Package 'UPDhmm'

June 5, 2026

Title Detecting Uniparental Disomy through NGS trio data

Version 1.9.0

BugReports https://github.com/martasevilla/UPDhmm/issues

Description Uniparental disomy (UPD) is a genetic condition where an individual inherits both copies of a chromosome or part of it from one parent, rather than one copy from each parent. This package contains a HMM for detecting UPDs through HTS (High Throughput Sequencing) data from trio assays. By analyzing the genotypes in the trio, the model infers a hidden state (normal, father isodisomy, mother isodisomy, father heterodisomy and mother heterodisomy).

biocViews Software, HiddenMarkovModel, Genetics

License MIT + file LICENSE

Encoding UTF-8

LazyData false

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports HMM, utils, VariantAnnotation, GenomicRanges, S4Vectors, IRanges, SummarizedExperiment, stats, BiocParallel, GenomeInfoDb

Suggests knitr, testthat (>= 2.1.0), BiocStyle, rmarkdown, markdown, karyoploteR, regioneR, dplyr, BiocManager

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

URL https://github.com/martasevilla/UPDhmm

Language en-US

git_url https://git.bioconductor.org/packages/UPDhmm

git_branch devel

git_last_commit deaeef8

git_last_commit_date 2026-04-28

Repository Bioconductor 3.24

Date/Publication 2026-06-04

Author Marta Sevilla [aut, cre] (ORCID:
<https://orcid.org/0009-0005-0179-920X>),
Carlos Ruiz-Arenas [aut] (ORCID:
<https://orcid.org/0000-0002-6014-3498>)

Maintainer Marta Sevilla <marta.sevilla@upf.edu>

Contents

`UPDhmm-package` *UPDhmm: Detecting Uniparental Disomy through NGS trio data*

Description

Uniparental disomy (UPD) is a genetic condition where an individual inherits both copies of a chromosome or part of it from one parent, rather than one copy from each parent. This package contains a HMM for detecting UPDs through HTS (High Throughput Sequencing) data from trio assays. By analyzing the genotypes in the trio, the model infers a hidden state (normal, father isodisomy, mother isodisomy, father heterodisomy and mother heterodisomy).

Author(s)

Maintainer: Marta Sevilla <marta.sevilla@upf.edu> (ORCID)

Authors:

- Carlos Ruiz-Arenas <cruizarenas@unav.es> (ORCID)

See Also

Useful links:

- https://github.com/martasevilla/UPDhmm
- Report bugs at https://github.com/martasevilla/UPDhmm/issues

`applyViterbi` *Apply the hidden Markov model using the Viterbi algorithm.*

Description

Apply the hidden Markov model using the Viterbi algorithm.

Usage

```
applyViterbi(largeCollapsedVcf, hmm)
```

Arguments

`largeCollapsedVcf`	input vcf file
`hmm`	Hidden Markov Model used to infer the events

Value

the input largeCollapsedVcf object updated with a new metadata column *states*, which contains the inferred hidden state for each variant.

`blocksVcf` *Collapse contiguous variants into genomic blocks based on HMM states*

Description

This function constructs block-level representations from a CollapsedVCF object. Consecutive variants sharing the same inferred HMM state are grouped into blocks. For each block, the function summarizes genomic coordinates, number of variants, HMM state, genotype codes and optionally computes per-block depth ratios normalized by per-sample mean read depth across the entire VCF.

Usage

```
blocksVcf(
  largeCollapsedVcf,
  add_ratios = FALSE,
  field_DP = NULL,
  total_mean = NULL,
  ratio_cols = c("ratio_father", "ratio_mother", "ratio_proband")
)
```

Arguments

largeCollapsedVcf	A CollapsedVCF object containing the metadata columns *states* (inferred HMM states) and *geno_coded* (numeric genotype codes). This object should be the result of first applying `vcfCheck()` and then `applyViterbi()`.
add_ratios	Logical; default = FALSE. If TRUE, computes normalized per-block read depth ratios for each individual based on total mean depth.
field_DP	Optional character string specifying which VCF FORMAT field to use for depth metrics (e.g., DP, AD, or a custom field).
total_mean	Optional numeric vector of per-sample mean read depths across the entire VCF, used to normalize per-block depth ratios computed via `computeTrioTotals()` in `calculateEvents()`.
ratio_cols	Character vector of column names to assign to the ratio output when total_mean is provided. Default: `c("ratio_proband", "ratio_mother", "ratio_father")`.

