Example data analysis

2023-05-20

Contents

1 Create the example dataset

- We start by loading the necessary R packages and functions.

```
library(SimRVSequences)
library(RVMethods)
library(Matrix)
# Load the cd_new() function from its R source file
source("cd_new.R")
# Load worker functions we will call from a source file
source("workerfuncs.R")
```

- Next we read into our R environment the population exome sequences that will be sampled to create the data.

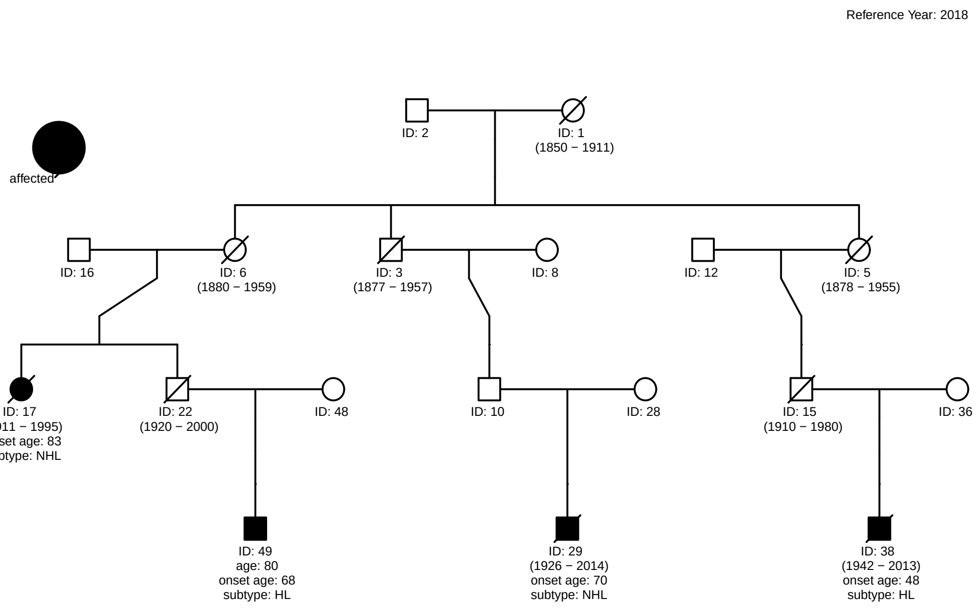
```
infileDir <- "FJdata"
load(paste0(infileDir,"/chr8.RData"))
```

- We also create the vector of carrier probabilities to consider for calculating global likelihood-ratio statistics and configuration probabilities under the null hypothesis (for p-values in the global approaches).
  - The true value of the carrier probability used to simulate the data is 0.00032. We will consider this value plus misspecified values of 1/100,1/10, 1/2, 2 times, 10 times and 100 times the true value.

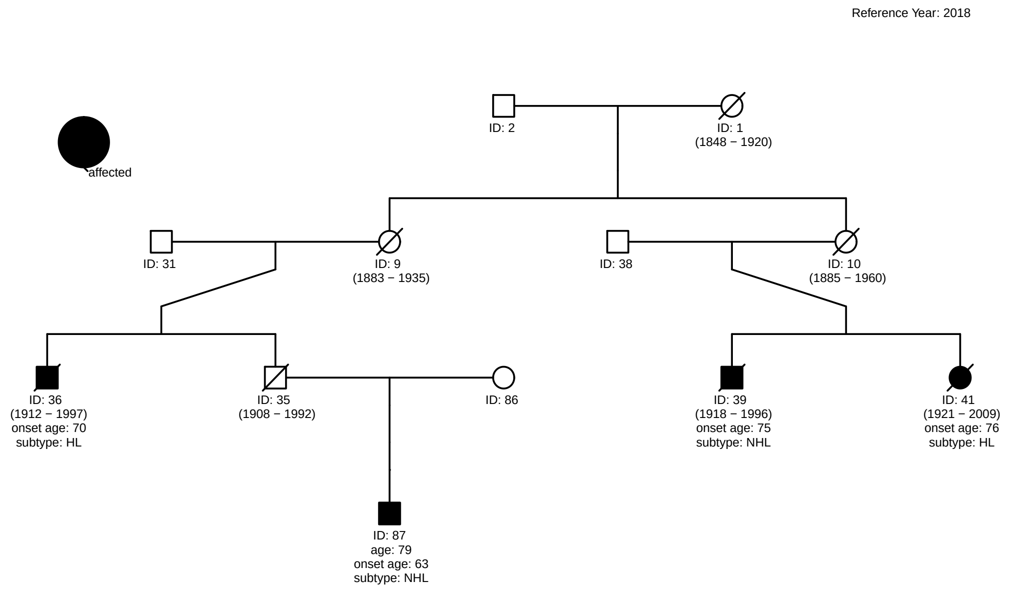
```
true_carrier_prob <- 0.00032
carrier_probs <- true_carrier_prob*c(1/100,1/10,1/2,1,2,10,100)
```

