Detection of de novo copy number alterations in case-parent trios using the R package MinimumDistance

Rob Scharpf

October 17, 2016

Abstract

For the analysis of case-parent trio genotyping arrays, copy number variants (CNV) appearing in the offspring that differ from the parental copy numbers are often of interest (de novo CNV). This package defines a statistic, referred to as the minimum distance, for identifying de novo copy number alterations in the offspring. We smooth the minimum distance using the circular binary segmentation algorithm implemented in the Bioconductor package DNAcopy. Trio copy number states are inferred from the maximum a posteriori probability of the segmented data, incorporating information from the log R ratios and B allele frequencies. As both log R ratios and B allele frequencies are estimable from Illumina and Affymetrix arrays, this package supports de novo copy number inference in both platforms.

1 Introduction

There are numerous R packages available from Bioconductor for smoothing copy number alterations. For example, the biocview `CopyNumberVariants` in the 2.9 release of Bioconductor lists 27 packages. For the analysis of germline diseases, hidden Markov models have emerged as a popular tool as inference regarding the latent copy number state incorporates information from both the estimates of copy number (or relative copy number) and the allelic frequencies, such as the B allele frequency (**?**) or genotype calls (**???**). For the analysis of somatic cell diseases such as cancer, algorithms that segment the genome into regions of constant copy number (referred to here as segmentation algorithms) may be more preferable for the detection of copy number aberrations (CNA) as a mixture of cells with different copy numbers give rise to mosaic (non-integer) copy numbers. Examples include circular binary segmentation implemented in the R package DNAcopy and the GLAD, both of which were orginally developed for array CGH platforms **???**. One disadvantage of segmentation algorithms is that inference regarding duplication and deletions is not directly available.

More recently, HMMs and segmentation algorithms have been developed for inferring common regions of copy number alterations in multiple samples. However, relatively few algorithms are available for inferring copy number alterations, especially de novo copy number alterations, in family-based study designs involving case-parent trios. Instead, a common strategy has been a two-step approach of identifying copy number alterations in the individual samples and then comparing the results across samples to infer whether an alteration observed in the offspring is de novo. A disadvantage of the two-step approach is that unadjusted sources of technical variation, such as waves, can contribute to false positives. The joint HMM implemented in the PennCNV software is one of the few algorithms to date that provides direct inference regarding de novo alterations in case parent study designs.

This package develops an alternative framework for inferring regions of de novo copy number alterations in case-parent trios. Like the PennCNV joint HMM, inference regarding de novo alterations integrates information from both the log R ratios and B allele frequencies. Differences in the two approaches are most apparent in non-HapMap experimental datasets in which technical and experimental sources of variation appear to contribute to a large number of false positives. This vignette describes the analysis pipeline from preprocessed and normalized estimates of copy number and allele frequencies to inference of de novo copy number alterations. The workflow is illustrated using a publicly available HapMap trio and a publicly available oral cleft trio, each assayed on the Illumina 610quad array.

2 Data input

```
R> library(oligoClasses)
R> library(VanillaICE)
R> library(MinimumDistance)
R> foreach::registerDoSEQ()
```

2.1 Reading and organizing the annotation on the markers

We require that marker-level annotation is represented as a *GRanges*-derived class. To read the plain text annotation file, we use the `fread` function provided in the data.table package. In addition, we define an indicator for whether the marker is polymophic using the 'Intensity Only' flag. The *SnpGRanges* class created from the `fgr` object in the following code-chunk ensures that this binary indicator is created and can be reliably accessed.

```
R> library(data.table)
R> extdir <- system.file("extdata", package="VanillaICE")
R> features <- suppressWarnings(fread(file.path(extdir, "SNP_info.csv")))
R> fgr <- GRanges(paste0("chr", features$Chr), IRanges(features$Position, width=1),
                  isSnp=features[["Intensity Only"]]==0)
R> fgr <- SnpGRanges(fgr)
R> names(fgr) <- features[["Name"]]
```

