alleHap: Allele Imputation and Haplotype Reconstruction from Pedigree Databases

Nathan Medina-Rodriguez

Angelo Santana

Package Version: 0.9.9

August 30, 2024

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Departamento de Matemáticas

Abstract

`alleHap` contains tools to simulate alphanumeric alleles, impute genetic missing data and reconstruct non-recombinant haplotypes from pedigree databases in a deterministic way. Allele simulations can be carried out taking into account many factors (such as number of families, markers, alleles per marker, probability and proportion of missing genotypes, recombination rate, etc). Genotype imputation can be performed with simulated datasets or real databases (previously loaded in .ped format). Haplotype reconstruction can be carried out even with missing data, since the program firstly tries to impute each family genotype (without a reference panel), to later reconstruct the corresponding haplotypes for each family member when possible. All these procedures are based on considering that each individual (due to meiosis) should have two alleles per marker, one inherited from each parent; the application of simple rules comparing genotypes of parents with offsprings and the offsprings against each other allow in many cases haplotypes to be deterministically reconstructed and genotypes unequivocally imputed. `alleHap` is very robust against inconsistencies within the genotypic data, warning when they occur. Furthermore, the package is handy, intuitive and have good performance properties in execution time and memory usage, even when handling large amounts of genetic data.

This vignette intends to explain in detail how `alleHap` package works for the desired applications, and it includes illustrated explanations and easily reproducible examples.

Contents

1 Introduction

Genotype imputation and haplotype reconstruction have achieved an important role in Genome-Wide Association Studies (GWAS) during recent years. Estimation methods are frequently used to infer either missing genotypes as well as haplotypes from databases containing related and/or unrelated subjects. The majority of these analyses have been developed using several statistical methods [BB11] which can impute probabilistically genotypes as well as perform haplotype phasing (also known as haplotype estimation) of the corresponding genomic regions, by using reference panels.

Current algorithms do not carry out genotype imputation or haplotype reconstruction using solely deterministic techniques on pedigree databases. These methods are usually focused on population data and in case of pedigree data, families normally are comprised by trios [BB09], being uncommon those studies consisting of more than two offspring for each line of descent. Studies composed of large pedigrees are also infrequent. When there is a lack of family genetic data, computational inference by probabilistic models using reference panels must be necessarily used and may cause many incorrect inferences.

On the other hand, certain regions are very stable against recombination but at the same time, they may be highly polymorphic. These stable loci may be occupied by tens to hundreds of different alleles. For this reason, in some well-studied regions (such as Human Leukocyte Antigen (HLA) loci [MCH+13] in the extended human Major Histocompatibility Complex (MHC) [dBMS+06]) an alphanumeric nomenclature is needed to facilitate later analysis. In these conditions, usual typing techniques may not take full advantage of the alleles composition.

Our package `alleHap` can combine genetic information of parents and children on stable and highly polymorphic regions with the double objective of imputing missing alleles and reconstructing family haplotypes. The procedure is deterministic and uses only the information contained in the family. When genotypes of several children are available, imputation and reconstruction are possible even when parents have fully or partially missing genotypes. Obviously, imputation and reconstruction rates are not always of 100%, but the output of the program could then be used by other existing probabilistic imputation and reconstruction algorithms that use reference panels, thus leading to improved inferences from these algorithms.

2 Theoretical Description

`alleHap` algorithms are based on a preliminary analysis of all possible combinations that may exist in the genotype of a family, considering that each child should unequivocally have inherited two alleles, one from each parent. The analysis begins with the differentiation of seven cases, as described in [BWSD+07]. These cases can be grouped considering the number of unique (or different) alleles per family. So, using the notation N_{par}: *Number of unique alleles in parents* and N_p: *Number of unique alleles in parent p*, the expression: (N_{par}, N_1, N_2) allow to identify all the non-recombinant configurations in families with one line of descent (i.e. parent-offspring pedigree). Table 1 shows the different configurations in biallelic mode:

Configurations	1	2	3	4	5	6	7
N_{par}	1	2	2	3	2	3	4
(N_1,N_2)	(1,1)	(1,1)	(1,2)	(1,2)	(2,2)	(2,2)	(2,2)
Parents	a/a a/a	a/a b/b	a/a a/b	a/a b/c	a/b a/b	a/b a/c	a/b c/d
Possible Offsring	a/a	a/b	a/a a/b	a/b a/c	a/a b/b a/b	a/a a/b a/c b/c	a/c a/d b/c b/d

Table 1: Biallelic configurations in a parent-offspring pedigree

As a previous step for the proper operation of `alleHap`, an identification of the homozygous genotypes for each family is necessary. An example of some unphased biallelic genotypes (left) and their corresponding Homozygosity (HMZ) matrix is shown in the next table; for any marker in a subject, the value of HMZ is 1 if it is homozygous and 0 if heterozygous:

	Unphased Data							Homozygosity Info.						
Marker	1	2	3	4	5	6	7	1	2	3	4	5	6	7
Parents	a/a	a/a	a/a	a/a	a/b	a/b	a/b	1	1	1	1	0	0	0
	a/a	b/b	a/b	b/c	a/b	a/c	c/d	1	1	0	0	0	0	0
Offsring	a/a	a/b	a/a	a/b	a/a	a/a	a/c	1	0	1	1	1	0	0
			a/b	a/c	b/b	a/b	a/d			0	1	1	0	0
					a/b	a/c	b/c					0	0	0
						b/c	b/d						0	0

Table 2: Biallelic unphased genotypes and HMZ matrix

To determine the haplotypes, `alleHap` creates an IDentified/Sorted (IDS) matrix from the genotypes of each family. In children, a genotype a/b of a marker is *phased* if it can be unequivocally determined that the first allele comes from the father and the second from the mother. In this way, the sequence of first (second) alleles of phased markers is the haplotype inherited from the father (or mother). So, when a marker in a child can be phased this way its IDS value is 1; in other case its value is 0. In parents, genotypes can be phased if there exists at least one child with all its genotypes phased (reference child). Then, for every marker, the alleles of the genotype in a parent are sorted in such a way that first allele coincides with the corresponding allele inherited from that parent in the reference child. When this sorting is achieved, the IDS value in the parent is 1; in other case its value is 0.

