The R-TIGER package

Rafael Campos-Martin

Contents

1 Introduction

Meiotic recombination is an essential mechanism in the sexual reproduction of diploid eukaryotic organisms that consists of the overcrossing (chiasmata) between pairs of homologous chromosomes [REF]. This process ensures equitable segregation of the genomic material in each gamete. Due to this process, new combinations of alleles are transmitted to new generations allowing genetic variability among individuals and benefits the adaptation of populations. Recombination occurs in the so-called meiosis I and is triggered by double strand breaks (DBSs). The DBSs can be repaired by crossover with the homologous chromosome (CO) or by the same chromosome (NCO). The CO/NCO ratio is controlled in each chromosome and taxa and the mechanism by which it is determined the proportion of DSBs is resolved as CO is unknown [REF].

In plants, the distribution of CO along the chromosome is largely random, with some locations such as the centromere being disfavored, and some CO hotspots [REF] . A better understanding of the crossing over process is essential for various branches of molecular biology such as medicine, agriculture, etc. [REF] The aim of this package is to offer a possible solution for the genotyping of recombinant populations, generated from ancestors with known genotype, by sparse sequencing of their genomes.

2 Genotyping seven sequencing samples from Arabidopsis Thaliana

As an example, we will show how to analyze the genotype of seven different plants of *Arabidopsis Thaliana*. The bam files have been previously processed to extract the SNP information at each position. Our package works with txt files which contain six columns. The first column defines the chromosome, second the position of the SNP third the allele of one of the parents, the read counts that support such allele, the allele of the second parent and the allele that supports these second allele.

```
library(RTIGER)
```

```
## Loading required package: GenomicRanges

## Loading required package: stats4

## Loading required package: BiocGenerics

## Loading required package: parallel
```

```
##
## Attaching package: 'BiocGenerics'

## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB

## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs

## The following objects are masked from 'package:base':
##
##     anyDuplicated, append, as.data.frame, cbind, colMeans,
##     colnames, colSums, do.call, duplicated, eval, evalq, Filter,
##     Find, get, grep, grepl, intersect, is.unsorted, lapply,
##     lengths, Map, mapply, match, mget, order, paste, pmax,
##     pmax.int, pmin, pmin.int, Position, rank, rbind, Reduce,
##     rowMeans, rownames, rowSums, sapply, setdiff, sort, table,
##     tapply, union, unique, unsplit, which, which.max, which.min

## Loading required package: S4Vectors

##
## Attaching package: 'S4Vectors'

## The following object is masked from 'package:base':
##
##     expand.grid

## Loading required package: IRanges

## Loading required package: GenomeInfoDb

## Loading required package: TailRank

## Loading required package: oompaBase
```

```
path = system.file("extdata",  package = "RTIGER")
files = list.files(path, full.names = T)[1:3]

file.example = read.delim(files[1], header = FALSE)
head(file.example)
```

```
##   V1     V2 V3 V4 V5 V6
## 1  1  38554  G  0  T  1
## 2  1  71326  G  0  T  3
## 3  1  71348  C  0  T  5
## 4  1  96770  C  0  A 10
## 5  1 100847  T  0  A  2
## 6  1 105064  C  0  T  4
```

2.1 Initialize

In order to initialize our model we need a data frame with minimum two columns. The files columns, which contains the full path to the files of each sample and the name column, which has a shorter name by which we will refer to each sample. In our example, we will use the file name as name of the sample:

```
expDesign = data.frame(files = files, name = as.character(sapply(files, function(samp)  unlist(strsplit
```

```
expDesign
```

```
##
## 1 /home/campos/R/x86_64-pc-linux-gnu-library/3.4/RTIGER/extdata/input_corrected3647_AA_run513_GTAGAG
## 2 /home/campos/R/x86_64-pc-linux-gnu-library/3.4/RTIGER/extdata/input_corrected3647_AC_run513_GTAGAG
## 3   /home/campos/R/x86_64-pc-linux-gnu-library/3.4/RTIGER/extdata/input_corrected3647_B_run513_CGAGG
##                                                 name
## 1 input_corrected3647_AA_run513_GTAGAGGA_S20_L007
## 2 input_corrected3647_AC_run513_GTAGAGGA_S21_L007
## 3   input_corrected3647_B_run513_CGAGGCTG_S3_L007
```

