Sequoia User Guide

Reconstruction of multi-generational pedigrees from SNP data

Jisca Huisman (jisca.huisman @ gmail.com)

17 May 2024

Contents

1 Quick-start examples

1.1 Simulated Data

An example pedigree and associated life history data are provided with the package, which can be used to try out some of the functions. This fictional pedigree consists of 5 generations with interconnected half-sib clusters (Pedigree II in Huisman (2017)).

```r
# install.packages("sequoia")  # only required first time
library(sequoia)              # load the package
#
# get the example pedigree and life history data
data(Ped_HSg5, LH_HSg5, package="sequoia")
tail(Ped_HSg5)    # have a look at the data
#
# simulate genotype data for 200 SNPs; 40% parents non-genotyped
Geno <- SimGeno(Ped = Ped_HSg5, nSnp = 200, SnpError=0.001, ParMis=0.4)
#
# run sequoia; duplicate check & parentage assignment only at first
ParOUT <- sequoia(GenoM = Geno,
                  LifeHistData = LH_HSg5,
                  Module = "par")

names(ParOUT)  # output: a list with the following elements
# [1] "Specs"  "AgePriors"  "LifeHist"  "PedigreePar"  "MaybeParent"
# "TotLikParents"
#
# run sibship clustering & grandparent assignment
# optional: use parents assigned above (in 'ParOUT$PedigreePar')
SeqOUT <- sequoia(GenoM = Geno,
                  SeqList = ParOUT,
                  Module = "ped")
#
# compare the assigned real and dummy parents to the true pedigree
chk <- PedCompare(Ped1 = Ped_HSg5, Ped2 = SeqOUT$Pedigree)
chk$Counts["TT",,]
#
# save results: single compressed .RData file, and/or folder with text files
save(SeqOUT, Geno, OtherStuff, file="Sequoia_output_date.RData")
writeSeq(SeqList = SeqOUT, GenoM = Geno, folder = "Sequoia-OUT")
```

For a detailed walk-through of this example, see here

1.2 Real Data

First get a subset of SNPs which are as informative (high MAF, high call rate), reliable (low error rate), and independent (low LD) as possible. Ideally around 400 – 700 for full pedigree reconstruction; fewer are necessary for only parentage assignment, while more may be necessary with very high levels of polygamy or inbreeding. In addition you need a dataframe with the sex and birth/hatching year of as many individuals as possible, in arbitrary order.

It is often prudent to first do a quick partial run to check for duplicates and genetic mismatches between known parent-offspring pairs, before running full pedigree reconstruction.

```r
# read in genotype data if already coded as 0/1/2, with missing=-9:
Geno <- as.matrix(read.csv("mydata.csv", header=FALSE, row.names=1))
```

```
CheckGeno(Geno)
# or read in from several other input formats (incl. vcf):
Geno <- GenoConvert(InFile = "mydata.ped", InFormat="ped")
#
# read in lifehistory data: ID-Sex-birthyear, column names ignored
LH <- read.table("LifeHistoryData.txt", header=T)
#
# duplicate check & parentage assignment (takes few minutes)
ParOUT <- sequoia(GenoM = Geno,  LifeHistData = LH_HSg5,  Err=0.005,
                  Module="par", quiet = FALSE, Plot = TRUE)
#
ParOUT$DupGenotype  # inspect duplicates (intentional or accidental)
#
# compare assigned parents to field pedigree (check column order!)
FieldPed <- read.table("FieldPed.txt", header=T)
PC.par <- PedCompare(Ped1 = FieldPed[, c("id", "dam", "sire")],
                     Ped2 = ParOUT$PedigreePar)
PC.par$Counts["TT",,]
#
# ..........................................................
# polish dataset:
# remove one indiv. from each duplicate pair
Geno2 <- Geno[!rownames(Geno) %in% ParOUT$DupGenotype$ID2,]
# & drop low call rate samples
# & drop SNPs with high error rate and/or low MAF
stats <- SnpStats(Geno, ParOUT$PedigreePar)
MAF <- ifelse(stats[,"AF"] <= 0.5, stats[,"AF"], 1-stats[,"AF"])
Geno2 <- Geno2[, -which(stats[,"Err.hat"]>0.05 | MAF < 0.1)]
#
Indiv.Mis <- apply(Geno2, 1, function(x) sum(x == -9)) / ncol(Geno2)
Geno2 <- Geno2[Indiv.Mis < 0.2, ]
# check histograms for sensible thresholds, iterate if necessary
#
# ..........................................................
# run full pedigree reconstruction (may take up to a few hours)
SeqOUT <- sequoia(GenoM = Geno2,
                  LifeHistData = LH_HSg5,
                  Module = "ped",
                  Err = 0.001)
#
SummarySeq(SeqOUT)  # inspect assigned parents, prop. dummy parents, etc.
#
# check full sibs, half sibs etc.
Pairwise <- ComparePairs(SeqOUT$Pedigree, patmat = TRUE)
#
# save results: single compressed .RData file, and/or folder with text files
save(SeqOUT, Geno, OtherStuff, file="Sequoia_output_date.RData")
writeSeq(SeqList = SeqOUT, GenoM = Geno, folder = "Sequoia-OUT")
```