An example of the values of this matrix (right) and the corresponding phased genotypes (left) is the following; note that when the genotype a/b is phased we denote it by $a|b$; the first child in each group is considered to be the reference child for phasing the parents:

	Phased Data							IDS Matrix						
Marker	1	2	3	4	5	6	7	1	2	3	4	5	6	7
Parents	a|a	a|a	a|a	a|a	a|b	a|b	a|b	1	1	1	1	1	1	1
	a|a	b|b	a|b	b|c	a|b	a|c	c|d	1	1	1	1	1	1	1
Offspring	a|a	a|b	a|a	a|b	a|a	a|a	a|c	1	1	1	1	1	1	1
			a|b	a|c	b|b	a|b	a|d			1	1	1	1	1
					a/b	a|c	b|c					0	1	1
						b|c	b|d						1	1

Table 3: Phased genotypes and IDS matrix

Sometimes, missing values may occur. These can be located either in parents or children. An example of this is depicted as follows:

	Phased Data							IDS Matrix						
Marker	1	2	3	4	5	6	7	1	2	3	4	5	6	7
Parents	a|a	a|a	a|a	a|a	NA	NA	NA	1	1	1	1	0	0	0
	NA	NA	NA	b|c	a|b	a|b	a|b	0	0	0	1	1	1	1
Offspring	a|a	a|b	a|a	NA	a|a	a|a	c|a	1	1	1	0	1	1	1
			a|b	a|c	b|b	a/b	d|a			1	1	1	0	1
					a/b	c|a	c|b					0	1	1
						c|b	d|b						1	1

Table 4: Phased genotypes and IDS matrix containing missing data

3 Input Format

`alleHap` uses PED files as input, although it can detect and adapt similar formats (with the same structure) to later load the data.

3.1 PED files

A PED file is a white-space (space or tab) delimited file where the first six columns are mandatory and the rest of columns are the genotype: *Family ID* (identifier of each family), *Individual ID* (identifier of each member of the family), *Paternal ID* (identifier of the paternal ancestor), *Maternal ID* (identifier of the maternal ancestor), *Sex* (gender of each individual: 1=male, 2=female, other=unknown), *Phenotype* (quantitative trait or affection status of each individual: -9=missing, 1=unaffected, 2=affected) and the ***genotype*** of each individual (in biallelic or coded format).

The IDs are alphanumeric: the combination of family and individual ID should uniquely identify a person. **PED files must have 1 and only 1 phenotype in the sixth column**. The phenotype can be either a quantitative trait or an affection status column. Genotypes (column 7 onwards) should also be white-space delimited; they can be any character (e.g. 1,2,3,4 or A,C,G,T or anything else) except 0 which is, by default, the missing genotype character. All markers should be biallelic and must have two alleles specified [PNTB+07]. For example, a family composed of three individuals typed for N SNPs is represented in Table 5:

Fam ID[a]	*Ind ID*	*Pat ID*	*Mat ID*	*Sex*	*Pheno*	*Mkr_1*		*Mkr_2*		*Mkr_3*			*Mkr_N*	
FAM001	1	0	0	1	0	A	A	G	G	A	C	...	C	G
FAM001	2	0	0	2	1	A	A	A	G	C	C	...	A	G
FAM001	3	0	0	1	0	A	A	G	A	A	C	...	C	A

Table 5: Example of a Family in .ped file format

[a]No header row should be given. It is shown here for clarity.

3.2 NA values

The missing values or Not Available (NA) values may be placed either in the first six columns and also in genotype columns. In the genotype colums, when some values are missing either both alleles should be 0, -9, NA. An example of this would be:

```
   famID indID patID matID sex phenot Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2
1 FAM001     1     0     0   1      1     1     2    NA    NA     1     2
2 FAM001     2     0     0   2      2     3     4     1     2     3     4
3 FAM001     3     1     2   1      2     1     3     1     2     1     3
4 FAM001     4     1     2   2      1    NA    NA     1     1     2     4
5 FAM001     5     1     2   1      2     1     4     1     1     2     4
```

4 Data Simulation

`alleHap` can simulate biallelic pedigree databases. Simulation can be performed taking into account different factors such as number of families to generate, number of markers, number of different alleles per marker, type of alleles (numeric or character; alleles of type character are assumed to take the values 'A','C','G' or 'T', and markers with this type of alleles can have only two different alleles; when alleles are of numeric type, a marker can have n different alleles, indexed from 1 to n; in a simulation all markers are of the same type, defined by the argument *chrAlleles*; when TRUE all markers are of type character, when FALSE all are of numeric type), number of different haplotypes in the population,

probability of missing genotypes in parent/offspring, proportion of missing genotypes per individual, probability of being affected by disease and recombination rate.

4.1 alleSimulator Function

`alleSimulator` function generates the clinical and genetic information of a group of families according to the previously defined parameters. This function performs several steps in order to simulate the data:

I. **Loading of Internal Functions**: In this step all the necessary functions to simulate the data are loaded. These functions are *labelMrk* (which creates the 'A','C','G','T' character labels), *simHapSelection* (which selects n different haplotypes between the total number of possible haplotypes; these selected haplotypes constitutes the population of haplotypes 'popHaplos', from which the different families will be derived), *simOffspring* (which generates n offspring by selecting randomly one haplotype from each parent), *simOneFamily* (which simulates one family from a population containing the haplotypes 'popHaplos') and *simRecombHap* (which simulates the recombination of haplotypes).