Ideally, one should include the genome build and the chromosome lengths appropriate to the build. Here, we extract the metadata on the chromosomes using the BSgenome Bioconductor package for the hg18 build. Finally, we sort the `fgr` object such that the chromosomes are ordered by their `seqlevels` and the markers are ordered by their genomic position along the chromosome.

```
R> library(BSgenome.Hsapiens.UCSC.hg18)
R> sl <- seqlevels(BSgenome.Hsapiens.UCSC.hg18)
R> seqlevels(fgr) <- sl[sl %in% seqlevels(fgr)]
R> seqinfo(fgr) <- seqinfo(BSgenome.Hsapiens.UCSC.hg18)[seqlevels(fgr),]
R> fgr <- sort(fgr)
```

2.2 Organizing the marker-level summaries

The abbreviated plain text files included with this package that contain the log R ratios and B allele frequencies from 2 trios are listed below.

```
R> files <- list.files(extdir, full.names=TRUE, recursive=TRUE, pattern="FinalReport")
```

We parse these files into a specific format such that all downstream steps for copy number estimation no longer depend on the format of the source files. At this point, we encapsulate the names of the source files ('sourcePaths'), the location of where we intend to store the parsed data ('parsedPath'), and the genomic marker annnotation created in the preceding section ('rowRanges') in a single object called an *ArrayViews*.

```
R> ##
R> ## Where to keep parsed files
R> ##
R> parsedDir <- "ParsedFiles"
R> if(!file.exists(parsedDir)) dir.create(parsedDir)
R> views <- ArrayViews(rowRanges=fgr, sourcePaths=files, parsedPath=parsedDir)
R> show(views)
```

```
class 'ArrayViews'
   No. files  : 6
   No. markers: 11780
```

Because the format of the source files depends on upstream software, we read one file and store information regarding its parsing so that subsequent files can be parsed in a similar fashion. We use the `fread` to read in the first file.

```
R> ## read the first file
R> dat <- fread(files[1])
R> head(dat,n=3)
```

```
      SNP Name Allele1 - AB Allele2 - AB B Allele Freq
1: cnvi0005126            -            -        0.0085
2: cnvi0005127            -            -        0.0081
3: cnvi0005128            -            -        0.0063
   Log R Ratio
1:     -0.7214
2:     -0.8589
3:     -0.1640
```

Next, we select which columns we plan to keep. Again, the required data for downstream processing is the name of the SNP identifier, the log R ratios, and B allele frequencies.

```
R> ## information to store on the markers
R> select_columns <- match(c("SNP Name", "Allele1 - AB", "Allele2 - AB",
                           "Log R Ratio", "B Allele Freq"), names(dat))
```

We also specify the order in which we will store the marker-level summaries by matching the `rownames` of the `views` object with the names of the markers in the source file:

```
R> index_genome <- match(names(fgr), dat[["SNP Name"]])
```

Similar to the parameter classes defined in Rsamtools, we encapsulate the information for parsing the columns and rows of the source files in a class. In addition, we specify which variable names in the source file refers to log R ratios ('cnvar'), B allele frequences ('bafvar'), and genotypes ('gtvar').

```
R> scan_params <- CopyNumScanParams(index_genome=index_genome, select=select_columns,
                                    cnvar="Log R Ratio",
                                    bafvar="B Allele Freq",
                                    gtvar=c("Allele1 - AB", "Allele2 - AB"))
```

The `parseSourceFile` will parse a single file in the `views` object (by default, the first file) according to the parameters for reading the data in the `scan_params` object and write the output to the `parsedPath` directory. In particular, the `parseSourceFile` returns *NULL*.

```
R> parsedPath(views)
```

```
[1] "ParsedFiles"
```

```
R> parseSourceFile(views[, 1], scan_params)
```

To apply the function `parseSourceFile` to all arrays in the `views` object, one can use the functional `sapply`.

```
R> invisible(sapply(views, parseSourceFile, param=scan_params))
```

