Amplicomsat: Scoring microsatellite genotypes from amplicon sequencing and eDNA samples

Filipe Alberto

2025-02-10

Background

Amplicon sequencing represents an efficient way to genotype microsatellite markers, allowing for increased multiplexing of samples and markers. A particular case where amplicon sequencing is favorable is when more than two microsatellite alleles occur in each sample locus combination. Examples of such samples are forensic analysis with possibly more than one offender DNA mixed or eDNA analysis where several individual microscopic life stages might be present in each sample (e.g., kelp gametophytes). Amplicomsat can output the observed (sequenced) counts of alleles (OAC) produced from such forensic or eDNA samples. Alleles can be scored based on their fragment size or sequence variation. Genotype tables for classical microsatellite analysis can also be produced. A variety of diagnostic plots are created throughout the pipeline.

Required files:

Two set-up files and one directory, where sample fastq files are saved, are required.

1) A tab-delimited file with primer information saved in the working directory is required. The format is shown in Table 1 for an example with 11 markers. The *nmotif* (last column) indicates how many microsatellite sequence motif repeats a read should have to be retained. Consider changing this for different types of microsatellites (di-nucleotides, three-nucleotides, might require larger numbers).

Table 1: Argument LocusInfo takes a table with this format, containing loci name, FWD and RVS primer sequence, and microsatellite repeat motif

Locus	Fprimer	Rprimer	Motif	nmotif
Nl21	AACGCGTCATAAATACATACCAGGT	AACCTGCGAATCCAAATTGCACGCT	ACAT	2
Nl24	TGGGAGCAAGGAATTAGGACGACCC	AGGTACATTCTTGTTCGTCTTGGTT	AGAT	2
Nl25	CGGTGACCGTAAGCGAAGAGCCTTC	TAGCGCCATCGTGAGGTTAACCTGC	CCTG	2
Nl26	CGTCAACAATTTGAAAGTGACACCC	AGCAACAGGATCTAGGCTGTAACCT	ATCC	2
Nl27	ACCTCTGCTGTGAACCGGAGAAACGA	AACAGAGGCGGTGAGAGCGAGCTAA	ACAT	2
Nl30	GAAGCATCTCCACTGTGCGACCGAG	TCGTGCTCATTTCGTCTTCTTCCTT	GTTT	2
Nl40	CTTCTTGCCGCCTTTGCCCTAGCTC	TTGTAGGGAGATAAGGCGTGCGGGA	GCCT	2
Nl47	ACTGTAGCGCCTGAAGCTTCTGAGGT	ACGTAACCCGCCGTCAATAGG	AAAC	2
Nl50	CGAGATGAACAACTCCGCGGCGAAA	GGGTTACCAGGTACCGGGCAGTCAA	GCCT	2
Nl59	TGCTGATAATGACTGCTGTTTGCTCC	GAGAGCGGTAGTGGCAAACGAACGG	GCTT	2
Ner2	GAAGAGGTGCGGTGGCTT	GGAATCGGAACCCAAAATTAGT	ATTCGG	2

2) A calibration file for the allele-scoring algorithm, saved as a tab-delimited file in the R working directory. The example in Table 2 shows the first two columns (loci) of a calibration file needed for running the *GenotypeModel* function. The default name for the file is "CalibrationSamples.txt".

Table 2: Example of the first two columns (loci) in the calibration file

Nl21	Nl24
100-240	100-210
KELP0004_S4.extendedFrags.F.fastq	KELP0005_S5.extendedFrags.F.fastq
KELP0013_S12.extendedFrags.F.fastq	KELP0010_S9.extendedFrags.F.fastq
KELP0014_S13.extendedFrags.F.fastq	KELP0019_S18.extendedFrags.F.fastq
KELP0015_S14.extendedFrags.F.fastq	KELP0026_S25.extendedFrags.F.fastq
KELP0017_S16.extendedFrags.F.fastq	KELP0066_S66.extendedFrags.F.fastq
KELP0018_S17.extendedFrags.F.fastq	KELP0067_S67.extendedFrags.F.fastq
110	100

The first row should have the loci names. The loci names should be written exactly as in the lociInfo object in Table 1. The second row has the range in bp to look for alleles. Here, the advice is to balance a too-narrow range that might miss alleles and a too-broad range that will slow down the code. The following rows contain file names for samples that represent a homozygous pattern. Utilizing a few (4-6) to calibrate the algorithm is better to average out any minor differences between the relative frequency of sequences with different sizes that constitute the "phenotype" of a single microsatellite allele. The last row should contain a coverage threshold for each loci. Filtered reads of a given length with coverage below this value will not be counted as possible alleles. However, to identify homozygous samples for each locus, the appropriate coverage thresholds, and the allele ranges, we still need to visualize the amplification patterns for each locus before completing this calibration file.

Samples directories required

By default, Amplicomsat expects a directory written in the working directory, named "samples," containing the fastq files. Amplicom sequencing of tandem repeats like microsatllites are normally sequenced using paired-end reads, thus, we first use *pear* (https://cme.h-its.org/exelixis/web/software/pear/doc.html) to merge read 1 with read 2 into a single fastq read (see example below).

eDNA or mixed samples, where multiple alleles (possibly more than two) are expected per sample loci combination, should be saved in a directory written in the R working directory called, by default, "samplesF.MA" (more details below).

When using Amplicomsat, almost no objects are written to the R session. The code mostly writes and reads files from the working directory and creates a few new directories in there, too:

Directories created:

"samplesF" (filtered fastq reads) created by function *read.quality.filter* containing fastq files filtered for sequence quality. A file named "filterLog" is also written by function *read.quality.filter* with the proportions of reads filtered out for each sample.