II. **Alleles per Marker**: The second step is the generation of the number of different alleles in each marker. Users can specify the vector *numAlleles* with the desired number of alleles in each marker. If *numAlleles* is NULL, for markers of character type, two alleles are assigned; for markers of numeric type, a number of alleles between 2 and 9 are randomly generated for each marker.

III. **Haplotypes in population**: *alleSimulator* generates the genotypes in a family from the haplotypes in parents. For every family the four haplotypes of the parents must be randomly selected from the pool of possible haplotypes in the population. As with many markers (and/or many alleles per marker) the number of possible different haplotypes can be very high, *alleSimulator* first generates a subset of *nHaplos* haplotypes of the population, from which the parental haplotypes will be selected. By default, the value of *nHaplos* is limited to 1200 haplotypes.

IV. **Data Generation and Concatenation**: In this step, for each family identification data (family ID, parent ID, etc), clinical data (affected/non-affected), and genetical data are simulated. Data of all families are sequentially concatenated in a unique dataframe.

V. **Data Labelling**: The fifth step is the labelling of the previously concatenated data ("famID", "indID", "patID", "matID", "sex", "phen", "markers", "recombNr", "ParentalHap", "MaternalHap").

VI. **Data Conversion**: The sixth step is the conversion of the previously generated data into the most suitable type (integer and/or character).

VII. **Missing Data Generation**: The seventh step is the insertion of missing values in the previously generated dataset (only when users require it). Missing values may be generated taking into account four different factors: *missParProb* (probability of occurrence of a missing genotype in parents), *missOffProb* (probability of occurrence of a missing genotype in a child), *ungenotPars* (probability of a parent to be fully ungenotyped) and *ungenotOffs* (probability of a child to be fully ungenotyped).

VIII. **Function Output**: The last step is the creation of a list containing two different dataframes, for genotype and haplotypes respectively. This may be useful in order to compare simulated haplotypes with later reconstructed haplotypes.

4.2 alleSimulator Examples

Next examples show how `alleSimulator` works:

alleSimulator Example 1: *Simulation of one family with two children where three markers are observed, and containing parental missing data.*

```
> simulatedFam1 <- alleSimulator(1,2,3,missParProb=0.3)
> simulatedFam1[[1]] # Alleles (genotypes) of the 1st simulated family

  famID indID patID matID sex phen Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2
1 FAM01     1     0     0   1    1  <NA>  <NA>     C     G     C     T
2 FAM01     2     0     0   2    2     C     C     C     G     C     T
3 FAM01     3     1     2   1    1     C     T     C     G     C     T
4 FAM01     4     1     2   2    2     C     C     C     C     C     C

> simulatedFam1[[2]] # 1st simulated family haplotypes (without missing values)

  famID indID patID matID sex phen Paternal_Hap Maternal_Hap recomP recomM
1 FAM01     1     0     0   1    1        C-C-C        T-G-T      0      0
2 FAM01     2     0     0   2    2        C-C-C        C-G-T      0      0
3 FAM01     3     1     2   1    1        T-G-T        C-C-C      0      0
4 FAM01     4     1     2   2    2        C-C-C        C-C-C      0      0
```

alleSimulator Example 2: *Same as before but containing offspring missing data instead of parental missing data.*

```
> simulatedFam2 <- alleSimulator(1,2,3,missOffProb=0.3)
> simulatedFam2[[1]] # Alleles (genotypes) of the 2nd simulated family

  famID indID patID matID sex phen Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2
1 FAM01     1     0     0   1    1     G     G     C     C     C     C
2 FAM01     2     0     0   2    1     A     G     C     T     C     T
3 FAM01     3     1     2   1    1     G     G     C     T     C     C
4 FAM01     4     1     2   2    1  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>

> simulatedFam2[[2]] # 2nd simulated family haplotypes (without missing values)

  famID indID patID matID sex phen Paternal_Hap Maternal_Hap recomP recomM
1 FAM01     1     0     0   1    1        G-C-C        G-C-C      0      0
2 FAM01     2     0     0   2    1        G-T-C        A-C-T      0      0
3 FAM01     3     1     2   1    1        G-C-C        G-T-C      0      0
4 FAM01     4     1     2   2    1        G-C-C        G-T-C      0      0
```

5 Workflow

The usual workflow of `alleHap` comprises mainly three stages: *Data Loading*, *Data Imputation* and *Data Haplotyping*. The next subsections will describe each of them.

5.1 Data Loading

The package can be used with either simulated or real data, and can handle missing genetic information. As has been mentioned in section 3, PED files are the default input format for `alleHap`, and although

its loading process is quite simple, it is important to note that a file containing **a large number of markers could slow down the process**. Therefore, to avoid the preceding, it is highly recommended that users *split the data into non-recombinant chunks*, where each chunk should be later loaded by the alleLoader function.

Previously to the loading process, users should check how missing values have been coded in the intended PED file. If those values are different from "-9" or "-99", the parameter *"missingValues"* of alleLoader has to be updated with the corresponding value. Per example, if the PED file has been codified with zeros as missing values, `missingValues=0` must be specified.

5.1.1 alleLoader Function

The `alleLoader` function tries to load the user dataset into a fully compatible format. This dataset can be loaded from an external file or from an R dataframe. For this purpose the function goes through the following four steps:

I. **Internal Functions**: In this step, the auxiliary function *recodeNA* (which recodes pre-specified missing values as NA (Not Available) values) is loaded.

II. **Extension check and data read**: In this step, `alleLoader` first checks if a dataframe, matrix or file name has been passed as an argument. In this last case, the file extension is checked and if it is `".ped"` the dataset is loaded into R as a data.frame. In other cases, the message *"The file must have a .ped extension"* is returned, and data will not be loaded.