Value

A data.frame with one row per block, containing:

- `ID` – sample identifier
- `chromosome`, start, end – genomic coordinates
- `group` – HMM state of the block
- `n_snps` – number of variants in the block
- `geno_coded` – list of numeric genotype codes per block
- Ratio columns relative to total_mean (always present; if add_ratios = FALSE, filled with NA)

`calculateEvents` *Calculate UPD events in trio VCFs.*

Description

This function predicts the hidden states by applying the Viterbi algorithm using the Hidden Markov Model (HMM) from the UPDhmm package. It takes the genotypes of the trio as input and includes a final step to simplify the results into blocks.

Usage

```
calculateEvents(
  largeCollapsedVcf,
  hmm = NULL,
  field_DP = NULL,
  add_ratios = FALSE,
  BPPARAM = BiocParallel::SerialParam(),
  verbose = FALSE
)
```

Arguments

largeCollapsedVcf	The VCF file in the general format (largeCollapsedVcf) with VariantAnnotation package. Previously edited with `vcfCheck()` function from UPDhmm package.
hmm	Default = `NULL`. If no arguments are added, the package will use the default HMM already implemented, based on Mendelian inheritance.
field_DP	Default = `NULL`. Character string specifying which FORMAT field in the VCF contains the read depth information to use in `addRatioDepth()`. If `NULL` (default), the function will automatically try `"DP"` (standard depth) or `"AD"` (allelic depths, summed across alleles). Use this parameter if your VCF uses a non-standard field name for depth, e.g. `field = "NR"` or `"field_DP"`.
add_ratios	Logical; default = FALSE. If TRUE, per-sample mean depth is computed across the entire VCF and used to calculate normalized per-block depth ratios.
BPPARAM	Parallelization settings, passed to bplapply. By default `BiocParallel::SerialParam()` (serial execution). To enable parallelization, provide a BiocParallel backend, e.g. `BiocParallel::MulticoreParam(workers = min(2, parallel::detectCores()))` or `BiocParallel::SnowParam(workers = 2)`. Note: when running under R CMD check or Bioconductor build systems, the number of workers may be automatically limited (usually less or equal to 2).
verbose	Logical, default = `FALSE`. If `TRUE`, progress messages will be printed during processing.

Details

Custom HMM structure. The user can implement its own HMM.:

A custom HMM must be a list following the structure of the HMM package, containing:

- States – character vector of hidden state names
- Symbols – vector of allowed observation symbols (genotype codes)
- startProbs – named vector of initial state probabilities
- transProbs – state transition probability matrix
- emissionProbs – matrix of emission probabilities for each state × symbol

Value

A `data.frame` object containing all detected events in the provided trio. Columns include:

- chromosome – chromosome name
- start, end – genomic coordinates
- group – inferred HMM state
- n_snps – number of SNPs in the block
- n_mendelian_error – number of Mendelian errors in the block
- depth-ratio metrics (always present; if add_ratios = FALSE, filled with NA)

If no events are found, the function will return an empty `data.frame`.

Examples

```
file <- system.file(package = "UPDhmm", "extdata", "test_het_mat.vcf.gz")
vcf <- VariantAnnotation::readVcf(file)
processedVcf <- vcfCheck(vcf,
    proband = "NA19675",
    mother = "NA19678",
    father = "NA19679"
)

# Run in serial mode (default)
res <- calculateEvents(processedVcf)

# Run in parallel with 2 cores
library(BiocParallel)
param <- MulticoreParam(workers = 2)
res_parallel <- calculateEvents(processedVcf, BPPARAM = param)
```

`collapseEvents` *Collapse events per sample and chromosome*

Description

This function collapses genomic events per individual, chromosome, and group, summarising the number of events, total Mendelian errors, the total span size, and a string listing all merged event coordinates.