Apart from confirming their existance, the user should not have a need to directly access the parsed files. Utilities for querying these files are provided through the `views` object.

```
R> head(list.files(parsedPath(views)), n=3)
```

```
[1] "FinalReport1664_baf.rds" "FinalReport1664_gt.rds"
[3] "FinalReport1664_lrr.rds"
```

2.3 Accessors for the parsed data

The preference for writing the parsed data to disk rather than keeping the data in RAM is simply that the latter does not scale to projects involving thousands of samples. For the former, slices of the parsed data easily be accessed from the `parsedPath` directory via methods defined for the *ArrayViews* class. For example, one can use accessors for the low-level summaries directly: `lrr`, `baf`, and `genotypes` for log R ratios, B allele frequencies, and genotypes, respectively. The user has the option of either subsetting the views object or subsetting the matrix returned by the accessor to extract the appropriate data slice. In the following examples, we access data on the first 2 markers and sample indices 2-4.

```
R> lrr(views)[1:2, 2:4]
```

```
           FinalReport1675.txt FinalReport1686.txt
rs12789205              -0.169               0.191
rs2114088               -0.182              -0.114
           FinalReport6841.txt
rs12789205              -0.084
rs2114088                0.253
```

```
R> ## or
R> lrr(views[1:2, 2:4])
```

```
           FinalReport1675.txt FinalReport1686.txt
rs12789205              -0.169               0.191
rs2114088               -0.182              -0.114
           FinalReport6841.txt
rs12789205              -0.084
rs2114088                0.253
```

```
R> ## B allele frequencies
R> baf(views[1:2, 2:4])
```

```
           FinalReport1675.txt FinalReport1686.txt
rs12789205               0.529               0.549
rs2114088                0.000               0.000
           FinalReport6841.txt
rs12789205               0.481
rs2114088                0.000
```

```
R> ## potentially masked by function of the same name in crlmm
R> VanillaICE::genotypes(views)[1:2, 2:4]
```

```
           FinalReport1675.txt FinalReport1686.txt
rs12789205                   2                   2
rs2114088                    1                   1
           FinalReport6841.txt
rs12789205                   2
rs2114088                    1
```

More often, it is useful to extract the low-level data in a *RangedSummarizedExperiment*-derived class such that meta-data on the samples remains bound to the columns of the assay data (log R ratios / B allele frequencies) and meta-data on the rows remains bound to the rows of the assay data. This is accomplished by applying the `MinDistExperiment` function to a `views` object, as described in the following section.

3 Binding the genomic, marker, and individual-level data

3.1 Pedigree annotation

The container for a single pedigree is the *ParentOffspring* class.

```
R> ped_hapmap <- ParentOffspring(id = "hapmap", father="12287_03",
                                 mother="12287_02",
                                 offspring="12287_01",
                                 parsedPath=parsedPath(views))
```

For families with multiple affected offspring, the argument to offspring can be a character-vector with length greater than 1. We can store any number of *ParentOffspring* objects in a list using the *ParentOffspringList* class. In particular, for the 2 trios provided in the VanillaICE package

```
R> ped_list <- ParentOffspringList(pedigrees=list(
                                   ParentOffspring(id = "hapmap", father="12287_03",
                                                   mother="12287_02",
                                                   offspring="12287_01",
                                                   parsedPath=parsedPath(views)),
                                   ParentOffspring(id = "cleft",
                                                   father="22169_03",
                                                   mother="22169_02",
                                                   offspring="22169_01",
                                                   parsedPath=parsedPath(views))))
R> pedigreeName(ped_list)

[1] "hapmap" "cleft"
```

For a single pedigree, the *MinDistExperiment* encapsulates the annotation on the pedigree, the genomic annotation of the markers, and the marker-level summary statistics. Note, however, that the sample identifiers in the pedigree are not the same as the file identifiers used by default to create the R object `views`. To resolve the naming issue, we read in separate file provided in the VanillaICE package:

```
R> sample_info <- read.csv(file.path(extdir, "sample_data.csv"), stringsAsFactors=FALSE)
R> ind_id <- setNames(gsub(" ", "", sample_info$IndividualID), sample_info$File)
R> colnames(views) <- ind_id[gsub(".txt", "", colnames(views))]
```