This together with the sequence length of each chromosome of our organism is all we need to start the package. Moreover, we will need to decide some thresholds by which the chromosomes will be binned in order to increase the sequencing counts at each position.

```
library(Rpackages, quietly = TRUE)
```

```
##
## Attaching package: 'Rpackages'
```

```
## The following objects are masked from 'package:RTIGER':
##
##     analysis.samples, calcCOnumber, calcDoubleCOnumber,
##     DataSetImportFromtxt, fitModel, haplotypes.width,
##     plotGenotype, R.Viterbi, width.DoubleCO
```

```
data("ATseqlengths")


myDat = DataSetImportFromtxt(experimentDesign = expDesign,
                             bin.length = 200,
                             min.samples = 2,
                             quant = .2,
                             min.counts = 5,
                             seqlengths = ATseqlengths)
```

```
## [1] "Loading file:  /home/campos/R/x86_64-pc-linux-gnu-library/3.4/RTIGER/extdata/input_corrected364
## [1] "Loading file:  /home/campos/R/x86_64-pc-linux-gnu-library/3.4/RTIGER/extdata/input_corrected364
## [1] "Loading file:  /home/campos/R/x86_64-pc-linux-gnu-library/3.4/RTIGER/extdata/input_corrected364
## [1] "matrix"
## New bin length =  400
## New bin length =  600
## New bin length =  800
## New bin length =  1000
## New bin length =  1200
## New bin length =  1400
## New bin length =  1600
## New bin length =  1800
## New bin length =  2000
## New bin length =  2200
## New bin length =  2400
## New bin length =  2600
## New bin length =  2800
## New bin length =  3000
## New bin length =  3200
## New bin length =  3400
## New bin length =  3600
## New bin length =  3800
## New bin length =  4000
## New bin length =  4200
## New bin length =  4400
## New bin length =  4600
## New bin length =  4800
## New bin length =  5000
## New bin length =  5200
## New bin length =  5400
## New bin length =  5600
## New bin length =  5800
## New bin length =  6000
## New bin length =  6200
## New bin length =  6400
```

In our initialization, we have specified that the bins should start with 200 nt and increase if the minimum condition does not hold. The condition we have used is that only 20% (quant) of the observations should have less than 5 (min.counts) counts. Also, each position must be supported by at least 2 samples to be taken in account. This functions generates an object of class "RTIGER". A quick look into it will give us some information about the sample names, chromosomes and number of observations per chromosome:

```
myDat
```

```
## An object of class "RTIGER"
##  Number of samples:  3
## Sample names(3): input_corrected3647_AA_run513_GTAGAGGA_S20_L007
##   input_corrected3647_AC_run513_GTAGAGGA_S21_L007
##   input_corrected3647_B_run513_CGAGGCTG_S3_L007
##  Number of chromosomes per sample: 5
##  Chromosome names: 1 2 3 4 5
##  Number of observations per chromosome: 3791 2558 2923 2269 3479
```

2.2 Analize samples

Our threshold previously set, is applied to all the observations as a whole. But does it apply to each individual sample? What is the library size for each sample? We can learn a bit more about our samples using a simple function:

The three barplots in red are samples that do not pass the threshold set for all observations. That means that their quantile 20% is lower that 5. This could create problems in further analysis since lower library size will produce higher number of false positives when we want to infer the number of COs.

Since this is an example, we will continue with those samples.