- Next, we re-create the pedigrees from Figure 4.4 of Christina's thesis.
- We don't have the random seeds Christina used but we can re-create the pedigree data structures in a CSV file and read that into an R data frame called `expeds`. Pedigrees are distinguished by a `FamID` variable in the data frame.
- To plot a pedigree, we select it from `expeds`. We hide the RV status in the pedigree plots by setting the RV status variables `DA1` and `DA2` in the pedigree data structure to `NULL`.

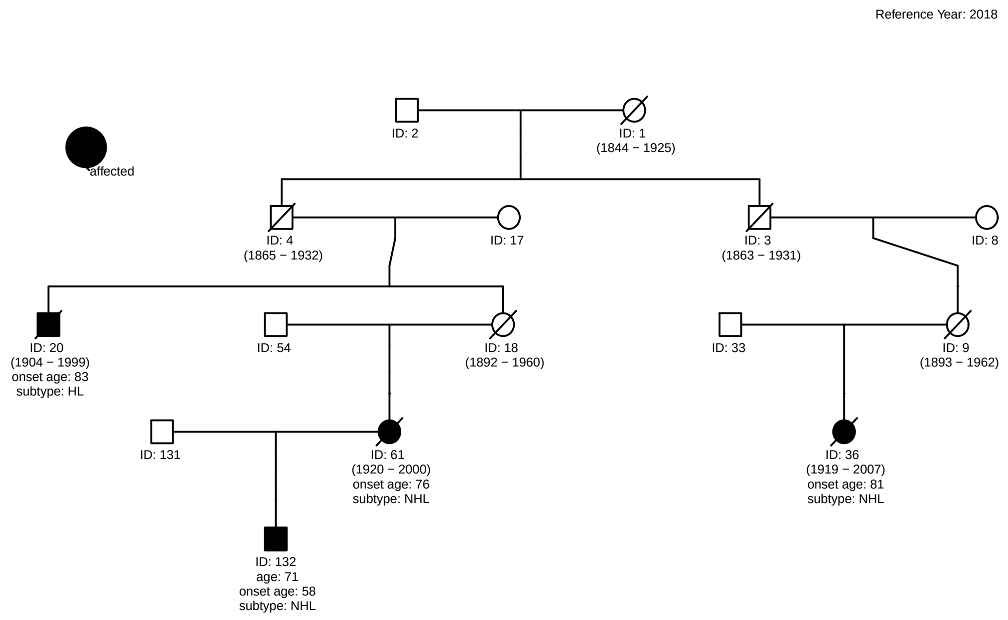
```r
expeds <- read.csv("exdatpeds.csv")
class(expeds) <- c("ped","data.frame")
ped1 <- expeds[expeds$FamID==1,]
ped1$DA1 <- ped1$DA2 <- NULL
plot(ped1,ref_year=2018,cex=.4)
```



```r
ped2 <- expeds[expeds$FamID==2,]
ped2$DA1 <- ped2$DA2 <- NULL
plot(ped2,ref_year=2018,cex=.4)
```



```r
ped3 <- expeds[expeds$FamID==3,]
ped3$DA1 <- ped3$DA2 <- NULL
plot(ped3,ref_year=2018,cex=.4)
```