III. **Data check**: the number of families, individuals, parents, children, males, females and markers of the dataset are counted. The ranges of paternal IDs, maternal IDs, genotypes and phenotype values are also identified.

IV. **Missing data count**: In this step, the missing/unknown data which may exist in genetic data or in subjects' identifiers are counted.

V. **Function output**: In the final step, the dataset is exported as a data.frame and a summary of previous data counting, ranges, and missing values is printed into the screen.

5.1.2 alleLoader Examples

Next example depicts how `alleLoader` should be used:

alleLoader Example 1: *Loading of a dataset in .ped format with alleles A,C,G,T.*

```
> example1 <- file.path(find.package("alleHap"), "examples", "example1.ped")
>
> example1Alls <- alleLoader(example1)  # Loaded alleles of the example 1

==========================================
===== alleHap package: version 0.9.9 ======
==========================================

Data have been successfully loaded from:
/tmp/Rtmpjpaj2q/Rinst3986828aa8c/alleHap/examples/example1.ped

===== DATA COUNTING ======
Number of families: 50
Number of individuals: 227
```

```
Number of founders: 100
Number of children: 127
Number of males: 118
Number of females: 109
Number of markers: 12
============================

======== DATA RANGES =========
Family IDs: [1,...,50]
Individual IDs: [1,...,8]
Paternal IDs: [0,1]
Maternal IDs: [0,2]
Sex values: [1,2]
Phenotype values: [1,2]
==============================

========= MISSING DATA =========
Missing founders: 0
Missing ID numbers: 0
Missing paternal IDs: 0
Missing maternal IDs: 0
Missing sex: 0
Missing phenotypes: 0
Missing alleles: 0
Markers with missing values: 0
================================
```

```
> example1Alls[1:14,1:12]  # Alleles of the first 17 subjects
```

	famID	indID	patID	matID	sex	phen	Mk1_1	Mk1_2	Mk2_1	Mk2_2	Mk3_1	Mk3_2
1	1	1	0	0	1	1	T	T	C	T	A	A
2	1	2	0	0	2	1	A	T	C	G	C	C
3	1	3	1	2	1	2	A	T	G	T	A	C
4	1	4	1	2	2	1	A	T	C	G	A	C
5	1	5	1	2	2	1	A	T	C	G	A	C
6	2	1	0	0	1	1	A	T	A	G	A	G
7	2	2	0	0	2	1	G	T	A	C	A	G
8	2	3	1	2	2	1	G	T	A	C	A	G
9	2	4	1	2	1	1	A	G	C	G	G	G
10	2	5	1	2	1	1	T	T	A	A	A	A
11	3	1	0	0	1	1	A	T	A	G	G	T
12	3	2	0	0	2	1	C	C	G	G	A	A
13	3	3	1	2	1	2	A	C	G	G	A	G
14	3	4	1	2	1	1	C	T	A	G	A	T

alleLoader Example 2: *Loading of a dataset in .ped format with numerical alleles*

```
> example2 <- file.path(find.package("alleHap"), "examples", "example2.ped")
>
> example2Alls <- alleLoader(example2)  # Loaded alleles of the example 2
```

```
==========================================
===== alleHap package: version 0.9.9 ======
==========================================

Data have been successfully loaded from:
/tmp/Rtmpjpaj2q/Rinst3986828aa8c/alleHap/examples/example2.ped

===== DATA COUNTING ======
Number of families: 11
Number of individuals: 50
Number of founders: 22
Number of children: 28
Number of males: 26
Number of females: 22
Number of markers: 3
============================

======== DATA RANGES =========
Family IDs: [1036,...,1939]
Individual IDs: [1,...,7]
Paternal IDs: [0,1]
Maternal IDs: [0,2,99]
Sex values: [1,2]
Phenotype values: [1,2]
==============================

========= MISSING DATA =========
Missing founders: 0
Missing ID numbers: 0
Missing paternal IDs: 0
Missing maternal IDs: 0
Missing sex: 2
Missing phenotypes: 0
Missing alleles: 42
Markers with missing values: 3
================================

> example2Alls[1:9,]  # Alleles of the first 9 subjects

  famID indID patID matID sex phen Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2
1  1036     1     0     0   1    1   101  1601   101   102   501   502
2  1036     2     0     0   2    1   301   401   301   501   201   301
3  1036     3     1     2   1    2   301  1601   102   501   201   502
4  1036     4     1     2   1    2   301  1601   102   501   201   502
5  1239     1     0     0   1    1    NA    NA    NA    NA    NA    NA
6  1239     2     0     0   2    1    NA    NA    NA    NA    NA    NA
7  1239     3     1     2   2    2   301   401   301   501   201   302
8  1239     4     1     2   2    2   301   401   301   501   201   302
9  1239     5     1     2  NA    1    NA    NA    NA    NA    NA    NA
```

5.2 Data Imputation

The package `alleHap` tries to impute missing genotypes in two ways. The first is implemented in the function `alleImputer` that proceeds marker by marker through the members of a family. The second is carried out by the function `alleHaplotyper` (that takes into account the haplotypes) being the imputation performed only in those cases where there is only one genotype/haplotype structure compatible with both observed and missing genotypes in parents and children. Both imputation mechanisms lead to an unequivocal (and deterministic) imputation of the missing genotypes.

5.2.1 alleImputer Function

`alleImputer` sorts the alleles of each family marker (when possible) and then imputes the missing values. In order to perform data imputation, this function has been developed in six steps:

I. **Internal Functions**: In this step, all the necessary functions are loaded. The most important ones are: *mkrImputer* (which performs the imputation of a marker), *famImputer* (which imputes all the markers of a family) and *famsImputer* (which imputes all given families).