2.3 Fitting the model

The next step is to fit the parameters of our R-TIGER model. An HMM based model specific for gentoyping. For sake of time we will only use 1 iteration, but in practical cases it should run till convergence:

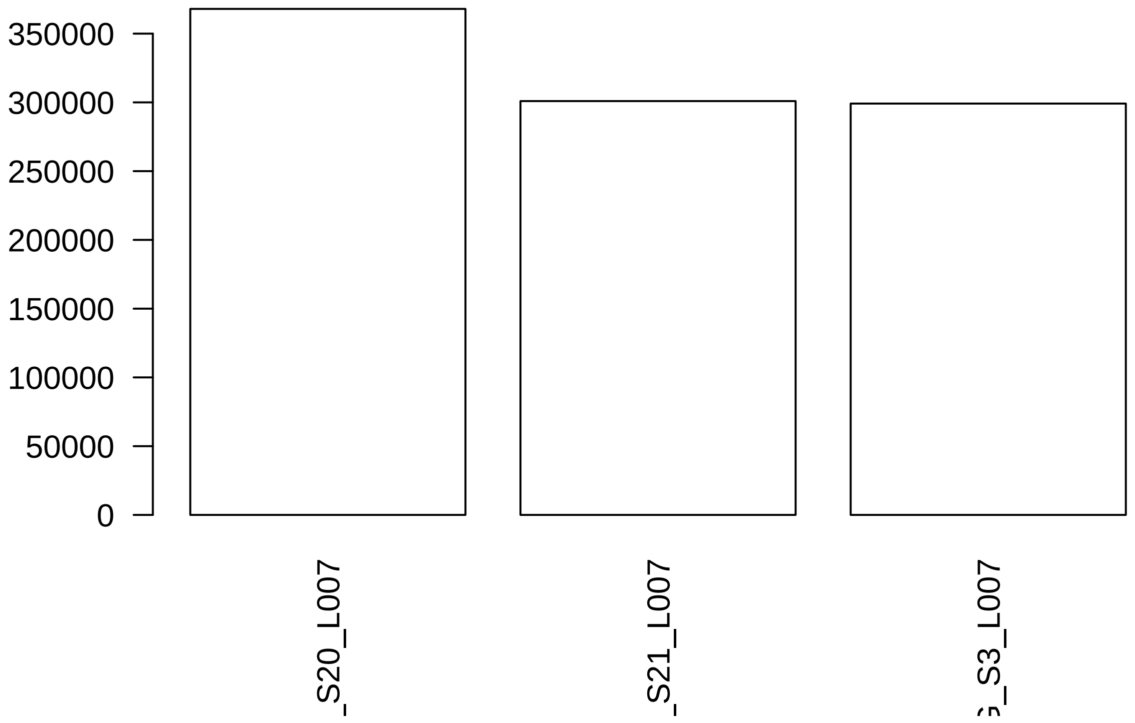
```
myDat = fitModel(myDat,
                 max.iter = 1,
                 verbose = TRUE)
```

```
## Iteration:  1
## Absolute difference of the parameters after maximization:  1940.223
## Marker state distribution:
##
##           +        badM        badP
## 0.969174434 0.005858855 0.024966711
##
##
## Reached maximum number of iterations (1) without convergence.
## Viterbi decoding...
```

```
myAnalysis = analysis.samples(myDat)

myAnalysis$lib.size
```

```
## input_corrected3647_AA_run513_GTAGAGGA_S20_L007
##                                          367984
## input_corrected3647_AC_run513_GTAGAGGA_S21_L007
##                                          300929
##   input_corrected3647_B_run513_CGAGGCTG_S3_L007
##                                          299129
```

```
barplot(myAnalysis$lib.size, col = (!myAnalysis$passThresh) +1, las = 2)
```