- Lastly, we simulate the chromosome 8 sequences for the three pedigrees.
- We use `sim_RVstudy()` from Christina's `SimRVSequences` package. `sim_RVstudy()` takes the study data-frame as input, but operates on the individual pedigrees separately.
- For each pedigree it (i) samples a cSNV from the pool of cSNVs, (ii) samples founder sequences from Nirodha's American Admixed population, and (iii) conditionally gene drops these sequences through the pedigree.
- After simulating sequences, we filter RVs to those that are observed in the affected individuals.
- Note: The next code chunk is the first use of the random number generator so we set the seed here.

```
set.seed(1)
#  Generate sequence data with sim_RVstudy() and then filter RVs further
# to those observed in affected individuals.
s_seqs <- sim_RVstudy(expeds,chr8) # *Christina's*
a_seqs <- filter2aff(s_seqs) # *worker*
```

2 Summarize the study's sequence data

- I use a modified version of Christina's summary method for objects of class "famStudy" (output of `sim_RVstudy`) to get counts for each RV observed in each family, *split by the subtype* of the affected individuals.

```
allele_summary <- mysummary.famStudy(a_seqs) # *worker*
```

- On chromosome 8, 74 RVs are observed in the affected individuals. Fifty seven of the RVs appear only once in the study, 14 appear twice and three appear three times:

```
total_counts <- rowSums(allele_summary[,c("tot1","tot2","tot3")])
table(total_counts)
```

```
## total_counts
##  1  2  3
## 57 14  3
```

```
sum(table(total_counts))
```

```
## [1] 74
```

- To reduce the output, we focus on the RVs that appear more than once in the study and construct a LaTeX table to show their counts in the paper. From the summary data in `allele_summary` we omit the columns of total counts within each pedigree.

```
exclude_cols <- c(5,8,11)
red_allele_sum <- allele_summary[total_counts > 1,-exclude_cols]
```

- A template for the header and footer of the LaTeX table is shown below, followed by the output of `xtable()`, which includes the body of the LaTeX table. Finally, I manually cut-and-paste the template and table body together to make the table.

```
\begin{table}[ht]
\centering
\begin{tabular}{lrrrrrrrr}
& & & \multicolumn{2}{c}{ped 1} & \multicolumn{2}{c}{ped 2} & \multicolumn{2}{c}{ped 3} \\ \cline{4-9}
\multicolumn{3}{l}{} & NHL & HL & NHL & HL & NHL & HL  \\
  \hline
\multicolumn{3}{l}{Number of affected individuals:} & 2 & 2 & 2 & 2 & 3 & 1 \\ \hline
RV  & chrom & position & & & & & &  \\
  \hline
% Table body starts here

% Table body ends here
\end{tabular}
\caption{Simulated RVs that appear in more than one affected individual, with counts by disease subtype within each pedigree.  Only one RV, 8\_6708389, appears in multiple families.}
\end{table}
```

```
library(xtable)
tem <- xtable(red_allele_sum,digits=0) #print integers
print(tem,type="latex")
```

```
## % latex table generated in R 4.0.2 by xtable 1.8-4 package
## % Thu May 25 16:19:43 2023
## \begin{table}[ht]
## \centering
## \begin{tabular}{rrrrrrrrr}
##   \hline
##  & chrom & position & NHL1 & HL1 & NHL2 & HL2 & NHL3 & HL3 \\
##   \hline
## 8\_6708389 & 8 & 6708389 & 1 & 0 & 0 & 1 & 0 & 0 \\
##   8\_7972232 & 8 & 7972232 & 0 & 0 & 0 & 2 & 0 & 0 \\
##   8\_8376739 & 8 & 8376739 & 1 & 1 & 0 & 0 & 0 & 0 \\
##   8\_12116292 & 8 & 12116292 & 0 & 0 & 2 & 0 & 0 & 0 \\
##   8\_23020454 & 8 & 23020454 & 0 & 2 & 0 & 0 & 0 & 0 \\
##   8\_23068413 & 8 & 23068413 & 0 & 0 & 0 & 2 & 0 & 0 \\
##   8\_28452476 & 8 & 28452476 & 0 & 0 & 0 & 0 & 2 & 1 \\
##   8\_30846116 & 8 & 30846116 & 0 & 0 & 2 & 0 & 0 & 0 \\
##   8\_39680176 & 8 & 39680176 & 1 & 2 & 0 & 0 & 0 & 0 \\
##   8\_41261959 & 8 & 41261959 & 0 & 0 & 2 & 0 & 0 & 0 \\
##   8\_54047456 & 8 & 54047456 & 0 & 0 & 1 & 1 & 0 & 0 \\
##   8\_55454701 & 8 & 55454701 & 0 & 0 & 0 & 0 & 2 & 0 \\
##   8\_60866956 & 8 & 60866956 & 1 & 1 & 0 & 0 & 0 & 0 \\
##   8\_65161224 & 8 & 65161224 & 0 & 0 & 0 & 0 & 2 & 1 \\
##   8\_103382957 & 8 & 103382957 & 1 & 1 & 0 & 0 & 0 & 0 \\
##   8\_107249922 & 8 & 107249922 & 0 & 0 & 1 & 1 & 0 & 0 \\
##   8\_142271244 & 8 & 142271244 & 0 & 0 & 1 & 1 & 0 & 0 \\
##    \hline
## \end{tabular}
## \end{table}
```