3.2 MinDistExperiment class

The constructor function for *MinDistExperiment* extracts from the views object only the data relevant for the provided pedigree.

```
R> me <- MinDistExperiment(views, pedigree=ped_list[[2]])
R> colnames(me)

[1] "22169_03" "22169_02" "22169_01"

R> me

class: MinDistExperiment
dim: 11780 3
metadata(0):
assays(2): cn baf
rownames(11780): rs12789205 rs2114088 ... rs5994329
  rs1055232
rowData names(1): isSnp
colnames(3): 22169_03 22169_02 22169_01
colData names(0):
```

4 Detection of de novo copy number variants

A container for the various parameters used to segment and call de novo copy number variants is provided by the *MinDistParam*. A constructor function of the same name provides a default parametrization that works well in most instances:

```
R> params <- MinDistParam()
R> show(params)
```

```
An object of class 'MinDistParam'
  call segments with |seg.mean|/MAD >  0.75
  Setting nMAD() to smaller values will increase the number of segments that are called.
  DNAcopy settings:
    alpha:  0.01
    min.width:  2
    undo.splits:  none
    undo.SD:  3
    See segment() for description of DNAcopy parameters
```

The parameters in this class are organized according to function. For example, the argument `dnacopy` takes an argument of class *DNAcopyParam* that contains setting that are passed to the `segment` function for the implementation of circular binary segmentation in the R package DNAcopy. Changing the parameters for the segmentation is most easily accomplished by a call to the constructor. E.g.,

```
R> segment_params <- DNAcopyParam(alpha=0.01)
R> params <- MinDistParam(dnacopy=segment_params)
```

Several of the parameters relate to a priori assumptions of the probability of a non-Mendelian transmission. These a priori assumptions are defined in PennCNV **?** and encapsulated in the *PennParam* class.

```
R> penn_param <- PennParam()
R> show(penn_param)
```

```
Object of class `PennParam'
   121  trio states
   state names: 000,100,200,300,400,010 ...
   reference state: 222
   PennCNV Table 1:  121  vector
   PennCNV Table 3: 5 x 5 x 5 x 5 x 5 x 5 array
   probability non-Mendelian: 1.5e-06
   initial state probabilities:  0.2,0.2,0.2,0.2,0.2
   transition prob: 5 x 5 matrix
   See table1(), table3(), state(), stateNames(), referenceState()
```

Finally, parameters for the emission probabilities computed by the R package VanillaICE are encapsulated in the *EmissionParam* class. Again, default values for the emission probabilities will be created automatically as part of the `params` object if none is specified.

Segmentation and posterior calls. For a given trio, the signed minimum absolute difference of the offspring and parental $\log_2$ R ratios ($\boldsymbol{r}$) is defined as

$$\boldsymbol{d} \equiv (\boldsymbol{r}_{\mathrm{O}} - \boldsymbol{r}_{\mathrm{M}}) \times \mathbb{I}_{[|\boldsymbol{r}_{\mathrm{O}} - \boldsymbol{r}_{\mathrm{F}}| > |\boldsymbol{r}_{\mathrm{O}} - \boldsymbol{r}_{\mathrm{M}}|]} + (\boldsymbol{r}_{\mathrm{O}} - \boldsymbol{r}_{\mathrm{F}}) \times \mathbb{I}_{[|\boldsymbol{r}_{\mathrm{O}} - \boldsymbol{r}_{\mathrm{F}}| \leq |\boldsymbol{r}_{\mathrm{O}} - \boldsymbol{r}_{\mathrm{M}}|]}. \quad (1)$$

If the offspring copy number changes within a minimum distance segment (as determined by the segmentation of the offspring copy number), the start and stop position of the minimum distance segments may be edited. The approach currently implemented is to define a new start and stop position if a breakpoint for the offspring segmentation occurs in the minimum distance interval. To illustrate, the following diagram uses vertical dashes (|) to denote breakpoints:

```
1 ...--|--------------|--...     ## minimum distance segment (before editing)
2 ...----|--------|------...     ## segmenation of log R ratios for offspring

->

3 ...--|-|--------|---|--...     ## after editing
```