II. **Data Loading**: The second step tries to load user's data into a fully compatible format by means of the `alleLoader` function.

III. **Imputation**: This is the most important step of the `alleImputer` function. The imputation is performed marker by marker and then, family by family. The marker imputation is implemented by the `mkrImputer` internal function which operates in two stages: children imputation and parent imputation. Given a marker with missing values, these can be imputed only either the genotypes of a parent and/or a child are homozygous. If in a marker, one parent has missing alleles and the other not, and the heterozygous alleles of children are not present in the complete parent, those alleles are imputed to the other parent.

   If a parent is homozygous, all children receive that allele and so it can be imputed in children with missing values. And conversely, if a child is homozygous, the corresponding allele must be present in both parents, and so it can be imputed in a parent that lacks it. Also, when only a parent has missing values, and there is one (or two) allele in children not present in the parent with missing values, that allele can be imputed to the parent with missing values.

IV. **Data Summary**: Once the imputation is done, a summary of the imputed data are collected.

V. **Data Storing**: In this step, the imputed data are stored in the same path where the PED file was located.

VI. **Function Output**: In this final step, if *dataSummary=TRUE* an imputation summary is printed out. Also, if *invisibleOutput=FALSE*, imputed data are directly showed in the R console"

Incidence messages can be shown, if they are detected. These incidences can be:
a) *"Some children have no common alleles with a parent"*
b) *"More alleles than possible in this marker"*
c) *"Some children have alleles not present in parents"*
d) *"Some homozygous children are not compatible in this marker"*
e) *"Three or more unique heterozygous children share the same allele"*
f1) *"Heterozygous parent and more than two unique homozygous children"*
f2) *"Heterozygous parent, four unique alleles and more than one unique homozygous children"*
f3) *"Homozygous parent and more than two unique children"*
g1) *"More than four unique children genotypes in the family"*
g2) *"Homozygous genotypes and four unique alleles in children"*

5.2.2 alleImputer Examples

Next examples show how `alleImputer` works:

alleImputer Example 1: *Deterministic imputation for familial data containing parental missing values.*

```
> ## Simulation of a family containing parental missing data
> simulatedFam1 <- alleSimulator(1,2,3,missParProb=0.6)
> # Simulated alleles
> simulatedFam1[[1]]

  famID indID patID matID sex phen Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2
1 FAM01     1     0     0   1    2  <NA>  <NA>     A     G     A     A
2 FAM01     2     0     0   2    1  <NA>  <NA>     A     G  <NA>  <NA>
3 FAM01     3     1     2   2    1     T     T     A     G     A     G
4 FAM01     4     1     2   2    2     T     T     A     A     A     A

> ## Genotype imputation of previously simulated data
> imputedFam1 <- alleImputer(simulatedFam1[[1]])

==========================================
===== alleHap package: version 0.9.9 ======
==========================================

Data have been successfully loaded from:
/tmp/Rtmpjpaj2q/Rbuild398324ade82/alleHap/vignettes

===== DATA COUNTING ======
Number of families: 1
Number of individuals: 4
Number of founders: 2
Number of children: 2
Number of males: 1
Number of females: 3
Number of markers: 3
============================

======== DATA RANGES =========
Family ID: FAM01
Individual IDs: [1,...,4]
Paternal IDs: [0,1]
Maternal IDs: [0,2]
Sex values: [1,2]
Phenotype values: [1,2]
==============================

========= MISSING DATA =========
Missing founders: 0
Missing ID numbers: 0
Missing paternal IDs: 0
Missing maternal IDs: 0
```

```
Missing sex: 0
Missing phenotypes: 0
Missing alleles: 6
Markers with missing values: 2
================================

======= IMPUTATION SUMMARY =======
0 markers (0 alleles) have been
turned into missing in 0 families
due to familial inconsistencies.
Alleles initially missing: 6
Number of imputed alleles: 4
Imputation rate: 0.67
Imputation time: 0
==================================

> # Imputed alleles (markers)
> imputedFam1$imputedMkrs

  famID indID patID matID sex phen Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2
1 FAM01     1     0     0   1    2     T  <NA>     A     G     A     A
2 FAM01     2     0     0   2    1     T  <NA>     A     G     A     G
3 FAM01     3     1     2   2    1     T     T     A     G     A     G
4 FAM01     4     1     2   2    2     T     T     A     A     A     A
```

alleImputer Example 2: *Deterministic imputation for familial data containing offspring missing values.*

```
> ## Simulation of two families containing offspring missing data
> simulatedFam2 <- alleSimulator(2,2,3,missOffProb=0.6)
>
> # Simulated alleles
> simulatedFam2[[1]]

  famID indID patID matID sex phen Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2
1 FAM01     1     0     0   1    1     A     G     A     A     C     C
2 FAM01     2     0     0   2    1     A     A     A     A     C     C
3 FAM01     3     1     2   2    1  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
4 FAM01     4     1     2   1    1  <NA>  <NA>     A     A  <NA>  <NA>
5 FAM02     1     0     0   1    1     A     G     A     G     C     T
6 FAM02     2     0     0   2    1     G     G     A     A     C     T
7 FAM02     3     1     2   1    1  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
8 FAM02     4     1     2   1    1  <NA>  <NA>     A     G  <NA>  <NA>

> ## Genotype imputation of previously simulated data
> imputedFam2 <- alleImputer(simulatedFam2[[1]], dataSummary=FALSE)
>
> # Imputed alleles (markers)
> imputedFam2$imputedMkrs
```

```
  famID indID patID matID sex phen Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2
1 FAM01     1     0     0   1    1     A     G     A     A     C     C
2 FAM01     2     0     0   2    1     A     A     A     A     C     C
3 FAM01     3     1     2   2    1  <NA>     A     A     A     C     C
4 FAM01     4     1     2   1    1  <NA>     A     A     A     C     C
5 FAM02     1     0     0   1    1     A     G     A     G     C     T
6 FAM02     2     0     0   2    1     G     G     A     A     C     T
7 FAM02     3     1     2   1    1  <NA>     G  <NA>     A  <NA>  <NA>
8 FAM02     4     1     2   1    1  <NA>     G     A     G  <NA>  <NA>
```

5.3 Data Haplotyping

At this stage, the corresponding haplotypes of the biallelic pedigree databases are generated. To accomplish this task, based on the user's knowledge of the intended genomic region to analyze, it is necessary to **slice the data into non-recombinant chunks** in order to perform the haplotype reconstruction to each of them.