- Table 1 below shows the resulting table of RVs appearing more than once in the affected individuals of the study.

3 Analyze the study's sequence data

- We want to prioritize the RVs in our dataset for further investigation. To prioritize, we use the value of the RV test statistics to rank them with each method. We first call `cd_new()` to get lookup tables of the statistic values.
- Coding note: As the call to `cd_new()` takes a while, we tell RMarkdown to "cache" the results by setting the option `cache=TRUE` in the code chunk. When you specify this option RMarkdown will create two folders to hold the cached results in your working directory; in this case the folders are `exdatan_files` and `exdatan_cache`. If you remove these folders or change the code chunk, RMarkdown will start over with the code chunk and rerun it from scratch.

```
lookupTabs <- cd_new(peds = expeds, subtypes = c("HL", "NHL"),
                     carrier_probs = carrier_probs)
```

			ped 1		ped 2		ped 3	
			NHL	HL	NHL	HL	NHL	HL
Number of affected individuals:			2	2	2	2	3	1
RV	chrom	position						
8_6708389	8	6708389	1	0	0	1	0	0
8_7972232	8	7972232	0	0	0	2	0	0
8_8376739	8	8376739	1	1	0	0	0	0
8_12116292	8	12116292	0	0	2	0	0	0
8_23020454	8	23020454	0	2	0	0	0	0
8_23068413	8	23068413	0	0	0	2	0	0
8_28452476	8	28452476	0	0	0	0	2	1
8_30846116	8	30846116	0	0	2	0	0	0
8_39680176	8	39680176	1	2	0	0	0	0
8_41261959	8	41261959	0	0	2	0	0	0
8_54047456	8	54047456	0	0	1	1	0	0
8_55454701	8	55454701	0	0	0	0	2	0
8_60866956	8	60866956	1	1	0	0	0	0
8_65161224	8	65161224	0	0	0	0	2	1
8_103382957	8	103382957	1	1	0	0	0	0
8_107249922	8	107249922	0	0	1	1	0	0
8_142271244	8	142271244	0	0	1	1	0	0

Table 1: Counts of RVs within each pedigree by disease subtype, for RVs that appear in more than one affected individual in the study. Only one RV, 8_6708389, appears in multiple families.

- All of the statistical methods assume that cRVs enter pedigrees through at most one founder. We therefore filter out "invalid" RVs that are not consistent with this assumption.

```
a_seqs <- remove_invalid_configs(a_seqs,lookupTabs) # *worker*
nrow(a_seqs$SNV_map)
```

```
## [1] 72
```

- The above filtering removes 2 of the 74 RVs, leaving 72.

3.1 Summary of rank results

- We use the global configurations of the RVs from the lookup tables returned by the `cd_new()` function.
- The ordering of the affected individuals in these configurations does not match the ordering of the affected individuals in the rows of the genotypes matrix derived from Christina's `sim_RVstudy()` function.
- We must therefore reorder the rows of the genotype matrix to match the order of the affected individuals in the configurations of the lookup table.
- The order of affected individuals in the global configurations is contained in the variable `binKey` returned by `cd_new()`.
- We use `binKey` and the worker function `reorder_genos()` to reorder the rows of the genotype matrix.

```
a_seqs <- reorder_genos(a_seqs,lookupTabs$binKey)
```