The directory "ReadSizesRepeatNumbers" contains files with information on the size of each read, filtered by sample and loci combination, and the number of msat motif repeats for all reads where the two primers and a user-selected number of msat motif repeats are found. The files in this folder are for internal use of Amplicomsat functions.

The directory "Plots" will contain plots with the distribution of reads along the range, as created by function *reads.plots*. A single plot per locus with all samples is produced. Two plots per sample; in addition to the above plot, a second plot with read size in bp vs number of msat motif repeat number can be produced (optionally).

The directory "Plots.Scored", containing plots similar to those described for the directory "Plots" but where a different color is used to mark the scored alleles.

The directory "ModelTables" will contain tables with expected relative frequencies of reads for different combinations of possible diploid genotypes. The files in this folder are for internal use of the functions scoring alleles.

The directory "GenotypeOut" will contain the genotype tables in two formats (one or two columns per locus).

The directory "M.plots" will contain the read peak distribution for samples scored for multiple alleles (eDNA or mixed samples). Alleles scored are marked in these plots with blue color.

The directory "AllelesBySeq" will contain a) a genotype file for sequence-variation-based allele scoring (Genotypes.by.Sequence.txt); b) a database of allele sizes, sequences, and numerical code translation (Allele DataBase.txt); and a file with a genotype table coded using numerical codes, as most pop gen software requires this (Genotype Codes.txt). The correspondence between sequence-based, size-only, and numerical codes is available in the Allele DataBase.txt file.

Most of these folders contain information of little interest to users. The "Plots," "M.plots," "GenotypeOut," and "AllelesBySeq" directories contain information for users.

Running the Amplicomsat pipeline

Before we get into R and use Amplicomsat, there are a couple of set-up steps recommended to deal with Illumina pair-ended reads. First, we remove the Nextera sequencing adapters using *fastp*. In our experience, removing the adapters resulted in more reads being merged in the next step. Second, we will use *pear* to merge the pair-ended fastq files into a single fastq file per sample. This assumes that you received fastq files already demultiplexed per sample.

Removing the Nextera sequencing adapters

The shell code below uses fastp to trim the Nextera adapters. Note the use of *–adapter_sequence* for read 1 and *–adapter_sequence_r2* for read 2. We first show the example for a single sample and then for all samples, using gnu parallel, where *qzfiles* is a text file containing in each row the unique part of sample file names that will be replaced in the {} fieds.

Single sample

```
 fastp --cut_tail --adapter_sequence TCGTCGGCAGCGTCAGATGTGTATAAGAGACAG \
--adapter_sequence_r2 GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAG \
-i KELP0001_S1_L001_R1_001.fastq -o ./trim/KELP0001_S1_L001_R1_001.trim.fq.gz \
-I KELP0001_S1_L001_R2_001.fastq -O ./trim/KELP0001_S1_L001_R2_001.trim.fq.gz
```

All samples

```
cat qzfiles | parallel -j 4 fastp --cut_tail
--adapter_sequence TCGTCGGCAGCGTCAGATGTGTATAAGAGACAG \
--adapter_sequence_r2 GTCTCGTGGGCTCGGAGATGTGTATAAGAGACAG \
-i {}_L001_R1_001.fastq.gz -o ./trim/{}_L001_R1_001.trim.fq.gz \
-I {}_L001_R2_001.fastq.gz -O ./trim/{}_L001_R2_001.trim.fq.gz
```

Using *pear* to merge paired-end reads

Run this in the shell after installing pear

```
pear -g 2  -j 4 -f  ./trim/KELP0001_S1_L001_R1_001.trim.fq.gz \
-r KELP0001_S1_L001_R2_001.trim.fq.gz -o KELP0001
```

#The parallel way to run all samples at once.

```
cat qzfiles | parallel  pear -g 2  -j 4 -f  ./trim/{}_L001_R1_001.trim.fastq.gz \
-r ./trim/{}_L001_R2_001.trim.fastq.gz -o {}
```

We can now move into R

Amplicomsat package installation

The Amplicomsat package can be installed from GitHub

```
# We need package devtools to install from GitHub
library(devtools)
install_github("UWMAlberto-Lab/Amplicomsat")
```

We can now load the package

```
# Loading the package
library(Amplicomsat)
```

Filtering reads for Phred 30 or higher

```
read.quality.filter(sampleDir = "./samples", Phred.threshold = 30,
    All.below.Q = 5, filterDir = "samplesF")
```

The fastq merged files will be read and the number of bases with quality below *Phred.threshold* counted (defaults to Q30). Reads will be retained if containing *All.below.Q* (defaults to 5) or fewer bases below the *Phred.threshold*. The filtered reads are written in a " sampleF " directory created by the function in the R working directory. A log file, "filterLog" is written to the working directory with information on the number and proportion of reads removed. Finally, a pdf is produced showing the distribution of bases in each read in each sample that are below *Phred.threshold*.

Finding the reads with primers and microsatellite repeat motif

The next step is to find reads for each primer that contain the microsatellite motif (repeated *nmotif* times) while saving some summary stats (read size and repeat number).

```
read.size.freqs(locusInfo = "primers", sampleDir = "./samplesF")
```

The *locusInfo* is the path to a file containing loci names: fwd primers, rvs primers, microsatellite motif sequences, and *nmotif* values. See the file format above in the required files section. The *sampleDir* argument indicates the folder where the files are stored.

Plotting the distribution of the read sequence sizes

Now, we can use the *reads.plot* function to plot the distribution of read sequence size (x-axis bp range) by filtered read coverage (y-axis).

```
reads.plots(sampleDir = "./samplesF", locusInfo = "primers", x.range = c(90, 350))
```

The most important arguments are shown; check the function manual for other options. The plots are created inside the directory *Plots* written by the function in the working directory. Examples of these plots for a heterozygous and homozygous sample are shown in Fig. 1 and 2, respectively. The function will produce a single PDF file per locus containing all samples scored.