Users may choose among several symbols in order to specify the non-identified and missing values in the haplotypes. It is also possible to define the character which will be used as a separator between alleles when generating the corresponding haplotypes.

5.3.1 alleHaplotyper Function

`alleHaplotyper` creates the haplotypes family by family taking into account the previously imputed genotypes, along with the matrix IDS. In order to generate the haplotypes, this function has been developed in six steps:

I. **Internal Functions**: In this step, numerous functions to reconstruct the haplotypes were implemented, being the most important the *famHaplotyper* function (which if responsible for carrying out the haplotyping per family), *famsHaplotyper* (which reconstructs the haplotypes for multiple families) and *summarizeData* (which generates a summary of the haplotyped data).

II. **Re-Imputation**: This step calls the `alleImputer` function which performs the imputation marker by marker and then, family by family.

III. **Haplotyping**: This part is the most important of `alleHaplotyper`, since it tries to solve the haplotypes when possible. The process is the following: once each family genotype has been imputed marker by marker, those markers containing two unique heterozygous alleles (both in parents and offspring) are excluded from the process. Then, an IDentified/Sorted (IDS) matrix is generated per family. Later, the internal function `famHaplotyper` tries to solve the haplotypes of each family, comparing the information between parents and children in an iterative and reciprocal way. When there are not genetic data in both parents and there are two or more "unique" offspring (not twins or triplets), the internal functions `makeHapsFromThreeChildren` and `makeHapsFromTwoChildren` try to solve the remaining data. Finally, the HoMoZygosity (HMZ) matrix is updated, and the excluded markers are again included. *Even if both parental alleles are missing in each marker, it is possible to reconstruct the family haplotypes, identifying the corresponding children's haplotypes, although* ***in some cases their parental provenance will be unknown****.*

IV. **Data Summary**: Once the data haplotyping is done, a summary data is collected.

V. **Data Storing**: In this step, the imputed data are stored in the same path where the PED file was located.

VI. **Function Output**: In this final step, a summary of the generated data may be printed out, if *dataSummary*=TRUE. All the results can be directly returned, whether *invisibleOutput* is deactivated. Incidence messages can also be shown, if they are detected. These may be caused by haplotype recombination on children, genotyping errors or inheritance from non-declared parents.

The messages shown in such cases are:

- *"Not enough informative markers"*
- *"Genotyping error or recombination event in marker K and/or subject S"*
- *"Genotyping error, recombination event or inheritance from non-declared parent"*
- *"Parental information is not compatible with haplotypes found in children"*
- *"Haplotypes in one child are not compatible with the haplotypes found in the rest of the offspring"*
- *"Less than three children detected. Haplotypes can not be generated at this stage"*
- *"Multiple compatible haplotypes in parents"*

The final output the `alleHaplotyper` is a list comprised of five elements: *imputedMkrs* (which contains the preliminary imputed marker's alleles), *IDS* (which includes the resulting IDentified/Sorted matrix), *reImputedAlls* (which includes the re-imputed[1] alleles) and *haplotypes* (which stores the reconstructed haplotypes) and *haplotypingSummary* (which shows a summary of the haplotyping process).

5.3.2 alleHaplotyper Examples

Next examples depict how `alleHaplotyper` works:

alleHaplotyper Example 1: *Haplotype reconstruction for a dataset containing parental missing data.*

```
> ## Simulation of families containing parental missing data
> simulatedFams1 <- alleSimulator(2,3,6,missParProb=0.2,ungenotPars=0.3)
> ## Haplotype reconstruction of previously simulated data
> fams1List <- alleHaplotyper(simulatedFams1[[1]])

==========================================
===== alleHap package: version 0.9.9 ======
==========================================

Data have been successfully loaded from:
/tmp/Rtmpjpaj2q/Rbuild398324ade82/alleHap/vignettes

===== DATA COUNTING ======
Number of families: 2
Number of individuals: 10
Number of founders: 4
Number of children: 6
Number of males: 6
Number of females: 4
Number of markers: 6
==========================
```