- Now we are ready to rank each RV in our filtered dataset using the value of the test statistics for its global configuration.
- The ranks from the first six global LR methods, with assumed carrier probabilities of 1/100, 1/10, 1/2, 1, 2 and 10 times the true value, are all identical and are represented by the variable `globalLR1to6` in

the following code.

```
RVs <- a_seqs$SNV_map$marker
is_CRV <- a_seqs$SNV_map$is_CRV
allranks <- get_ranks(RVs,lookupTabs,a_seqs) # *worker*
rankcols <- c(paste0("globalLR",1:length(carrier_probs)),
               "globaltrans","localLR","RVS","modRVS")
colnames(allranks) <- rankcols
rownames(allranks) <- RVs
# Rankings of first 6 LR methods are the same; e.g.,
all.equal(allranks[,"globalLR1"],allranks[,"globalLR6"])
```

```
## [1] TRUE
```

```
# Use globalLR1 as a representative of the first 6; i.e., remove
# globalLR2, ..., globalLR6 and relabel globalLR1 as globalLR1to6
allranks <- allranks[,-c(2:6)]
colnames(allranks)[1] <- "globalLR1to6"
```

- We print out the names of top-10-ranked RVs and mark cRVs with asterisks.

```
RVs <- a_seqs$SNV_map$marker
is_CRV <- a_seqs$SNV_map$is_CRV
# add one asterisk to name of the first cRV, two asterisks to the
# name of the second cRV and three to the third cRV
RVs[is_CRV] <- paste0(RVs[is_CRV],c("*","**","***"))
top10RVs <-  matrix(nrow=10,ncol=ncol(allranks))
colnames(top10RVs) <- colnames(allranks)
for(i in 1:ncol(top10RVs)){
  ord <- order(allranks[,i])
  top10RVs[,i] <- RVs[ord][1:10]
}
```

```
options(width=100)
noquote(top10RVs)
```

```
##       globalLR1to6 globalLR7    globaltrans  localLR      RVS          modRVS
##  [1,] 8_39680176   8_39680176   8_39680176   8_39680176   8_65161224   8_39680176
##  [2,] 8_23020454*  8_23020454*  8_23020454*  8_23020454*  8_39680176   8_65161224
##  [3,] 8_7972232    8_7972232    8_7972232    8_7972232    8_23020454*  8_23020454*
##  [4,] 8_23068413** 8_23068413** 8_23068413** 8_23068413** 8_8376739    8_8376739
##  [5,] 8_65161224   8_65161224   8_65161224   8_65161224   8_12116292   8_12116292
##  [6,] 8_8376739    8_8376739    8_8376739    8_8376739    8_30846116   8_30846116
##  [7,] 8_54047456   8_60866956   8_54047456   8_54047456   8_41261959   8_41261959
##  [8,] 8_107249922  8_103382957  8_107249922  8_107249922  8_7972232    8_7972232
##  [9,] 8_60866956   8_54047456   8_60866956   8_60866956   8_23068413** 8_23068413**
## [10,] 8_103382957  8_107249922  8_103382957  8_103382957  8_60866956   8_60866956
```

- The raw ranks of the three cRVs sampled in the study are shown below.

```
cRVs <- unique(a_seqs$haplo_map$FamCRV) # in peds 1, 2 and 3, respectively
allranks[rownames(allranks)%in%cRVs,]*nrow(allranks)
```

```
##              globalLR1to6 globalLR7 globaltrans localLR RVS modRVS
## 8_23020454              2         2           2       2   3      3
## 8_23068413              4         4           4       4   9      9
## 8_118923860            25        25          49      15  40     29
```

- We can also look at the summaries of the three cRVs.

```
allele_summary[rownames(allele_summary) %in% cRVs,]
```

```
##              chrom  position NHL1 HL1 tot1 NHL2 HL2 tot2 NHL3 HL3 tot3
## 8_23020454       8  23020454    0   2    2    0   0    0    0   0    0
## 8_23068413       8  23068413    0   0    0    0   2    2    0   0    0
## 8_118923860      8 118923860    0   0    0    0   0    0    0   1    1
```

- We see three cRVs, each of which is unique to a pedigree.
- The cRVs 8_23020454 from family 1 and cRV 8_23068413 from family 2 had top-10 ranks from all methods. From the diagram we see that these cRVs are in two HL-affected individuals who are second cousins (8_23020454, family 1) and first cousins (8_23068413, family 2), respectively.
- The cRV 8_118923860 from pedigree 3 did not have a top-10 ranking from any of the methods. This RV is only observed in one HL-affected individual.