2 Background

The core of `Sequoia` is to

- Assign genotyped parents to genotyped individuals ('parentage assignment'), even if the sex or birth year of some candidate parents is unknown;
- Cluster genotyped half- and full-siblings for which the parent is not genotyped into sibships, assigning a 'dummy parent' to each sibship
- Find grandparents to each sibship, both among genotyped individuals and among dummy parents to other sibships.

`Sequoia` provides a conservative hill-climbing algorithm to construct a high-likelihood pedigree from data on hundreds of single nucleotide polymorphisms (SNPs), described in Huisman (2017). Explicit consideration of the likelihoods of alternative relationships (parent-offspring, full siblings, grandparent–grand-offspring, ...) before making an assignment reduces the number of false positives, compared to parentage assignment methods that rely on the likelihood ratio between parent-offspring versus unrelated only (Thompson and Meagher 1987). The heuristic, sequential approach used is considerably quicker than most alternative approaches such as MCMC, and when genetic information is abundant there is little to no loss in accuracy. Typical computation times are a few minutes for parentage assignment, and a few hours for full pedigree reconstruction when not all individuals are genotyped.

A word of caution: the *most likely* relationship is not necessarily the *true* relationship between a pair, due to the random nature of Mendelian segregation, and possible genotyping errors. In addition, the most likely relationship for a *pair* will not necessarily result in the highest *global* likelihood, and may therefore not have been assigned.

3 Key Points

3.1 Mendelian Errors

Parent and offspring will never be opposing homozygotes (one aa, other AA) at a locus, except due to genotyping errors. This metric is easy and quick to calculate even in very large datasets, and provides a good way to subset candidate parents (done by `sequoia` internally), or to check if an existing pedigree is consistent with new genetic data (function `CalcOHLLR(, CalcLLR=FALSE)`).

The count of opposing homozygous (OH) SNPs is not always a reliable indicator to distinguish parents from not-parents. When the genotyping error rate is high, true parent–offspring pairs may have a high OH count, while some full siblings and other close relatives may have a very low OH count. Therefore, `sequoia()` uses OH only as a filter to reduce the number of candidate parents for each individual, and uses likelihood ratios for actual assignments.

3.1.1 Parent-pairs

The number of Mendelian errors in an offspring-mother-father trio includes the OH counts between offspring and each parent, as well as cases where the offspring should have been heterozygous, but isn't (mother aa, father AA, offspring aa or AA), and cases where it is heterozygous, but shouldn't be (when mother and father are identical homozygotes). This metric is used to subset 'compatible' parent-pairs when there are candidate parents from both sexes, or with unknown sex.

3.1.2 Maximum mismatches

The maximum number of Mendelian mismatches, and mismatches between potential duplicates, is from version 2.0 onward calculated by `CalcMaxMismatch()`, based on the (average) genotyping error rate. This parameter can no longer be altered directly, but can be changed via parameters `Err` and `ErrFlavour`.

3.2 Likelihoods

Assignments are based on the likelihoods for all different possible relationships between a pair, calculated conditional on all assignments made up onto that point. If the focal relationship is more likely than all alternatives (by a margin `Tassign`), the assignment is made.

The relationships are condensed into seven categories (Table 1). Under the default settings (`Complx='full'`), these also include their combinations (see section Unusual relationships), e.g. a pair may be both paternal half-siblings and maternal aunt/niece if they were produced by a mother-daughter pair mating with the same male.

Table 1: Main relationships considered

	Relationship
PO	Parent - offspring
FS	Full siblings
HS	Half siblings
GP	Grandparental
FA	Full avuncular (aunt/uncle)
HA	Half avuncular (+ other 3rd degree)
U	Unrelated

To reduce computational time, these likelihoods are only calculated after filtering pairs based on the number of Mendelian errors, their age difference, and a quick calculation of LLR(focal / unrelated) without conditioning on any earlier assigned relatives (must pass `Tfilter`).