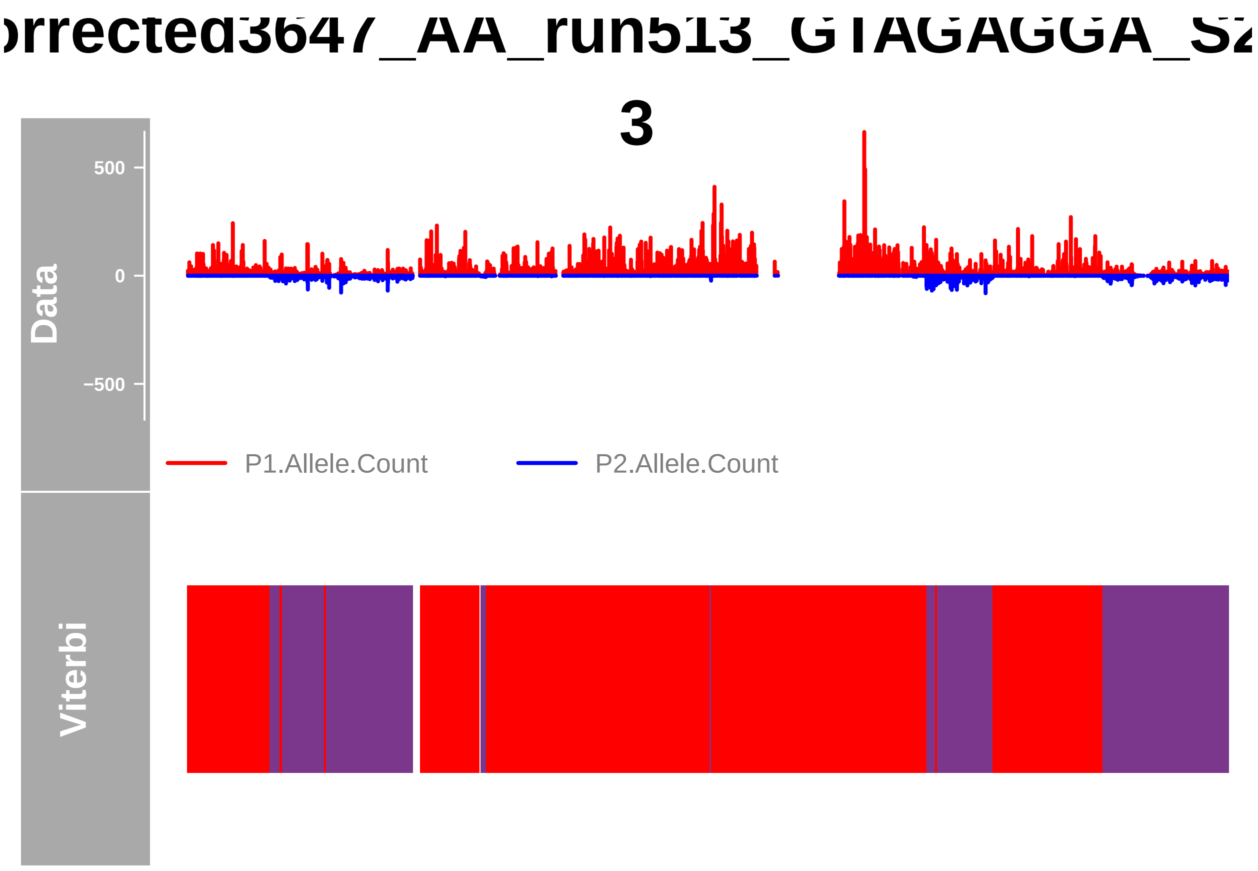


2.4 Plotting

During the fitting the single observations have been decoded into the 3 possible states: homozygous parent one, homozygous parent two or heterozygous. We can plot the result for chromosome 3 of the sample input_corrected3647_AA_run513_GTAGAGGA_S20_L007

```
samp = "input_corrected3647_AA_run513_GTAGAGGA_S20_L007"
chr = 3
```