Creating tables with expected relative frequency of reads for all combinations of possible genotypes

These expected frequency tables will be matched to the observed relative frequencies (the sequencing data for each sample) to score alleles when using function *alleleFinder*. A few calibration homozygous samples are

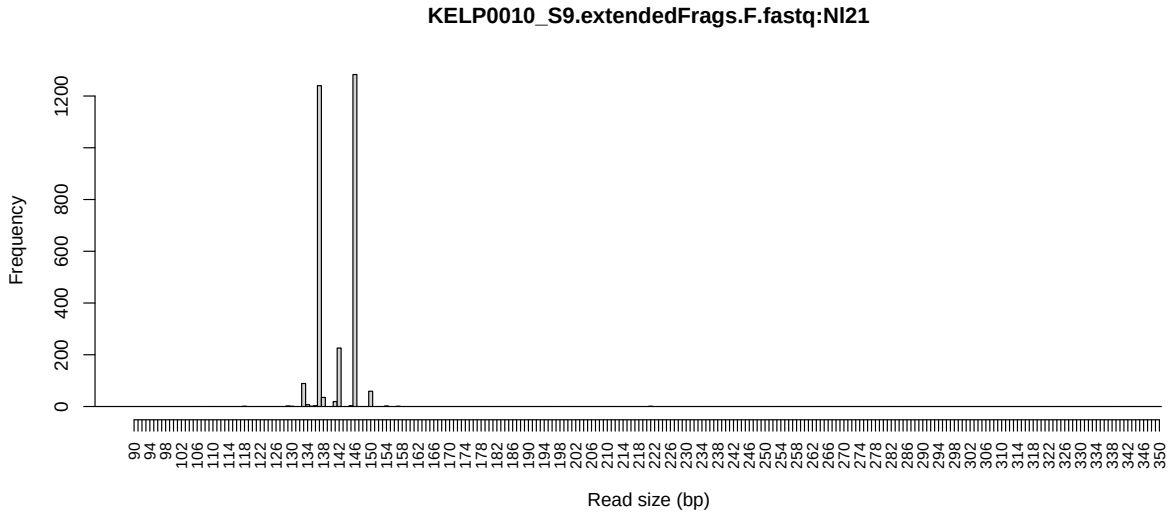

Figure 1: Example of the read distribution plot for one heterozygous sample in locus Nl21.

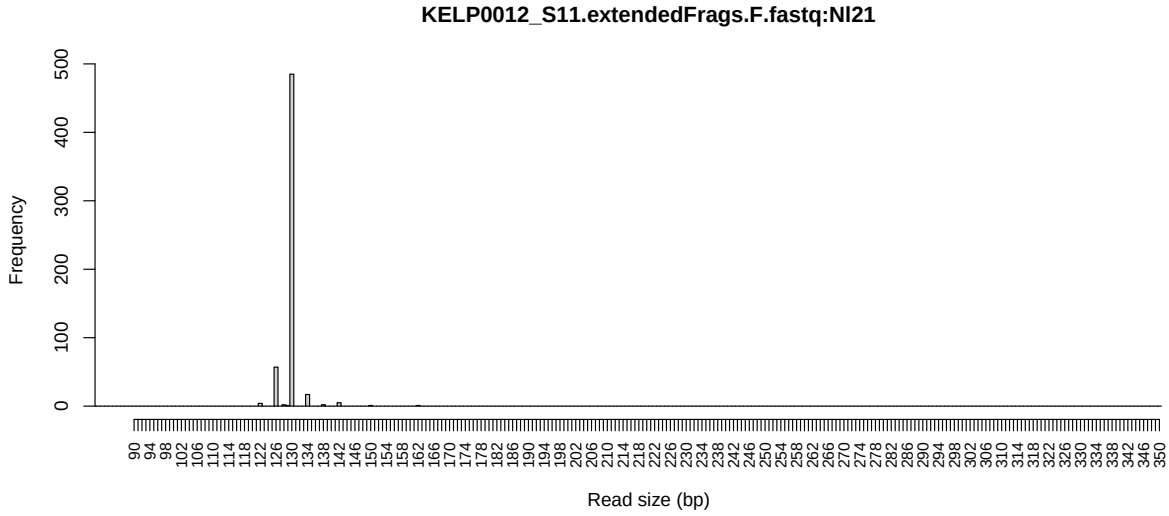

Figure 2: Example of the read distribution plot for one homozygous sample in locus Nl21.

necessary to generate these tables. Select one to six homozygous samples by analyzing the plots created in the previous step. Figure 2 shows an example of a homozygous pattern for locus Nl21. Write the names of these samples in the different rows of the calibration file (see *Required Files* above). (tip: copy the name from the plot directly).The function will use the information of intervals between amplified reads for a single allele to generate the expectation for all possible combinations of alleles in a diploid genotype within the allele range supplied in the calibration file.

Upon observing the plots created in the previous step, 1) select an allele range to bind the left and right limit of alleles that the function will consider. Scrutinize all samples so that the selected range does not exclude possible alleles. However, use a range that is not much wider than necessary because this will slow down the code. Write the range for each locus in the second row of the calibration file (see required files above, e.g., 100-250); and 2) inspect the patterns of possible alleles and determine a level of coverage (y-axis) that captures reads you believe are alleles. Write down this coverage in each locus's last row of the calibration file. Reads scored as alleles need coverage larger than this value; otherwise, they are written as missing data NA.

