinski

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

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

Marcin Imielinski

Marcin Imielinski

Index