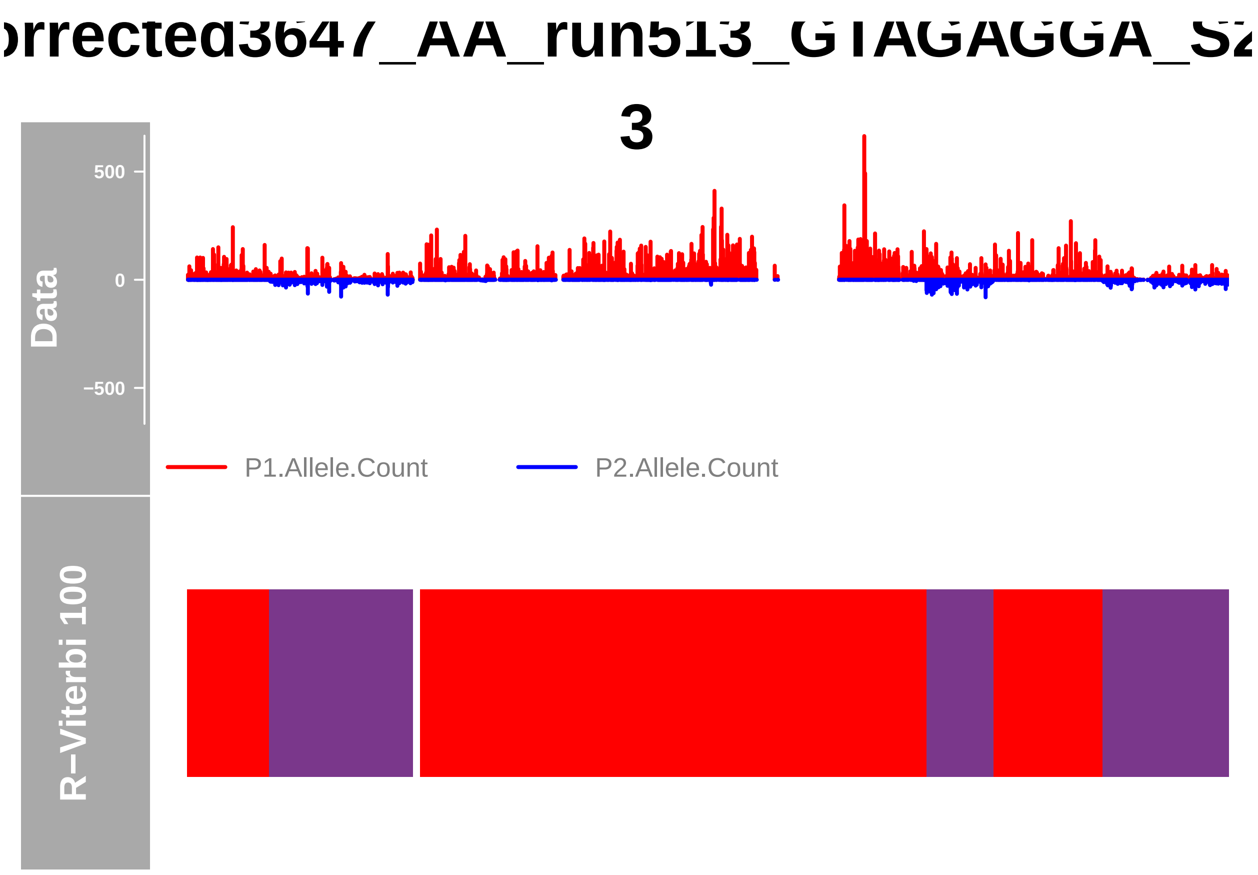
```
plotGenotype(myDat, samp = samp, chr = chr)
```



2.5 R-Viterbi algorithm

Nevertheless, this is not an optimal result. There are more jumps than expected in one simple chromosome. This is a known problem of this type of data. We have implemented a new version of the viterbi algorithm in which we force to stay in a single state r times before being possible to jump to another state.