Finally, the function has an argument to adjust the reduction in large allele coverage for heterozygotes (allele dropout); *drop.out.b* is the proportion of coverage loss per base pair distance between the two alleles, it defaults to 0.0035. For example, a difference of 100 bp between the two alleles would mean that the second allele expected coverage would be 35% of the small allele. Adjust this parameter care, if increased too much you can get negative values in the expected coveargе tables produced, which will stop the function with a warning message.

We can now run the *GenotypeModel* function.

```
GenotypeModel(sampleDir = "./samplesF", locusInfo = "primers",
    calibrationFiles = "CalibrationSamples.txt")
```

Scoring alleles

With the model tables produced, we can now score fragment size-based alleles. The function will also export a genotype table to the local environment and save two files inside the *GenotypesOut* directory created in the working directory. Both files have the diploid genotypes but one or two columns per locus (one: locus table or two: alleles table).

```
genos.DB <- alleleFinder(sampleDir = "./samplesF", locusInfo = "primers",
    calibrationFiles = "CalibrationSamples.txt")
```

Plotting the alleles scored

After producing the genotype tables we can now use them to plot the alleles scored (Figure 3). This will help visualize how well the scoring algorithm performed. We will use again the *reads.plots* function but this time we supply the path to the genotype table created by *alleleFinder* (the one with two columns per locus) to argument *G.DB*. The code below will create a folder called *Plots.Scored* in the working environment, where the new "scored" plots are saved.

```
reads.plots(sampleDir = "./samplesF", locusInfo = "primers", x.range = c(90,
    315), G.DB = "./GenotypesOut/Genotypes allele table.txt")
```

At this stage, recheck the genotype tables and plots to ensure that alleles were not missed. If adjustments are needed, edit the respective loci's coverage value in the calibration file and rerun alleleFinder.

Plotting read distributions for all loci and samples

We can now visualize the alleles scored for each loci using cumulative distribution plots (similar to those produced by MsatAllele [Alberto 2009]).

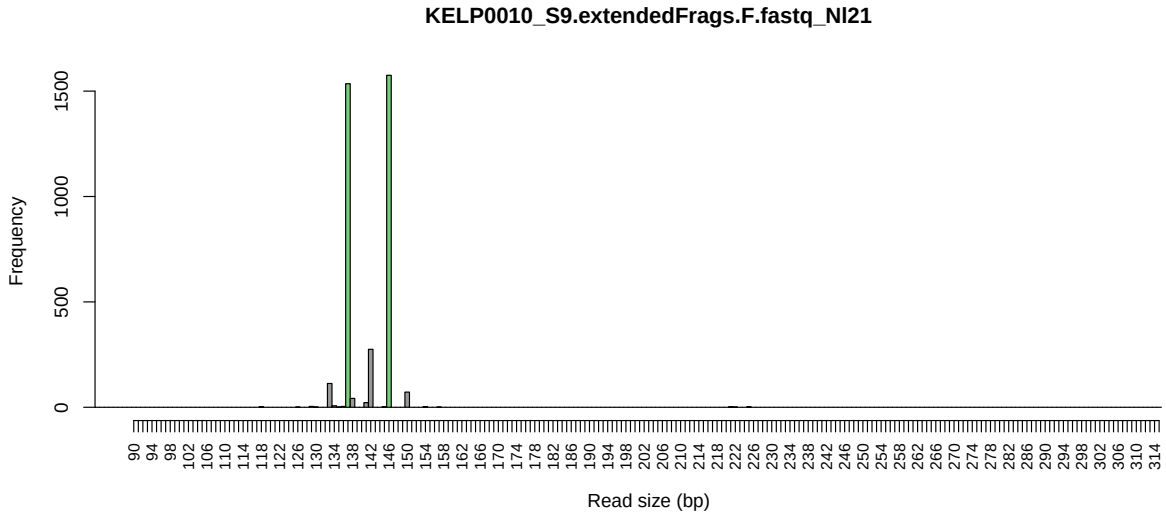

Figure 3: Example of the read distribution plot for a scored heterozygous sample in locus Nl21. Scored alleles are shown in green.

```
# reading the table with the scored alleles that was produced by alleleFinder
genos.DB <- read.delim("./GenotypesOut/Genotypes allele table.txt")

AlleleCum(GenotypeA.DF = genos.DB, locus = "Nl25", locusInfo = "primers", ymin = NULL,
    ymax = NULL)
```

Figure 4 shows the plot produced by the code above. The example is for locus Nl25 for all alleles contained in 10 samples. When plotting much larger data sets, the *ymin* and *ymax* arguments can be used to zoom in.

Further interaction with the plot is possible using the function *getpoints*, which will require the user to click the mouse pointer twice on the plot to retrieve the names of samples carrying a particular range of alleles. The first click is below the desired range of alleles, and the second is above it. Below, we show the output of running *getpoints* and clicking below and above the 168 bp allele (Table 3).

```
getpoints(GenotypeA.DF = genos.DB, locus = "Nl25")
```

Table 3: Example of samples carrying allele 168 for locus Nl25, as retrieved by interacting with the cumulative allele distribution plot for this locus (Fig.3) using function getpoints

Sample	Nl25.a1	Nl25.a2
KELP0017_S16.extendedFrags.F.fastq	168	168
KELP0022_S21.extendedFrags.F.fastq	168	188
KELP0023_S22.extendedFrags.F.fastq	168	176
KELP0027_S26.extendedFrags.F.fastq	168	168
KELP0029_S28.extendedFrags.F.fastq	168	168

Scoring sequence based alleles

So far, the pipeline has identified only alleles based on read size. We will now identify alleles based on sequence composition. When using amplicon sequencing, these are alleles of the same size that differ in their sequence and are hereafter called sequence variants.