For any pair of genotyped individuals, these seven likelihoods can be calculated with `CalcPairLL()`.

3.3 Genotyping errors

3.3.1 Effect

A higher rate of genotyping errors leads to more false negatives and more false positives during pedigree reconstruction, as illustrated in Figure 1. The magnitude of the effect varies: One wrong assignment due to a few genotyping errors in a key individual may have numerous knock-on effects, while dozens of genotyping errors in 'dead end' individuals may have no consequences at all.

The simulations (`SimGeno()`) assume amongst others that genotyping errors are independent from each other and from a sample's call rate, and likely give a more optimistic picture than real data.

3.3.2 Assumed genotyping error rate `Err`

The reduced performance at high genotyping error rates can be somewhat mitigated by running `sequoia()` with a roughly correct presumed genotyping error rate `Err` (Figure 1: compare red vs yellow symbols at an error rate of 1-2%).

Among others, when `Err` is much lower than the actual genotyping error rate, some true parent-offspring pairs will be assigned as full siblings. On the other hand, when `Err` is much higher than the actual genotyping error rate some true full siblings may be classified as parent-offspring, potentially leading to various 'secondary' errors. This trade-off can be explored using `CalcPairLL()` if some known parent-offspring and full sibling pairs are available.

NOTE that the default assumed genotyping error rate of 0.01% is based on data from a SNP array after stringent quality control; in many cases the genotyping error rate will be (much) higher [1]. It is typically worthwhile to explore if and how results change when you vary this parameter, possibly in combination with non-default values for the thresholds `Tfilter` and `Tassign`. Do keep in mind the relative consequences of potential misassignments (false positives) versus non-assignments (false negatives) during later use of the pedigree.

[1]but changing the default value is likely to cause problems for people re-using old code.

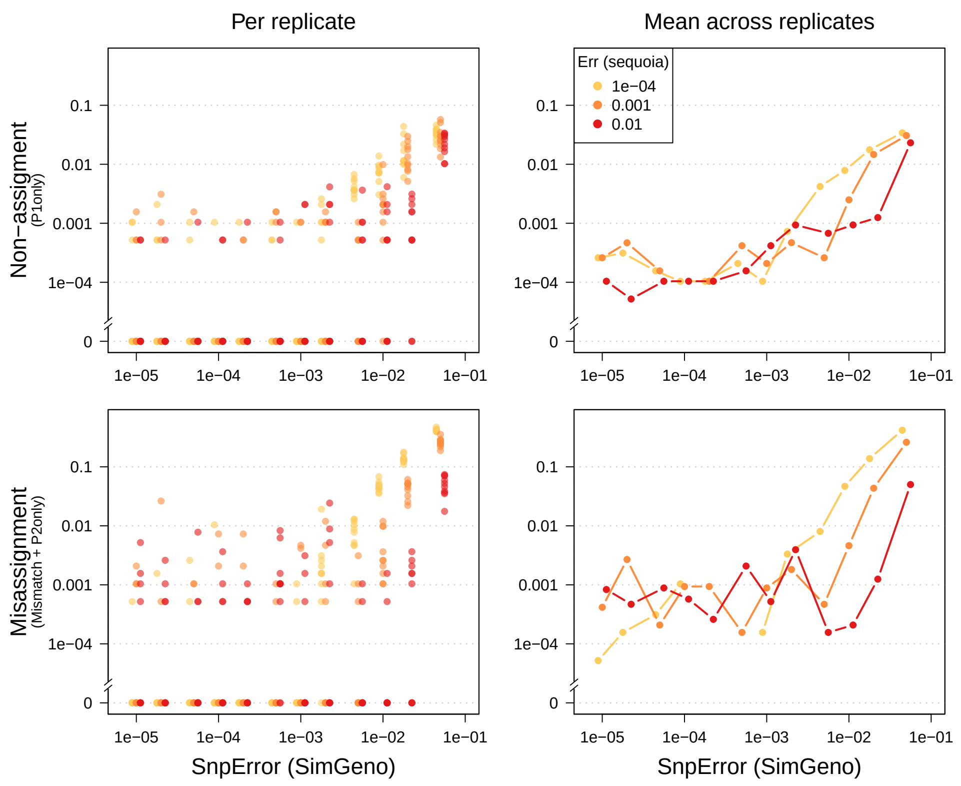


Figure 1: High simulated genotyping error rate (x-axes) increases both false negatives (top) and false positives (bottom). A roughly correct presumed error rate (colours) improves performance. Results from pedigree `Ped_HSg5` with 200 SNPs, call rate 0.99, 40% of parents non-genotyped. Diamonds: means across 10 replicates.