[1]The term "reimpute" does not mean that there is a re-imputation of what is already imputed, but rather new alleles can be added (imputed) by the `alleImputer` function.

```
======== DATA RANGES =========
Family IDs: [FAM01,...,FAM02]
Individual IDs: [1,...,5]
Paternal IDs: [0,1]
Maternal IDs: [0,2]
Sex values: [1,2]
Phenotype values: [1,2]
==============================

========= MISSING DATA =========
Missing founders: 0
Missing ID numbers: 0
Missing paternal IDs: 0
Missing maternal IDs: 0
Missing sex: 0
Missing phenotypes: 0
Missing alleles: 48
Markers with missing values: 6
================================

======= IMPUTATION SUMMARY =======
0 markers (0 alleles) have been
turned into missing in 0 families
due to familial inconsistencies.
Alleles initially missing: 48
Number of imputed alleles: 28
Imputation rate: 0.58
Imputation time: 0.01
==================================

========= HAPLOTYPING SUMMARY ==========
Re-imputation rate: 0.7083
Proportion of phased alleles: 0.6667
Proportion of non-phased alleles: 0.3333
Proportion of missing haplotypes: 0
Proportion of partial haplotypes: 1
Proportion of full haplotypes: 0
Haplotyping time: 0.015
========================================

> # Original data
> simulatedFams1[[1]][,-(1:6)]

   Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2 Mk4_1 Mk4_2 Mk5_1 Mk5_2 Mk6_1 Mk6_2
1   <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
2   <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
3      C     T     T     T     C     C     C     C     G     G     G     G
4      C     C     C     T     C     C     C     T     G     G     G     G
5      C     C     C     C     C     C     C     T     A     G     G     G
6   <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
```

```
7   <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
8      T     T     T     T     C     T     T     T     A     G     T     T
9      C     C     T     T     C     T     C     T     A     A     G     T
10     C     T     T     T     T     T     C     T     A     A     T     T

> # Re-imputed alleles
> fams1List$reImputedAlls[,-(1:6)]

   Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2 Mk4_1 Mk4_2 Mk5_1 Mk5_2 Mk6_1 Mk6_2
1   <NA>     C     T     C     C     C     C  <NA>     G  <NA>     G     G
2   <NA>     C     T     C     C     C     C  <NA>     G  <NA>     G     G
3      C     T     T     T     C     C     C     C     G     G     G     G
4      C     C     C     T     C     C     C     T     G     G     G     G
5      C     C     C     C     C     C     C     T     A     G     G     G
6      T     C     T     T     T  <NA>     T  <NA>  <NA>     A     T  <NA>
7      T     C     T     T     T  <NA>     T  <NA>  <NA>     A     T  <NA>
8      T     T     T     T     C     T     T     T     A     G     T     T
9      C     C     T     T     C     T     C     T     A     A     G     T
10     C     T     T     T     T     T     C     T     A     A     T     T

> # Reconstructed haplotypes
> fams1List$haplotypes

   famID indID patID matID sex phen   hap1   hap2
1  FAM01     1     0     0   1    1 ?TCCGG CCC??G
2  FAM01     2     0     0   2    1 ?TCCGG CCC??G
3  FAM01     3     1     2   1    1 ?TCCGG ?TCCGG
4  FAM01     4     1     2   1    1 C?C?GG C?C?GG
5  FAM01     5     1     2   1    2 CCC??G CCC??G
6  FAM02     1     0     0   1    1 TT?T?T CT??A?
7  FAM02     2     0     0   2    2 TT?T?T CT??A?
8  FAM02     3     1     2   2    1 TT?T?T TT?T?T
9  FAM02     4     1     2   2    1 CT??A? CT??A?
10 FAM02     5     1     2   1    1 ?TT?AT ?TT?AT
```

alleHaplotyper Example 2: *Haplotype reconstruction for a dataset containing offspring missing data.*

```
> ## Simulation of families containing offspring missing data
> simulatedFams2 <- alleSimulator(2,3,6,missOffProb=0.3,ungenotOffs=0.2)
> ## Haplotype reconstruction of previously simulated data
> fams2List <- alleHaplotyper(simulatedFams2[[1]],dataSummary=FALSE)
> # Original data
> simulatedFams2[[1]][,-(1:6)]

   Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2 Mk4_1 Mk4_2 Mk5_1 Mk5_2 Mk6_1 Mk6_2
1      C     C     C     T     A     A     C     T     A     G     G     G
2      T     T     C     C     C     C     C     C     A     A     G     G
3   <NA>  <NA>     C     C     A     C  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
4      C     T     C     C  <NA>  <NA>     C     T     A     G     G     G
```

```
5   <NA>  <NA>     C     T  <NA>  <NA>  <NA>  <NA>     A     A     G     G
6      C     T     C     C     A     C     T     T     A     G     A     G
7      T     T     T     T     A     C     C     T     A     A     A     G
8      C     T     C     T  <NA>  <NA>     T     T     A     A     A     G
9   <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
10     C     T     C     T     C     C     T     T     A     A     A     G

> # Re-imputed alleles
> fams2List$reImputedAlls[,-(1:6)]

   Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2 Mk4_1 Mk4_2 Mk5_1 Mk5_2 Mk6_1 Mk6_2
1      C     C     C     T     A     A     T     C     G     A     G     G
2      T     T     C     C     C     C     C     C     A     A     G     G
3      C     T     C     C     A     C     T     C     G     A     G     G
4      C     T     C     C     A     C     T     C     G     A     G     G
5      C     T     T     C     A     C     C     C     A     A     G     G
6      C     T     C     C     C     A     T     T     A     G     A     G
7      T     T     T     T     C     A     T     C     A     A     A     G
8      C     T     C     T     C     C     T     T     A     A     A     G
9   <NA>     T     C     T  <NA>  <NA>     T  <NA>  <NA>     A  <NA>  <NA>
10     C     T     C     T     C     C     T     T     A     A     A     G

> # Reconstructed haplotypes
> fams2List$haplotypes

   famID indID patID matID sex phen   hap1   hap2
1  FAM01     1     0     0   1    1 CCATGG CTACAG
2  FAM01     2     0     0   2    1 TCCCAG TCCCAG
3  FAM01     3     1     2   1    2 CCATGG TCCCAG
4  FAM01     4     1     2   2    1 CCATGG TCCCAG
5  FAM01     5     1     2   2    2 CTACAG TCCCAG
6  FAM02     1     0     0   1    1 CCCTA? TCATG?
7  FAM02     2     0     0   2    1 TTCTA? TTACA?
8  FAM02     3     1     2   2    1 CCCTA? TTCTA?
9  FAM02     4     1     2   1    1 ?C?T?? TT??A?
10 FAM02     5     1     2   1    1 CCCTA? TTCTA?
```