In the above illustration, posterior calls are provided for the 3 segments indicated in line 3 instead of the single segment in line 1. Two additional steps are therefore required: (1) segmentation of the offspring log R ratios and (2) editing of the minimum distance breakpoints when appropriate. The following codechunk, segments the log R ratios for all chromosomes in the `me` object and edits the ranges for the minimum distance when conflicts with the offspring segmentation boundaries arise (as illustrated above).

```
R> mdgr <- segment2(me, params)
```

```
Analyzing: X22169_03
Analyzing: X22169_02
Analyzing: X22169_01
Analyzing: md_22169_01
```

For each minimum distance segment in which the mean minimum distance is above a user-specificed cutoff in absolute value (specified in terms of the median absolute deviations of the minimum distance), a trio copy number state is assigned from the maximum a posteriori estimate.

```
R> ## the threshold in terms of the number of median absolute deviations from zero
R> nMAD(params)
R> md_g <- MAP2(me, mdgr, params)
R> show(md_g)
```

5 Inspecting, Filtering, and plotting de novo CNV inference

5.1 Filtering

There are several meta-data columns stored in the *GRanges*-derived summary useful for filtering the set of genomic intervals. All the available parameters for filtering the `fit` object are stored in the parameter class *FilterParam* of which an instance can be created by its constructor of the same name without any arguments.

```
R> filter_param <- FilterParamMD()
R> show(filter_param)
```

```
An object of class 'FilterParamMD'
   min. posterior probability of trio CNV call: 0.99
   min. no. of markers spanned by segment     : 10
   min. width of segment                      : 1
   121 selected trio CN states               : 000, 100, 200, 300, 400, 010, ...
   selected seqnames                          : chr1, chr2, chr3, chr4, chr5, chr6 ...
```

To apply the default filter parameters to the `fit` object, we use the `cnvFilter` function. The `cnvFilter` function returns only the set of genomic ranges satisfying the filters. For example,

```
R> cnvFilter(md_g, filter_param)
```

```
MDRanges object with 5 ranges and 9 metadata columns:
      seqnames               ranges strand |      sample
         <Rle>            <IRanges>  <Rle> | <character>
  [1]    chr11 [55139733, 55204003]      * | md_22169_01
```

```
  [2]    chr22 [17060279, 17270615]      * | md_22169_01
  [3]    chr22 [17281004, 17633332]      * | md_22169_01
  [4]    chr22 [17670969, 18670691]      * | md_22169_01
  [5]    chr22 [19066315, 19792353]      * | md_22169_01
      seg.mean    log_RR  log_odds  prob_MAP  prob_222
     <numeric> <numeric> <numeric> <numeric> <numeric>
  [1]   -0.4712    72.737   -82.214         1         0
  [2]   -0.0970     0.000   -91.046         1         1
  [3]   -0.3178   152.429       Inf         1         0
  [4]   -0.3540   697.008       Inf         1         0
  [5]   -0.3359   430.425       Inf         1         0
      prob_221       calls number_probes
     <numeric> <character>     <integer>
  [1]         0         302            25
  [2]         0         222            25
  [3]         1         221            88
  [4]         1         221           278
  [5]         1         221           208
  -------
  seqinfo: 3 sequences from hg18 genome
```