Usage

```
collapseEvents(subset_df, min_ME = 2, min_size = 5e+05)
```

Arguments

`subset_df`	A data.frame containing event-level data with columns: • ID – Sample identifier. • chromosome – Chromosome name. • start – Start position of the event. • end – End position of the event. • group – Event group/class. • n_snps – Number of SNPs in the event. • n_mendelian_error – Number of Mendelian errors in the event. • ratio_proband, ratio_mother, ratio_father – depth-ratio metrics.
`min_ME`	Minimum number of Mendelian errors required to retain an event before collapsing (default: 2).
`min_size`	Minimum genomic span size required to retain an event before collapsing, in base pairs (default: 500e3).

Details

When values are available in the ratio columns (`ratio_proband`, `ratio_mother`, `ratio_father`), weighted mean ratios are computed across the collapsed events. The weighted mean ratio is calculated as:

$$\frac{\sum_i r_i \times N_i}{\sum_i N_i}$$

where r_i is the ratio of each individual event and N_i the number of SNPs in that event. If all values in a ratio column are `NA`, the corresponding collapsed value will be `NA_real_`.

Value

A data.frame with collapsed events and columns:

- ID – Sample identifier.
- chromosome – Chromosome name.
- start, end – Genomic span of the collapsed block.
- group – HMM state of the block.
- n_events – Number of events collapsed.
- total_mendelian_error – Sum of Mendelian errors across events.
- total_size – Total genomic span size covered by events.
- total_snps – Total SNPs in the overlapping events.
- prop_covered – Proportion of the region covered by events.
- ratio_proband, ratio_mother, ratio_father – Weighted mean ratios across the collapsed events. If all values are NA, the column will contain NA_real_.
- collapsed_events – Comma-separated list of collapsed events.

Examples

```
all_events <- data.frame(
ID = c("S1", "S1", "S1", "S2", "S2"),
chromosome = c("1", "1", "1", "2", "2"),
start = c(100e4, 200e4, 300e4, 500e4, 600e4),
end   = c(160e4, 260e4, 360e4, 560e4, 700e4),
group = c("iso_mat", "iso_mat", "het_pat", "iso_mat", "iso_mat"),
n_mendelian_error = c(5, 10, 2, 50, 30),
stringsAsFactors = FALSE
)
out <- collapseEvents(all_events)
```

computeTrioTotals *Compute per-sample total mean read depth for a trio*

Description

This internal helper function calculates the per-sample total mean read depth across a VCF for a trio, optionally using a specified FORMAT field. The resulting totals are used to normalize per-block depth ratios in downstream analyses.

Usage

```
computeTrioTotals(
  vcf,
  expected_samples = c("father", "mother", "proband"),
  field_DP = NULL
)
```

Arguments

vcf	A CollapsedVCF object containing the trio genotype data.
expected_samples	Character vector of length 3 specifying the column order of the trio: proband, mother, father. Default = c("proband","mother","father").
field_DP	Optional character string specifying the FORMAT field in the VCF to use for depth calculations.

Details

The function selects the depth or coverage field to use, giving priority to field_DP if specified and present in the VCF, followed by DP (standard depth) and then AD (allelic depth) if available. If AD is used, the depth for each variant is calculated as the sum across all alleles per sample. NA values are ignored when computing the per-sample mean depth.

Value

Numeric vector of per-sample mean read depths, named according to expected_samples. Returns NULL if no valid depth field is found.

hmm *HMM data for predicting UPD events in trio genomic data*

Description

This dataset provides Hidden Markov Model (HMM) parameters for predicting uniparental disomy (UPD) events in trio genomic data.

states Five different possible states.

symbols Code symbols used for genotype combinations.

startProbs The initial probabilities of each state.

transProbs Probabilities of transitioning from one state to another.

emissionProbs Given a certain genotype combination, the odds of each possible state.

Usage

```
data(hmm)
```

Format

A list with 5 different elements

Source

Created in-house based on basic Mendelian rules for calculating UPD events.