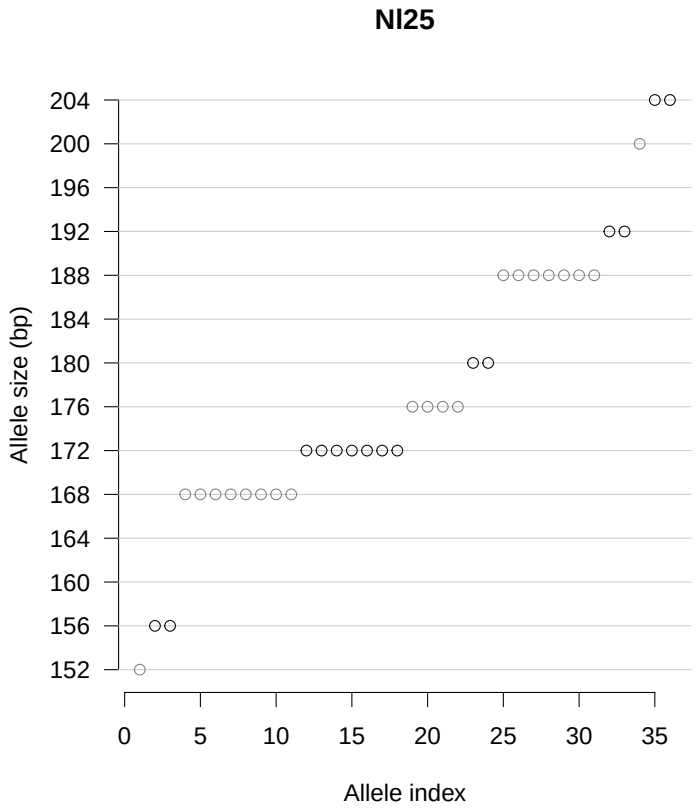

Figure 4: Cumulative allele distribution for locus Nl25, a tetra-nucleotide.

```
score.seq.variants(locusInfo="primers",
            RefGenotypes="./GenotypesOut/Genotypes allele table.txt",
            sampleDir="./samplesF", HeteroThreshold=0.4,alleleDB=NULL,
            calibrationFiles="CalibrationSamples.txt")
```

There are three new function arguments to mention here. *RefGenotypes* takes the "Genotype allele table.txt" written into the "GenotypeOut" directory by function *alleleFinder*. Note that the function will only look for sequence-variant alleles "within" the size-based alleles written to "Genotype allele table.txt".

When sequence variation exists within reads of the same size for the same loci, two variants will dominate the read frequencies, with many more in low frequency due to sequencing errors. The *HeteroThreshold* argument refers to the proportion of reads in the second most frequent variant compared to the first. The value sets the minimum allowed proportion required to score the second allele as a sequence-based variant allele, i.e., not a sequencing error. It defaults to 0.4.

alleleDB is the file name for a tab-delimited file containing the allele database with information on allele size, sequence variation, sequence-variant allele code, and the nucleotide sequence (Table 4). The default is NULL, in which case the function will produce the database from the data supplied. When a database is supplied, the function will update it with any new sequence variants that might be identified. Supplying a database allows users to combine different runs or studies to get allele codes that match.

The database is organized with different sequence variants by row and the following required columns:

Locus: The locus name for the sequence-based allele. Needs to match the loci names supplied in argument locusInfo;

SIZE: The size in bp for the sequence-based allele;

VARIANT: A low case letter code to distinguish the different sequence-based alleles that share the same sequence size;

SEQ: The nucleotide sequence for each sequence-based allele;

AlleleCode: A numeric code used to write alleles in a format most population genetics software requires.

The function writes three output files in a folder created in the R working directory named *AlleleBySeq*.

The files are "Allele DataBase.txt" (table 4), containing the reference database of allele sequence variants found;

"Genotypes.by.Sequence.txt" (Table 5) contains a population genetics file with two columns per locus and alleles represented by the concatenation of allele size and variant letter code (e.g., 150a, 150b);

"Genotype Codes.txt" contains a population genetics file with new numeric allele codes, one column per locus, required for most population genetics software.

The allele database file contains the correspondence between the different allele code formats. A "warningLog" file is also written directly in the working directory, reporting cases where more than two alleles were found. When this happens, the diploid genotype is ambiguous; thus, NAs are written to the output files.

Table 4: The database required by argument alleleDB in function score.seq.variants should have the format in this table. Note that sequence information in SEQ column is trimmed to fit the table. When a database is not supplied to the function, one is created as an output.

Locus	SIZE	VARIANT	AlleleCode	SEQ
Nl21	114	a	100	...CGTACATACATACATACATACATACA...
Nl21	122	a	101	...CGTACATACATACATACATACATACA...
Nl21	122	b	102	...CGTACATATATACATACATACATACA...

Locus	SIZE	VARIANT	AlleleCode	SEQ
Nl21	126	a	103	...CGTACATACATACATACATACATACA...
Nl21	130	a	104	...CGTACATACATACATACATACATACA...
Nl21	134	a	105	...CGTACATACATACATACATACATACA...
Nl21	134	b	106	...CGTACATACATACATACATACATACA...
Nl21	137	a	107	...CGTACATACATACATACATACATACA...
Nl21	137	b	108	...ATACGTACGTACATACATACATACAT...
Nl21	138	a	109	...CGTACATACATACATACATACATACA...
Nl21	138	b	110	...CGTACATACATACATACATACATACA...
Nl21	141	a	111	...ATACGTACGTACATACATACATACAT...
Nl21	142	a	112	...CGTACATACATACATACATACATACA...
Nl21	142	b	113	...CGTACATACATACATACATACATACA...
Nl21	142	c	114	...CGTACATACATACATACATACATACA...