3.2 P-values for sampled cRVS

- For each familial cRV, we find its p-value from the lookup tables.

```
cRVs <- unique(a_seqs$haplo_map$FamCRV)
pvals <- get_pvals(cRVs,lookupTabs,a_seqs) # *worker*
pvalcols <- c(paste0("globalLR",1:length(carrier_probs)),
              paste0("globaltrans",1:length(carrier_probs)),
              "localLR_1","RVS_1","modRVS_1")
colnames(pvals) <- pvalcols
rownames(pvals) <- cRVs
pvals
```

```
##                  globalLR1     globalLR2    globalLR3     globalLR4     globalLR5     globalLR6
## 8_23020454  0.00560338992 0.00560364111 0.0056047586 0.00560615796 0.00560896508 0.00563182814
## 8_23068413  0.01741064834 0.01741246720 0.0174205551 0.01743067449 0.01745094479 0.01761472332
## 8_118923860 0.35321461030 0.35323875975 0.3533461167 0.35348037265 0.35374908400 0.35590884717
##                  globalLR7 globaltrans1  globaltrans2 globaltrans3  globaltrans4  globaltrans5
## 8_23020454  0.00594579242 0.0056033936 0.00560367789 0.0056049426 0.00560652625 0.00560970278
## 8_23068413  0.02011330041 0.0174107042 0.01741302581 0.0174233496 0.01743626720 0.01746214491
## 8_118923860 0.38832427758 0.6143686747 0.61436915669 0.6143713052 0.61437400553 0.61437945500
##               globaltrans6  globaltrans7    localLR_1        RVS_1     modRVS_1
## 8_23020454  0.00563556167 0.00598609338 0.0087479497 0.0180426463 0.0087479497
## 8_23068413  0.01767122570 0.02029433669 0.0414507772 0.0777202073 0.0414507772
## 8_118923860 0.61442541797 0.61526246323 0.3177339901 0.7093596059 0.3152709360
```

- The cRVs 8_23020454 (from ped 1) and 8_23068413 (from ped 2) were observed in two HL-affected individuals and have reasonably small p-values (e.g., 0.0056 and 0.0174, respectively by the global methods).
- The cRV 8_118923860 was observed in only one affected individual in pedigree 3 and so, not surprisingly, has large p-values (> 0.31).

4 Appendix: Worker functions

- Here we document the worker functions written specifically for this example data analysis. The functions are sourced into the above analysis from the source file `workerfuncs.R`.

- `workerfuncs.R` also includes functions from R scripts in the simulation study workflow. These other functions are documented in their respective .Rmd files.

- The first worker function written for this analysis removes RVs whose configurations are incompatible with the assumption that RVs enter a pedigree through at most one founder.

- Christina calls such configurations “invalid”, and they have been removed from the rows of the lookup tables of p-values and statistics output by `cd_new()`.
- For each RV, we see if its configuration is in the lookup tables, and if not we remove it from the dataset.

```r
remove_invalid_configs <- function(a_seqs,lookupTabs) {
  # configs are identified in the lookup table by their base-10
  # representation, or "binID"
  valid_binIDs <- lookupTabs$statvals[,"binID"]
  # initialize an empty vector to hold info on which RV configs are valid
  include <- rep(NA,nrow(a_seqs$SNV_map))
  # loop over RVs
  for(i in 1:length(include)) {
    RV_binID <- find_binID(a_seqs$SNV_map$marker[i],a_seqs)
    include[i] <- (RV_binID %in% valid_binIDs)
  }
  # Use the include vector to subset the input a_seqs and return
  out <- list(ped_files=a_seqs$ped_files,
              ped_haplos=a_seqs$ped_haplos[,include],
              haplo_map=a_seqs$haplo_map,
              SNV_map=a_seqs$SNV_map[include,],
              ped_genos=a_seqs$ped_genos[,include])
  return(out)
}
```