Examples

```
data(hmm)
```

---

identifyRecurrentRegions

Identify recurrent genomic regions across samples

Description

This function identifies recurrent genomic regions supported by multiple samples based on overlapping genomic intervals. Intervals are first filtered by a Mendelian error threshold. Events are then clustered using an agglomerative hierarchical approach within each chromosome, with the distance between events defined based on their overlap. Clusters supported by a minimum number of distinct samples are retained and collapsed into recurrent regions summarizing all contributing events.

Usage

```
identifyRecurrentRegions(
  df,
  ID_col = "ID",
  error_threshold = 100,
  min_support = 3,
  max_dist = 0.3,
  linkage = "complete"
)
```

Arguments

df	A data.frame with columns: • ID: Sample identifier. • chromosome: Chromosome name. • start: Start coordinate of the region. • end: End coordinate of the region. • n_mendelian_error or total_mendelian_error: Number of Mendelian errors in the region.
ID_col	Character string indicating the column name containing sample identifiers. Default is "ID".

`error_threshold`
: Numeric, default = 100. Maximum number of Mendelian errors allowed for a region to be considered.

`min_support`
: Integer, default = 3. Minimum number of unique samples required to call a region recurrent.

`max_dist`
: Numeric, default = 0.3. Maximum distance allowed to group events in the same cluster when cutting the hierarchical clustering dendrogram.

`linkage`
: Character string specifying the linkage method for the agglomerative hierarchical clustering. Default `"complete"`, where the distance between two clusters is defined as the largest distance between all possible pairs of events belonging to the two clusters.

Details

The function operates chromosome-wise and proceeds in several steps. First, events are filtered based on a maximum Mendelian error threshold. Then, a pairwise distance matrix is computed for overlapping events, where the distance between two events e1 and e2 is defined as:

$$1 - (overlap\_width / max(width(e1), width(e2)))$$

Event pairs that do not overlap are not explicitly evaluated and are directly assigned a distance of 1. In addition, only the upper triangular part of the distance matrix is computed, exploiting its symmetry to avoid redundant calculations and reduce computational cost.

Agglomerative hierarchical clustering is applied to this distance matrix. At each step, the most similar clusters are merged according to the specified linkage method (`linkage`), with the default `"complete"` linkage using the maximum distance between all possible pairs of events from the two clusters. The resulting dendrogram is cut at a height defined by `max_dist`, which represents the maximum distance allowed for events to be grouped in the same cluster. Smaller values enforce stricter overlap similarity.

Only clusters supported by at least `min_support` distinct samples are retained. Each valid cluster is then collapsed into a single recurrent region whose coordinates span all contributing events, and the original events are stored as `supporting_events` with coverage metrics.

These steps are performed internally by a helper function `identifyRecurrentRegionsByChr`. Results from all chromosomes are then combined into a single GRanges object returned by this function.

Value

A GRanges object containing the recurrent regions that meet the minimum support threshold. Metadata columns include:

- `n_samples` – number of distinct samples supporting the region.
- `supporting_events` – a GRangesList with the individual events that compose the recurrent region.

The `supporting_events` correspond exactly to the original events that were grouped into the cluster and define the boundaries of the resulting recurrent region.

If no clusters meet the filtering criteria, the function returns `NULL`.

Examples

```
df <- data.frame(
  ID = c("S1", "S2", "S3", "S4"),
  chromosome = c("chr1", "chr1", "chr1", "chr1"),
  start = c(100, 120, 500, 510),
  end = c(150, 170, 550, 560),
  n_mendelian_error = c(10, 20, 5, 5)
)
identifyRecurrentRegions(df, ID_col = "ID", error_threshold = 50, min_support = 2)
```

---

markRecurrentRegions *Annotate regions as recurrent or non-recurrent*

Description

Given a results data.frame and a set of recurrent genomic regions, this function labels each row as "Yes" (recurrent) or "No" (non-recurrent) based on overlaps with a set of recurrent regions.

Usage

```
markRecurrentRegions(df, recurrent_regions, min_overlap = 0.7)
```

Arguments

df	Data.frame with region coordinates and sample IDs.
recurrent_regions	GRanges object from identifyRecurrentRegions().
min_overlap	Numeric between 0 and 1, default = 0.7. Minimum fraction of the input region that must overlap a recurrent region to be annotated as recurrent.