Table 5: View of the first three loci in the Genotypes.by.sequence.txt output file. Sequence-variant alleles are coded by concatenating allele size with a letter code. For example, for the first locus Nl21 (columns 2 and 3), we can see three sequence-variant alleles for size 150 (a, b, and c) in locus Nl21.

Sample	Nl21.a1	Nl21.a2	Nl24.a1	Nl24.a2	Nl25.a1	Nl25.a2
KELP0010_S9.extendedFrags.F.fastq	137a	137a	166a	166a	NA	NA
KELP0011_S10.extendedFrags.F.fastq	130a	194a	149a	166a	152a	152a
KELP0012_S11.extendedFrags.F.fastq	130a	130a	158a	166a	188a	188a
KELP0013_S12.extendedFrags.F.fastq	154a	154a	150a	170a	188a	188a
KELP0014_S13.extendedFrags.F.fastq	182a	182a	162a	173a	NA	NA
KELP0015_S14.extendedFrags.F.fastq	154a	154a	149a	166a	NA	NA
KELP0016_S15.extendedFrags.F.fastq	142a	150a	157a	169a	156a	156a
KELP0017_S16.extendedFrags.F.fastq	150b	150b	157a	169a	168a	168a
KELP0018_S17.extendedFrags.F.fastq	138a	138a	150a	177a	176a	180a
KELP0019_S18.extendedFrags.F.fastq	150b	158a	161a	161a	172a	172a
KELP0020_S19.extendedFrags.F.fastq	130a	130a	149b	169a	176b	188a
KELP0021_S20.extendedFrags.F.fastq	138a	138a	150a	177a	176a	180a
KELP0022_S21.extendedFrags.F.fastq	142a	142a	170a	173a	168a	188a
KELP0023_S22.extendedFrags.F.fastq	146a	146a	153a	170a	168a	176a
KELP0024_S23.extendedFrags.F.fastq	142b	142b	153a	169a	172a	192a
KELP0025_S24.extendedFrags.F.fastq	142a	182a	157a	169a	192b	200a
KELP0026_S25.extendedFrags.F.fastq	146a	158a	166a	166a	172a	172a
KELP0027_S26.extendedFrags.F.fastq	142a	150c	170a	174a	168a	168a
KELP0028_S27.extendedFrags.F.fastq	NA	NA	NA	NA	NA	NA
KELP0029_S28.extendedFrags.F.fastq	134a	182a	165a	169a	168a	168a
KELP0030_S29.extendedFrags.F.fastq	130a	150c	169a	170a	172a	172a

Working with eDNA or forensic samples with more than two alleles per sample

Amplicomsat can score multiple alleles (>2) in each locus. Examples of such samples come from forensic analysis, where DNA from multiple individuals might be present, or eDNA samples containing alleles from several individuals. Only alleles present in a reference population can be detected.

Scoring multiple size-based alleles per sample locus combination

Using a similar approach as above, we first score size-based alleles, allowing for multiple alleles to be scored for the same locus in one sample, and then use the resulting data to search for additional sequence-based variants.

Again, we start by filtering the reads for sequencing quality using the function *read.quality.filter* (see code above). In this case, the filtered reads were then copied to a directory we created in the working directory of the R session named "samplesF.MA."

Next, we use *read.size.freqs* to find and filter reads in these samples. The code for these two steps is not shown below, but it is similar to the examples above while changing the names parsed to arguments if necessary.

Next we use the function *M.allele.Finder* to score all alleles from each locus in our samples.

The argument *RefGenotypes* takes the two-column per locus genotype table of the reference population, as produced by the *alleleFinder* function (see above). Only alleles present in a reference population can be scored as multiple alleles in any sample; thus, the genotype file for such a reference population should be used here.

The argument *ReadSizesRepeatNumbers* takes the path to the folder created by function *read.size.freqs* above.

The function returns a list with the alleles found for each locus and individual. The function also writes a new directory called *M.plots* with pdf files for each sample, containing plots of read (allele) distribution, with separate plots per page showing the different loci and blue colored bars indicating the alleles scored (Figure 5).

```
AllelesSample<-M.allele.Finder(sampleDir="./samplesF.MA",
            locusInfo="primers",
            calibrationFiles="CalibrationSamples.txt",
      ReadSizesRepeatNumbers="./ReadSizesRepeatNumbers/",
            RefGenotypes="./GenotypesOut/Genotypes allele table.txt")
```

```
## Inspecting the list with the multiple alleles scored for
## each loci in the first sample
AllelesSample["S.1.simulated.Forensics.fastq"]
```

```
## $S.1.simulated.Forensics.fastq
## $S.1.simulated.Forensics.fastq$Nl21
## [1] 126 130 134 137 142 146
##
## $S.1.simulated.Forensics.fastq$Nl24
## [1] 158 165 166 169 177
##
## $S.1.simulated.Forensics.fastq$Nl25
## [1] 180 184 188
##
## $S.1.simulated.Forensics.fastq$Nl26
## [1] 175 187 191 195 199 203
##
## $S.1.simulated.Forensics.fastq$Nl27
## [1] 150 154 158 162 166
##
## $S.1.simulated.Forensics.fastq$Nl30
## [1] 142 148 162 164 166 198
##
## $S.1.simulated.Forensics.fastq$Nl40
## [1] 172
```

```
##
## $S.1.simulated.Forensics.fastq$N147
## [1] 155 163 167
##
## $S.1.simulated.Forensics.fastq$N150
## [1] 170 178 182 186
##
## $S.1.simulated.Forensics.fastq$N159
## [1] 120 128 136
##
## $S.1.simulated.Forensics.fastq$Ner2
## [1] 260
```