alleHaplotyper Example 3: *Haplotype reconstruction of a family containing parental and offspring missing data from a PED file.*

```
> ## PED file path
> family3path <- file.path(find.package("alleHap"), "examples", "example3.ped")
>
> ## Loading of the ped file placed in previous path
> family3Alls <- alleLoader(family3path,dataSummary=FALSE)
>
> ## Haplotype reconstruction of previously loaded data
> family3List <- alleHaplotyper(family3Alls,dataSummary=FALSE)
>
>
```

```
> # Original data
> family3Alls

  famID indID patID matID sex phen Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2 Mk4_1
1     1     1     0     0   2    1     C     T  <NA>  <NA>  <NA>  <NA>  <NA>
2     1     2     0     0   2    1  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
3     1     3     1     2   1    2     C     C     A     G     A     T     A
4     1     4     1     2   1    2     C     T     A     C  <NA>  <NA>     A
5     1     5     1     2   1    2     C     T     A     G     C     T  <NA>
6     1     6     1     2   1    2     C     T     A     G     C     T     A
7     1     7     1     2   2    1  <NA>  <NA>  <NA>  <NA>     C     G     A
  Mk4_2 Mk5_1 Mk5_2 Mk6_1 Mk6_2 Mk7_1 Mk7_2 Mk8_1 Mk8_2
1  <NA>  <NA>  <NA>     A     C  <NA>  <NA>  <NA>  <NA>
2  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>  <NA>
3     A     G     T     A     C     A     C     A     G
4     T     C     G     C     C     C     T     C     G
5  <NA>     G     T     A     C     A     C     A     A
6     A     G     T     A     C     A     C     A     A
7     T     C     G  <NA>  <NA>     C     T  <NA>  <NA>

> # Re-imputed alleles
> family3List$reImputedAlls

  famID indID patID matID sex phen Mk1_1 Mk1_2 Mk2_1 Mk2_2 Mk3_1 Mk3_2 Mk4_1
1     1     1     0     0   2    1     C     T     G     C     T     G     A
2     1     2     0     0   2    1     C     T     A     A     A     C     A
3     1     3     1     2   1    2     C     C     G     A     T     A     A
4     1     4     1     2   1    2     T     C     C     A     G     A     T
5     1     5     1     2   1    2     C     T     G     A     T     C     A
6     1     6     1     2   1    2     C     T     G     A     T     C     A
7     1     7     1     2   2    1     T     T     C     A     G     C     T
  Mk4_2 Mk5_1 Mk5_2 Mk6_1 Mk6_2 Mk7_1 Mk7_2 Mk8_1 Mk8_2
1     T     T     C     A     C     A     T     A     C
2     A     G     G     C     C     C     C     G     A
3     A     T     G     A     C     A     C     A     G
4     A     C     G     C     C     T     C     C     G
5     A     T     G     A     C     A     C     A     A
6     A     T     G     A     C     A     C     A     A
7     A     C     G     C     C     T     C     C     A

> # Reconstructed haplotypes
> family3List$haplotypes

  famID indID patID matID sex phen     hap1     hap2
1     1     1     0     0   2    1 CGTATAAA TCGTCCTC
2     1     2     0     0   2    1 CAAAGCCG TACAGCCA
3     1     3     1     2   1    2 CGTATAAA CAAAGCCG
4     1     4     1     2   1    2 TCGTCCTC CAAAGCCG
5     1     5     1     2   1    2 CGTATAAA TACAGCCA
6     1     6     1     2   1    2 CGTATAAA TACAGCCA
7     1     7     1     2   2    1 TCGTCCTC TACAGCCA
```

References

[BB09] Brian L Browning and Sharon R Browning. A unified approach to genotype imputation and haplotype-phase inference for large data sets of trios and unrelated individuals. *The American Journal of Human Genetics*, 84(2):210–223, 2009.

[BB11] Sharon R Browning and Brian L Browning. Haplotype phasing: existing methods and new developments. *Nature Reviews Genetics*, 12(10):703–714, 2011.

[BWSD+07] Tanya Y Berger-Wolf, Saad I Sheikh, Bhaskar DasGupta, Mary V Ashley, Isabel C Caballero, Wanpracha Chaovalitwongse, and S Lahari Putrevu. Reconstructing sibling relationships in wild populations. *Bioinformatics*, 23(13):49–56, 2007.

[dBMS+06] Paul IW de Bakker, Gil McVean, Pardis C Sabeti, Marcos M Miretti, Todd Green, Jonathan Marchini, Xiayi Ke, Alienke J Monsuur, Pamela Whittaker, Marcos Delgado, et al. A high-resolution hla and snp haplotype map for disease association studies in the extended human mhc. *Nature genetics*, 38(10):1166–1172, 2006.

[MCH+13] S. J. Mack, P. Cano, J. A. Hollenbach, J. He, C. K. Hurley, D. Middleton, M. E. Moraes, S. E. Pereira, J. H. Kempenich, E. F. Reed, M. Setterholm, A. G. Smith, M. G. Tilanus, M. Torres, M. D. Varney, C. E. M. Voorter, G. F. Fischer, K. Fleischhauer, D. Goodridge, W. Klitz, A.-M. Little, M. Maiers, S. G. E. Marsh, C. R. Müller, H. Noreen, E. H. Rozemuller, A. Sanchez-Mazas, D. Senitzer, E. Trachtenberg, and Marcelo Fernandez-Vina. Common and well-documented hla alleles: 2012 update to the cwd catalogue. *Tissue Antigens*, 81(4):194–203, 2013.

[PNTB+07] Shaun Purcell, Benjamin Neale, Kathe Todd-Brown, Lori Thomas, Manuel AR Ferreira, David Bender, Julian Maller, Pamela Sklar, Paul IW De Bakker, Mark J Daly, et al. Plink: a tool set for whole-genome association and population-based linkage analyses. *The American Journal of Human Genetics*, 81(3):559–575, 2007.