Details

A region is marked as recurrent if the fraction of its length overlapping a recurrent region is at least `min_overlap` (default 0.7).

Value

The same data.frame with two added columns:

- Recurrent: "Yes" or "No"
- n_samples: Number of supporting samples (if recurrent)

Examples

```
input <- data.frame(
    ID = c("S1", "S2", "S3", "S4"),
    chromosome = c("chr1", "chr1", "chr1", "chr2"),
    start = c(100, 150, 500, 100),
    end = c(150, 200, 550, 150),
    n_mendelian_error = c(10, 20, 5, 200)
)
```

```
recurrent_gr <- GenomicRanges::GRanges(
 seqnames = "chr1",
 ranges = IRanges::IRanges(
   start = 100,
   end = 170
 ),
 n_samples = 2
)
markRecurrentRegions(input, recurrent_gr)
```

`processChromosome` *Process a single chromosome for UPD detection*

Description

Internal helper function to run the full pipeline on one chromosome:

- applyViterbi
- blocksVcf

Usage

```
processChromosome(
  vcf_chr,
  hmm,
  add_ratios = FALSE,
  field_DP = NULL,
  total_mean = NULL,
  mendelian_error_values
)
```

Arguments

`vcf_chr`	CollapsedVCF object for one chromosome
`hmm`	Hidden Markov Model object
`add_ratios`	Logical; default = FALSE.
`field_DP`	Default = `NULL`. Character string specifying which FORMAT field in the VCF contains the read depth information to use in `addRatioDepth()`. If `NULL` (default), the function will automatically try `"DP"` (standard depth) or `"AD"` (allelic depths, summed across alleles). Use this parameter if your VCF uses a non-standard field name for depth, e.g. `field = "NR"` or `"field_DP"`.
	If TRUE, computes normalized per-block read depth ratios for each individual based on total mean depth.
`total_mean`	Optional numeric vector of per-sample mean read depths across the entire VCF, used to normalize per-block depth ratios computed via `computeTrioTotals()` in `calculateEvents()`.
`mendelian_error_values`	Character vector of genotype codes considered Mendelian errors (i.e., observations with minimal emission probability in the "normal" state). Provided by `calculateEvents()`.

Value

A data.frame of detected blocks for the chromosome, or NULL if error Columns include:

- `chromosome` – chromosome name
- `start`, `end` – genomic coordinates of the block
- `group` – inferred HMM state
- `n_snps` – number of SNPs in the block
- `n_mendelian_error` – number of Mendelian-inconsistent genotypes in the block
- depth-ratio metrics (always present; if add_ratios = FALSE, filled with NA)

`vcfCheck` *Check quality parameters (optional), change IDs and encode genotypes numerically*

Description

This function takes a VCF file and converts it into a largeCollapsedVcf object using the VariantAnnotation package. It also rename the sample for subsequent steps needed in UPDhmm package and generates a numeric encoding of the trio genotypes for each variant, which is stored in the metadata column *geno_coded*. Additionally, it features an optional parameter, quality_check, which triggers warnings when variants lack sufficient quality based on RD and GQ parameters in the input VCF.

Usage

```
vcfCheck(largeCollapsedVcf, father, mother, proband, check_quality = FALSE)
```

Arguments

`largeCollapsedVcf`	The file in largeCollapsedVcf format.
`father`	Name of the father's sample.
`mother`	Name of the mother's sample.
`proband`	Name of the proband's sample.
`check_quality`	Optional argument. TRUE/FALSE. If quality parameters want to be measured. Default = FALSE

Value

largeCollapsedVcf (VariantAnnotation VCF format) object identical to the input with samples renamed to standard names for the trio and a new metadata column *geno_coded* containing the numeric encoding of the trio genotypes for each variant.

Examples

```
fl <- system.file("extdata", "test_het_mat.vcf.gz", package = "UPDhmm")
vcf <- VariantAnnotation::readVcf(fl)
processedVcf <-
   vcfCheck(vcf, proband = "NA19675", mother = "NA19678", father = "NA19679")
```

Index