The helper functions `denovoHemizygous` and `denovoHomozygous` create commonly used filters.

```
R> denovoHemizygous(md_g)

MDRanges object with 3 ranges and 9 metadata columns:
      seqnames               ranges strand |      sample
         <Rle>            <IRanges>  <Rle> | <character>
  [1]    chr22 [17281004, 17633332]      * | md_22169_01
  [2]    chr22 [17670969, 18670691]      * | md_22169_01
  [3]    chr22 [19066315, 19792353]      * | md_22169_01
      seg.mean    log_RR  log_odds  prob_MAP  prob_222
     <numeric> <numeric> <numeric> <numeric> <numeric>
  [1]   -0.3178   152.429       Inf         1         0
  [2]   -0.3540   697.008       Inf         1         0
  [3]   -0.3359   430.425       Inf         1         0
      prob_221       calls number_probes
     <numeric> <character>     <integer>
  [1]         1         221            88
  [2]         1         221           278
  [3]         1         221           208
  -------
  seqinfo: 3 sequences from hg18 genome

R> denovoHomozygous(md_g)

MDRanges object with 0 ranges and 9 metadata columns:
   seqnames    ranges strand |      sample  seg.mean    log_RR
      <Rle> <IRanges>  <Rle> | <character> <numeric> <numeric>
    log_odds  prob_MAP  prob_222  prob_221       calls
   <numeric> <numeric> <numeric> <numeric> <character>
   number_probes
       <integer>
  -------
  seqinfo: 3 sequences from hg18 genome
```

Equivalently, one could customize the filter parameteters to acheive the same result:

```
R> select_cnv <- FilterParamMD(state=c("220", "221", "223"), seqnames="chr22")
R> cnvs <- cnvFilter(md_g, select_cnv)
R> cnvs

MDRanges object with 3 ranges and 9 metadata columns:
      seqnames              ranges strand |      sample
         <Rle>           <IRanges>  <Rle> | <character>
  [1]    chr22 [17281004, 17633332]      * | md_22169_01
  [2]    chr22 [17670969, 18670691]      * | md_22169_01
  [3]    chr22 [19066315, 19792353]      * | md_22169_01
       seg.mean    log_RR  log_odds  prob_MAP  prob_222
      <numeric> <numeric> <numeric> <numeric> <numeric>
  [1]   -0.3178   152.429       Inf         1         0
  [2]   -0.3540   697.008       Inf         1         0
  [3]   -0.3359   430.425       Inf         1         0
       prob_221       calls number_probes
      <numeric> <character>     <integer>
  [1]         1         221            88
  [2]         1         221           278
  [3]         1         221           208
  -------
  seqinfo: 3 sequences from hg18 genome
```

5.2 Visualization

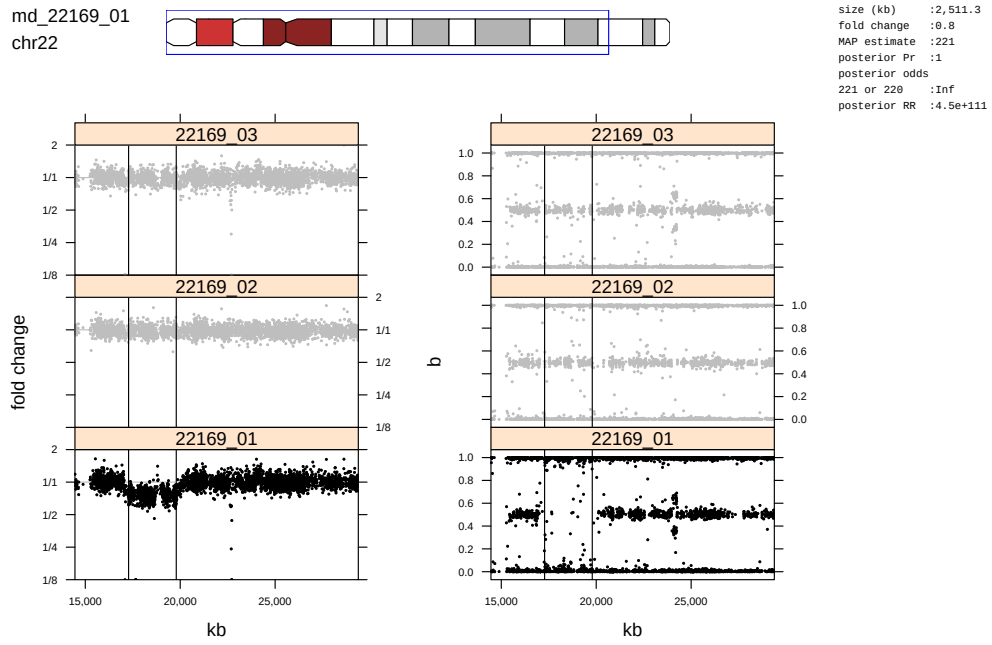
This produces a somewhat unsatisfactory result in that there are 2 additional denovo hemizygous deletions nearby that likely comprise one denovo segment.

```
R> denovoHemizygous(md_g)

MDRanges object with 3 ranges and 9 metadata columns:
      seqnames              ranges strand |      sample
         <Rle>           <IRanges>  <Rle> | <character>
  [1]    chr22 [17281004, 17633332]      * | md_22169_01
  [2]    chr22 [17670969, 18670691]      * | md_22169_01
  [3]    chr22 [19066315, 19792353]      * | md_22169_01
       seg.mean    log_RR  log_odds  prob_MAP  prob_222
      <numeric> <numeric> <numeric> <numeric> <numeric>
  [1]   -0.3178   152.429       Inf         1         0
  [2]   -0.3540   697.008       Inf         1         0
  [3]   -0.3359   430.425       Inf         1         0
       prob_221       calls number_probes
      <numeric> <character>     <integer>
  [1]         1         221            88
  [2]         1         221           278
  [3]         1         221           208
  -------
  seqinfo: 3 sequences from hg18 genome
```