- The second worker function is a modified version of Christina’s `summary.famStudy()` function.
- `famStudy` objects are the output of `sim_RVstudy()` and `summary.famStudy()` is a summary function that returns the counts of each variant in the affected members of each pedigree.
- My modification is to split the counts of each variant by disease subtype.
  - Note: This is a *very* quick-and-dirty function that assumes that there are three families in the study.

```r
mysummary.famStudy <- function (object, ...)
{
  Fids <- sort(unique(object$ped_files$FamID))
  # The following 6 lines are from Christina's summary function.
  # They get the total allele count in affected individuals
  aff_allele_counts <- lapply(Fids, function(x) {
    SimRVSequences:::affected_allele_count(
      ped_haps = object$ped_haplos[object$haplo_map$FamID == x, ],
      hap_map = object$haplo_map[object$haplo_map$FamID == x, ],
      ped_file = object$ped_files[object$ped_files$FamID == x, ])
  })
  tot_allele_count <- do.call(rbind, aff_allele_counts)
  totals <- colSums(tot_allele_count)
  # I now do the analogous thing to ge the alelle counts in those
  # affected with NHL
  aff_allele_counts <- lapply(Fids, function(x) {
    affected_allele_count_NHL(
      ped_haps = object$ped_haplos[object$haplo_map$FamID == x, ],
      hap_map = object$haplo_map[object$haplo_map$FamID == x, ],
      ped_file = object$ped_files[object$ped_files$FamID == x, ])
  })
  NHL_allele_count <- do.call(rbind, aff_allele_counts)
  # Finally, get the alelle counts in those affected with HL
  aff_allele_counts <- lapply(Fids, function(x) {
    affected_allele_count_HL(
      ped_haps = object$ped_haplos[object$haplo_map$FamID == x, ],
      hap_map = object$haplo_map[object$haplo_map$FamID == x, ],
      ped_file = object$ped_files[object$ped_files$FamID == x, ])
  })
  HL_allele_count <- do.call(rbind, aff_allele_counts)
  # We can now put together our counts and return the results
  fam_allele_count <- cbind(
    NHL_allele_count[ 1,],HL_allele_count[1,],tot_allele_count[1,],
    NHL_allele_count[2,],HL_allele_count[2,],tot_allele_count[2,],
    NHL_allele_count[3,],HL_allele_count[3,],tot_allele_count[3,]
  )
  colnames(fam_allele_count) <-
    c("NHL1","HL1","tot1","NHL2","HL2","tot2","NHL3","HL3","tot3")
  rownames(fam_allele_count) <- object$SNV_map$marker
  out <- data.frame(
    chrom = object$SNV_map$chrom,
    position = object$SNV_map$position,
    fam_allele_count)
  # only return RVs present in at least one affected individual
  return(out[totals>0,])
}

# Worker functions to count variants in NHL and HL subtypes. These are
# based on Christina's affected_allele_count() function.
affected_allele_count_NHL <- function (ped_haps, hap_map, ped_file)
{
  aff_IDs <- ped_file$ID[ped_file$affected & ped_file$subtype=="NHL"]
  aff_rows <- which(hap_map$ID %in% aff_IDs)
  total_count <- colSums(ped_haps[aff_rows, ])
  return(total_count)
}
affected_allele_count_HL <- function (ped_haps, hap_map, ped_file)
{
  aff_IDs <- ped_file$ID[ped_file$affected & ped_file$subtype=="HL"]
  aff_rows <- which(hap_map$ID %in% aff_IDs)
  total_count <- colSums(ped_haps[aff_rows, ])
  return(total_count)
}
```

- The third worker function reorders the rows of the simulated genotypes matrix. The rows of the genotypes matrix correspond to affected individuals with sequencing data, the columns to RVs and the entries to genotypes. The function `reorder_genos()` orders the rows of the genotypes matrix to be the same as the in the configurations of the lookup tables. With the rows of the genotypes matrix ordered this way, we can read the global configuration of any RV straight from its column in the matrix. The order of affected individuals in the configurations of the lookup tables is in the variable `binKey` returned by `cd_new()`.
  - **Note:** This function is also in the simulation scripts for the statistic and p-value lookups.

```
reorder_genos <- function(a_seqs,binKey) {
  a_seqs$ped_genos <- a_seqs$ped_genos[binKey,]
  return(a_seqs)
}
```