The code below can be used to count the number of alleles per locus found for a given sample. This is the observed allele count (OAC) required as an imput to package GenotypeQuant. The OAC can be used to estimate the number of contributor (NOC) genotypes in the sample trough Maximum likelihood estimation (Liggan et al. submitted, see https://github.com/UWMAlberto-Lab/GenotypeQuant)

```
## Inspecting the list with the multiple alleles scored for
## each loci in the first sample
sapply(AllelesSample["S.1.simulated.Forensics.fastq"][[1]], length)
```

```
## N121 N124 N125 N126 N127 N130 N140 N147 N150 N159 Ner2
##    6    5    3    6    5    6    1    3    4    3    1
```

Or for all samples at once.

```
FileNames <- list.files("./samplesF.MA")
locusNames <- read.delim("primers")[, 1]
n.locus <- length(locusNames)
n.files <- length(FileNames)
AlleleCounts <- data.frame(matrix(nrow = n.files, ncol = n.locus +
    1))
AlleleCounts[, 1] <- FileNames
names(AlleleCounts) <- c("Samples", locusNames)

for (f in 1:n.files) {
    AlleleCounts[f, 2:ncol(AlleleCounts)] <- as.numeric(sapply(AllelesSample[FileNames[f]][[1]],
        length))
}

AlleleCounts
```

```
##                          Samples N121 N124 N125 N126 N127 N130 N140 N147 N150
## 1   S.1.simulated.Forensics.fastq    6    5    3    6    5    6    1    3    4
## 2  S.10.simulated.Forensics.fastq    7    9    3    5    4    3    2    3    6
## 3   S.2.simulated.Forensics.fastq    5    7    3    5    7    5    2    3    5
## 4   S.3.simulated.Forensics.fastq    8    9    3    3    6    4    2    5    6
## 5   S.4.simulated.Forensics.fastq    3    8    5    3    5    5    2    3    5
## 6   S.5.simulated.Forensics.fastq    7    8    3    3    3    3    1    5    4
## 7   S.6.simulated.Forensics.fastq    6   10    3    4    4    3    3    5    6
## 8   S.7.simulated.Forensics.fastq    5    8    4    3    4    4    2    5    2
## 9   S.8.simulated.Forensics.fastq    6    7    3    2    5    2    0    4    4
## 10  S.9.simulated.Forensics.fastq    7    8    2    2    4    3    1    3    5
##    N159 Ner2
## 1     3    1
## 2     3    1
```

```
## 3      2     1
## 4      6     2
## 5      3     1
## 6      5     1
## 7      7     0
## 8      5     1
## 9      3     1
## 10     4     1
```

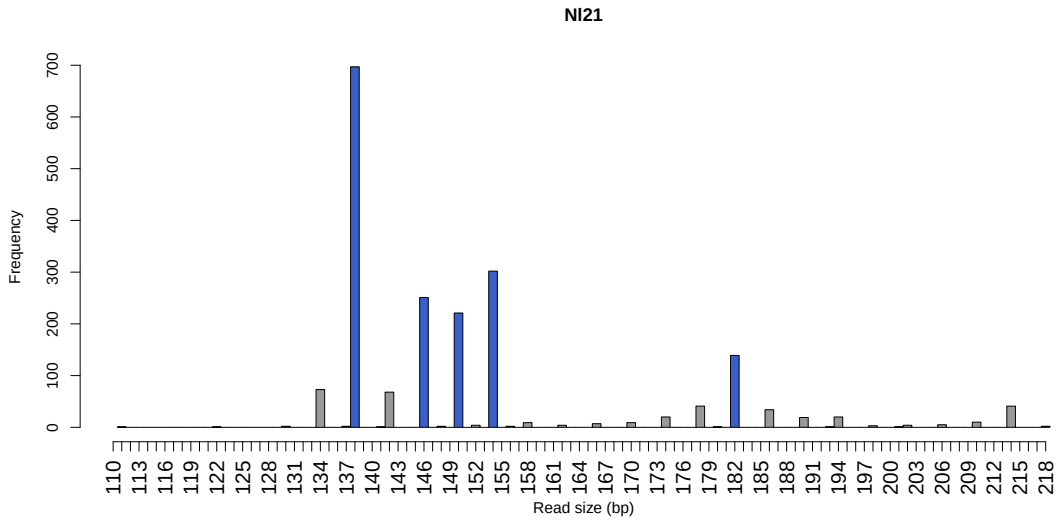

Figure 5: Read distribution with multiple alleles called (blue bars). This example shows the plot for locus Nl21 in one sample.

Scoring multiple sequence-based alleles per sample locus combination

Finally, we use function *M.allele.seq* to score sequence variant alleles that may occur within the multiple size-based alleles identified in the previous step. The user must supply the list of multiple alleles created using *M.allele.Finder* (previous step) to argument *AlleleList* below. Also required is the database of sequence variant alleles, like the one created by function *score.seq.variants* above.

The function does not create a database by itself because this type of forensics or DNA analysis often requires allele frequencies from a reference population that would be first analyzed with *score.seq.variants* creating a database to use here. However, the database might be updated while running *M.allele.seq*. This will happen if the differences in the read coverage filter are different between function uses (inspect values and plots), resulting in new sequence variants that pass the coverage filter. If the database of sequence variant alleles is updated, the new database is written to the working directory and named "Allele DataBase UPDATED.txt." The new entries to the database are also written in a file named "DataBase new entries.txt." This file might be useful in investigating the origin of the new alleles.

```
List.seq.basedAlleles <- M.allele.seq.variant(sampleDir = "./samplesF.MA",
    AlleleList = AllelesSample, locusInfo = "primers",
    alleleDB = "./AlleleBySeq/Allele DataBase.txt",
    HeteroThreshold = 0.4, calibrationFiles = "CalibrationSamples.txt")
```