In the following code-chunk, we reduce the denovo hemizygous deletions and update the posterior probabilities for the MAP estimates. We use a combination of lattice and grid to visualize the marker-level summaries and copy number inference for the trio.

```
R> library(grid)
R> g2 <- reduce(denovoHemizygous(md_g), min.gapwidth=500e3)
R> post <- MAP2(me, g2, params)
R> g2 <- denovoHemizygous(post)
R> vps <- pedigreeViewports()
R> grid.params <-  HmmTrellisParam()
R> p <- plotDenovo(me, g2, grid.params)
R> pedigreeGrid(g=g2, vps=vps, figs=p)
```



6 Acknowledgements

Moiz Bootwalla contributed to early versions of this vignette.

7 Session information

```
R> toLatex(sessionInfo())
```

- R version 3.3.1 (2016-06-21), `x86_64-apple-darwin13.4.0`
- Locale: `C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Base packages: base, datasets, grDevices, graphics, grid, methods, parallel, stats, stats4, utils
- Other packages: BSgenome 1.42.0, BSgenome.Hsapiens.UCSC.hg18 1.3.1000, Biobase 2.34.0, BiocGenerics 0.20.0, Biostrings 2.42.0, GenomeInfoDb 1.10.0, GenomicRanges 1.26.0, IRanges 2.8.0, MinimumDistance 1.18.0, S4Vectors 0.12.0, SummarizedExperiment 1.4.0, VanillaICE 1.36.0, XVector 0.14.0, data.table 1.9.6, oligoClasses 1.36.0, rtracklayer 1.34.0
- Loaded via a namespace (and not attached): BiocInstaller 1.24.0, BiocParallel 1.8.0, DBI 0.5-1, DNAcopy 1.48.0, GenomicAlignments 1.10.0, Matrix 1.2-7.1, RCurl 1.95-4.8, RSQLite 1.0.0, Rcpp 0.12.7, RcppEigen 0.3.2.9.0, Rsamtools 1.26.0, VGAM 1.0-2, XML 3.98-1.4, affyio 1.44.0, base64 2.0, beanplot 1.2, bit 1.1-12, bitops 1.0-6, chron 2.3-47, codetools 0.2-15, compiler 3.3.1, crlmm 1.32.0, ellipse 0.3-8, ff 2.2-13, foreach 1.4.3, illuminaio 0.16.0, iterators 1.0.8, lattice 0.20-34,

limma 3.30.0, matrixStats 0.51.0, mvtnorm 1.0-5, openssl 0.9.4, preprocessCore 1.36.0, splines 3.3.1, tools 3.3.1, zlibbioc 1.20.0