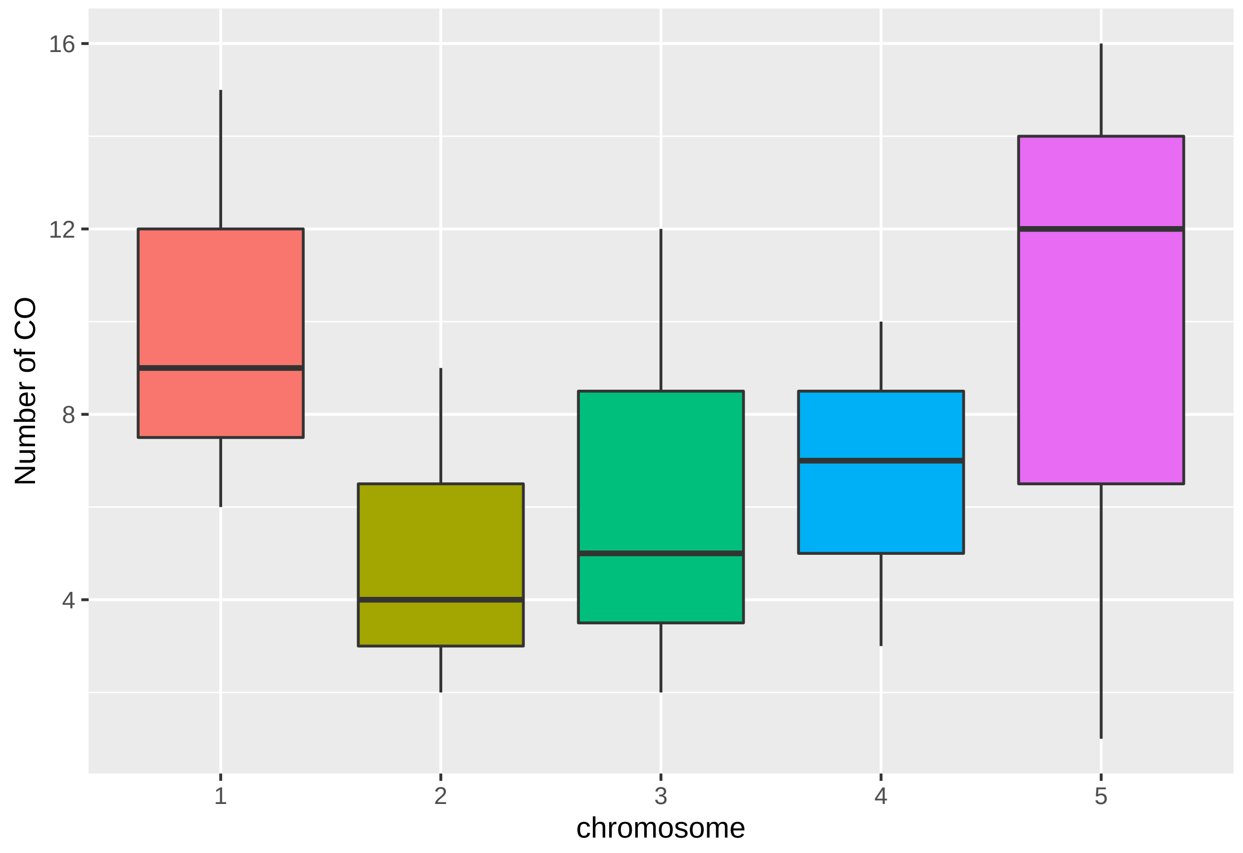
```
rviterbi = R.Viterbi(myDat, rigidity = 100)

plotGenotype(rviterbi, samp = samp, chr = chr)
```