The function returns a list with the multiple sequence variant alleles found for each sample and locus. The list has a similar format to the one produced by *M.allele.Finder*; we adapt the previous code below to summarize the results.

```
## Inspecting the list with the multiple alleles scored for
## each loci in the first sample
List.seq.basedAlleles["S.1.simulated.Forensics.fastq"]
```

```
## $S.1.simulated.Forensics.fastq
## $S.1.simulated.Forensics.fastq$Nl21
## [1] "126a" "130a" "134a" "137b" "142b" "146a"
##
## $S.1.simulated.Forensics.fastq$Nl24
## [1] "158b" "165a" "166a" "169a" "177a"
##
## $S.1.simulated.Forensics.fastq$Nl25
## [1] "180e" "184a" "188f"
##
## $S.1.simulated.Forensics.fastq$Nl26
## [1] "175a" "187a" "191a" "195d" "199a" "203b"
##
## $S.1.simulated.Forensics.fastq$Nl27
## [1] "150a" "154b" "154a" "158a" "162b" "166b"
##
## $S.1.simulated.Forensics.fastq$Nl30
## [1] "142a" "148a" "162a" "164a" "166a" "198a"
##
## $S.1.simulated.Forensics.fastq$Nl40
## [1] "172a"
##
## $S.1.simulated.Forensics.fastq$Nl47
## [1] "155e" "163a" "167a"
##
## $S.1.simulated.Forensics.fastq$Nl50
## [1] "170a" "178a" "182a" "186a"
##
## $S.1.simulated.Forensics.fastq$Nl59
## [1] "120a" "128a" "136a"
##
## $S.1.simulated.Forensics.fastq$Ner2
## [1] "260a"
```

The code below can be used to count the number of sequence variant alleles found for a given sample (OAC).

```
## Inspecting the list with the multiple alleles scored for
## each loci in the first sample
sapply(List.seq.basedAlleles["S.1.simulated.Forensics.fastq"][[1]],
    length)
```

```
## Nl21 Nl24 Nl25 Nl26 Nl27 Nl30 Nl40 Nl47 Nl50 Nl59 Ner2
##    6    5    3    6    6    6    1    3    4    3    1
```

Or for all samples at once.

```
Seq.Based.AlleleCounts <- data.frame(matrix(nrow = n.files, ncol = n.locus +
    1))
Seq.Based.AlleleCounts[, 1] <- FileNames
names(Seq.Based.AlleleCounts) <- c("Samples", locusNames)

for (f in 1:n.files) {
```

```
    AlleleCounts.f <- as.numeric(sapply(List.seq.basedAlleles[FileNames[f]][[1]],
        length))
    Seq.Based.AlleleCounts[f, 2:ncol(Seq.Based.AlleleCounts)] <- AlleleCounts.f
}

Seq.Based.AlleleCounts
```

```
##                          Samples N121 N124 N125 N126 N127 N130 N140 N147 N150
## 1   S.1.simulated.Forensics.fastq    6    5    3    6    6    6    1    3    4
## 2  S.10.simulated.Forensics.fastq    8    9    3    5    4    3    2    3    7
## 3   S.2.simulated.Forensics.fastq    7    7    3    5    8    5    2    3    6
## 4   S.3.simulated.Forensics.fastq    9    9    4    3    7    4    2    5    6
## 5   S.4.simulated.Forensics.fastq    3    8    5    3    5    5    2    3    5
## 6   S.5.simulated.Forensics.fastq    7    8    3    3    3    3    1    5    4
## 7   S.6.simulated.Forensics.fastq    6   10    3    4    7    3    3    5    8
## 8   S.7.simulated.Forensics.fastq    5    8    4    3    6    4    2    5    2
## 9   S.8.simulated.Forensics.fastq    6    7    3    2    7    2    0    4    4
## 10  S.9.simulated.Forensics.fastq    7    8    2    2    6    3    1    4    5
##    N159 Ner2
## 1     3    1
## 2     3    1
## 3     2    1
## 4     6    3
## 5     3    1
## 6     5    1
## 7     7    0
## 8     5    1
## 9     3    1
## 10    4    1
```

A quick subtraction of sequence-based allele counts by size-based alleles gives the difference in number of alleles revealed by each method per sample and loci combination.

```
Seq.Based.AlleleCounts[, 2:12] - AlleleCounts[, 2:12]
```

```
##    N121 N124 N125 N126 N127 N130 N140 N147 N150 N159 Ner2
## 1     0    0    0    0    1    0    0    0    0    0    0
## 2     1    0    0    0    0    0    0    0    1    0    0
## 3     2    0    0    0    1    0    0    0    1    0    0
## 4     1    0    1    0    1    0    0    0    0    0    1
## 5     0    0    0    0    0    0    0    0    0    0    0
## 6     0    0    0    0    0    0    0    0    0    0    0
## 7     0    0    0    0    3    0    0    0    2    0    0
## 8     0    0    0    0    2    0    0    0    0    0    0
## 9     0    0    0    0    2    0    0    0    0    0    0
## 10    0    0    0    0    2    0    0    1    0    0    0
```

Additional functions

TODO: The pipeline relies on filtering reads by primer and microsatellite sequencing, and most functions look for exact matches in the primer sequences. Fastq files might show primer sequences that differ from those in the user-supplied loci information file for various reasons, leading to a failure to extract read information for affected loci. If there is doubt that the loci information file might not precisely match the sequences in the fastq files, the function *Edit.Primer.File* can search primer sequences, allowing for mismatches of up to a

user-determined number of base pairs. If any mismatches are found, the function writes an edited primer info file for use in the pipeline above. Because this function searches all reads, allowing for mismatches, it is the slowest function in the package.