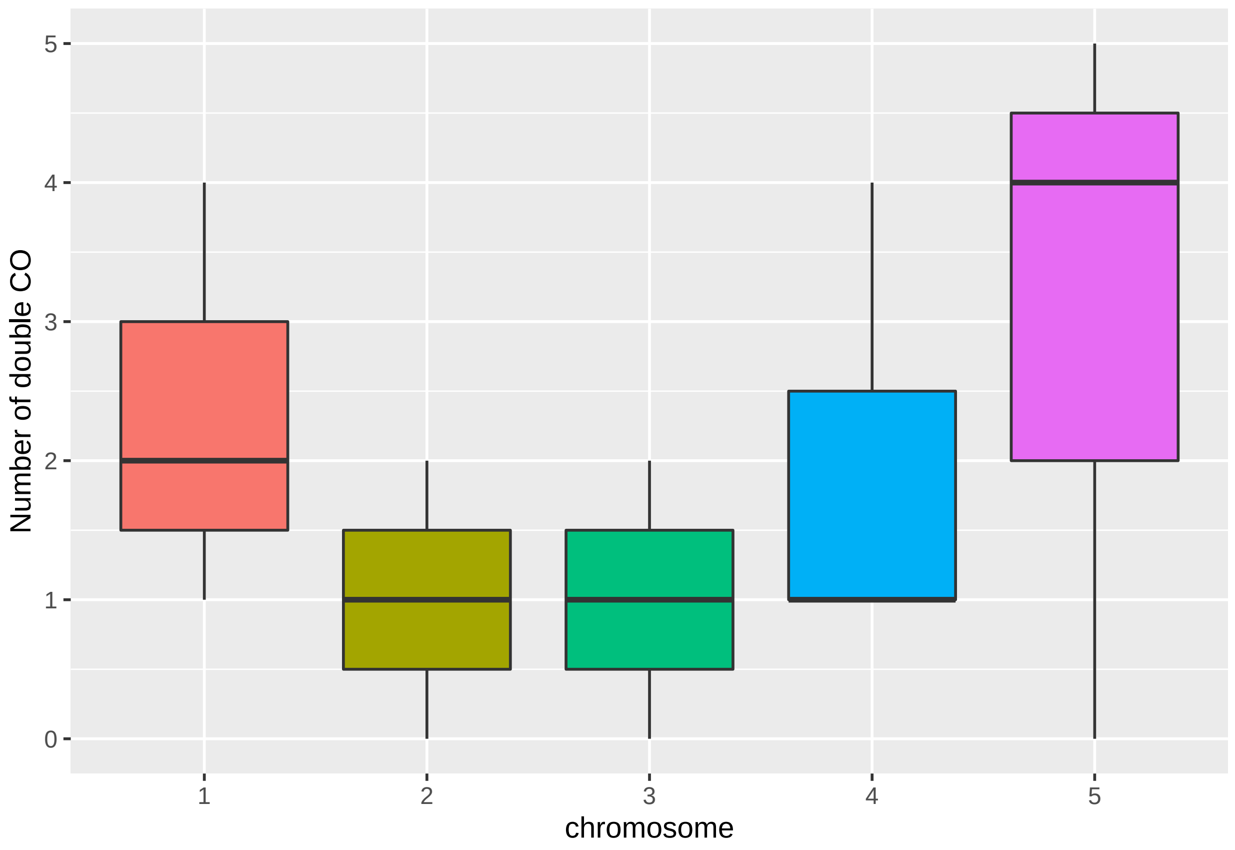
2.6 Number of Chrosover ovents per chromosomome

Our next goal is to study the number of CO events and double CO (dCO). We define the pure dCO those which are heterogeneous and have the same paternal genotype at each end. When this pattern is observerd, it means that there has been a double cross over in a single chromosome.

```
CO.number = calcCOnumber(rviterbi)

library(reshape2)
library(ggplot2)

Co.number.melt = melt(CO.number)

ggplot(Co.number.melt, aes( x = factor(Var1), y = value, fill = factor(Var1))) +
  geom_boxplot() +
  theme(legend.position = "none")+
  xlab("chromosome") +
  ylab("Number of CO")
```



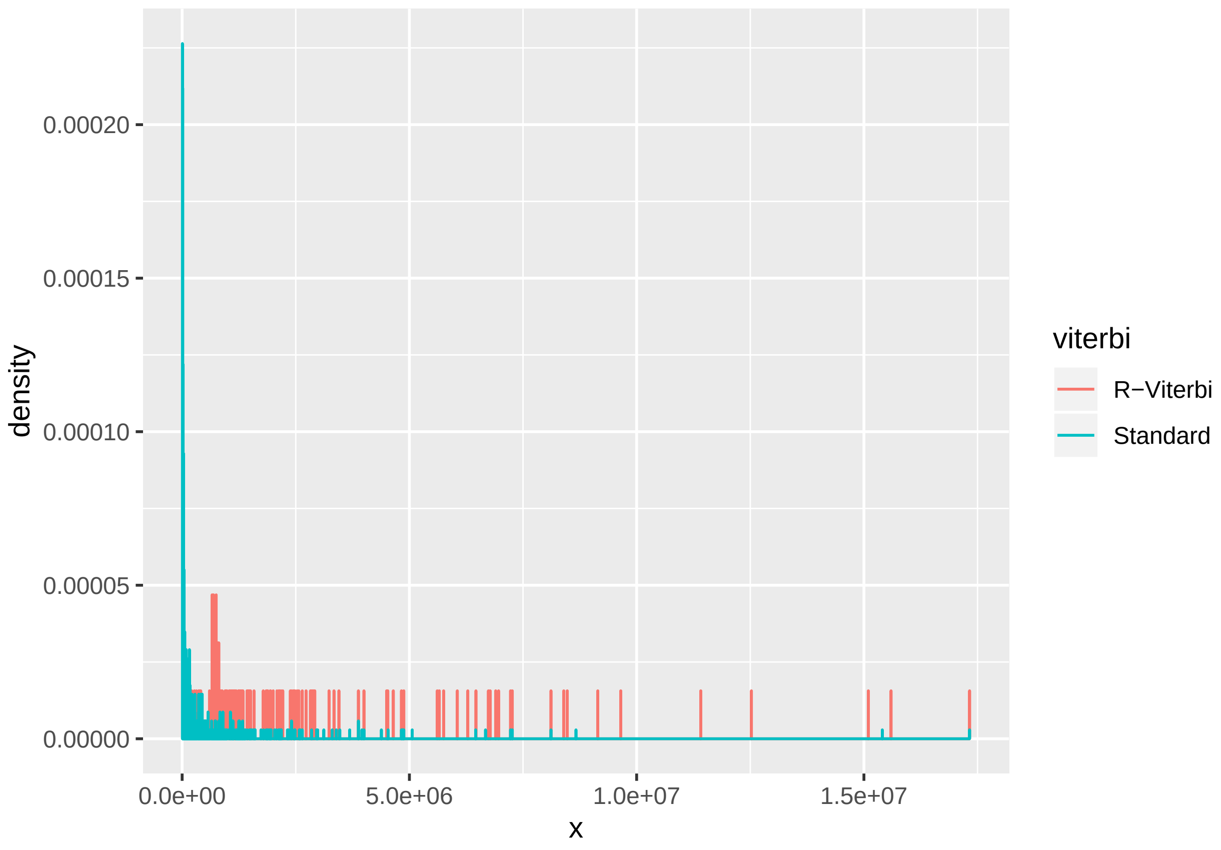
```
dco.number = calcDoubleCOnumber(rviterbi)
dco.number.melt = melt(dco.number)

ggplot(dco.number.melt, aes( x = factor(Var1), y = value, fill = factor(Var1))) +
  geom_boxplot() +
  theme(legend.position = "none")+
  xlab("chromosome") +
  ylab("Number of double CO")
```



2.7 CO widths

Another interesting information that we can obtain is the widht of each haplotype:

```
CO.widths = haplotypes.width(rviterbi)

CO.widths = data.frame(x = unlist(CO.widths), viterbi = rep("R-Viterbi", length(unlist(CO.widths))))

CO.HMM.widhts = haplotypes.width(myDat)

CO.HMM.widhts = data.frame(x = unlist(CO.HMM.widhts), viterbi = rep("Standard", length(unlist(CO.HMM.wid

comp.Table = rbind(CO.widths, CO.HMM.widhts)

ggplot(comp.Table, aes(x, stat(density), colour = viterbi)) +
  geom_freqpoly(binwidth = 500)
```



As we can observe the standard HMM viterbi algorithm produces more CO and shorter. This is not the expected behaviour