3.3.3 Performance with high genotyping error rates

`sequoia` does not cope well with high rates of genotyping errors, as the sequential approach (see Background) can lead to a cascading avalanche of assignment errors. If the rate is around 1%, be cautious when interpreting the results and please use `EstConf()` to estimate the assignment error. With genotyping error rates above 5% I would strongly suggest to use MCMC-based approaches, or other approaches more robust to poor genotyping quality.

Assignment of genotyped parents to genotyped offspring is fairly robust to low SNP number and high genotyping errors (left-most column in Figure 2), while pedigree links involving a dummy individual are more prone to mis-assignment and non-assignment (columns 2-4 in Figure 2). In these simulations, all SNPs are completely independent, call rate is high, and founders are unrelated; in real data the overall performance is probably lower but the patterns will be similar.

3.3.4 Effect on runtime

Computational time tends to increases with both simulated and assumed genotyping error rate, as the time-saving filtering steps become less effective (Figure 3).

3.3.5 Error matrix

Different genotyping methods are likely to have different error structures (Table 2). Therefore, from version 2.0 sequoia allows fine-scale control over how potential genotyping errors are accounted for. An error matrix with the probability to observe genotype G (columns), conditional on actual genotype g (rows), can now be specified via parameter `ErrFlavour`. The current default (`version2.0`) is given in Table 2, and previous

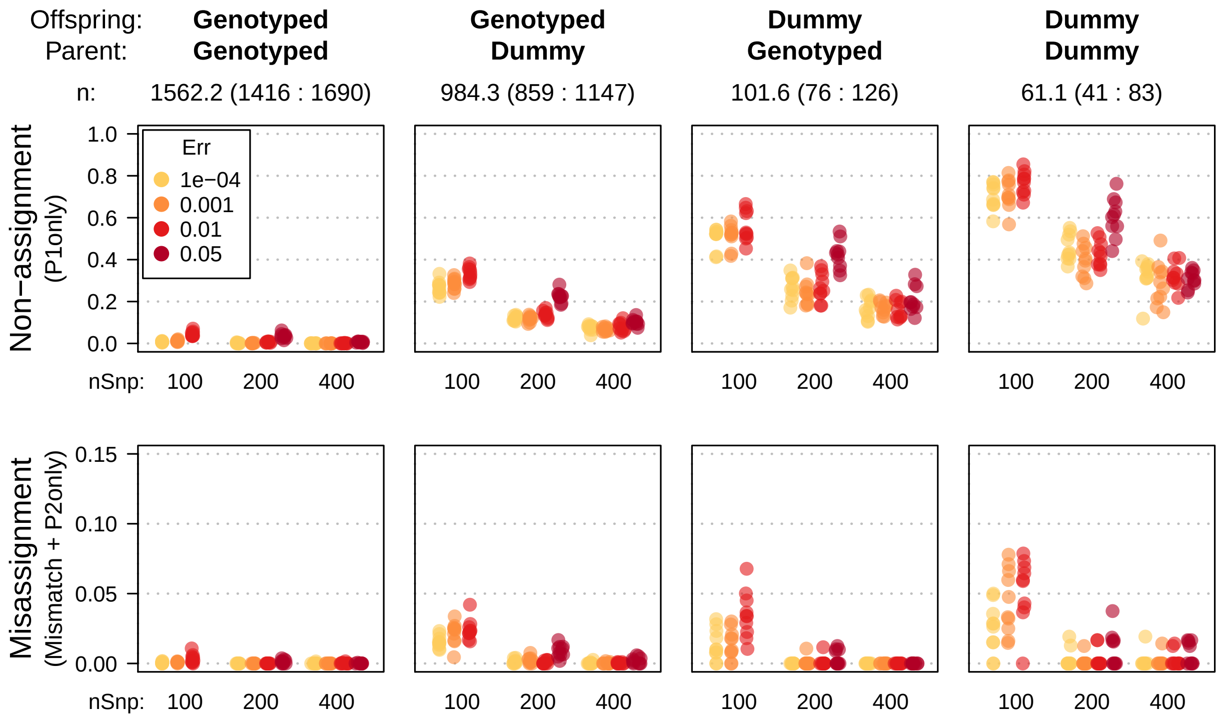


Figure 2: Non-assignment (top) and assignment errors (bottom) for a range of SNP numbers (x-axes) and simulated genotyping errors (colours). `sequoia` presumed error rates ('Err') set equal to simulated error rates. Results for a deer pedigree with overlapping generations and inbreeding; 40% of parents were presumed non-genotyped, call rate=0.99 for remainder.

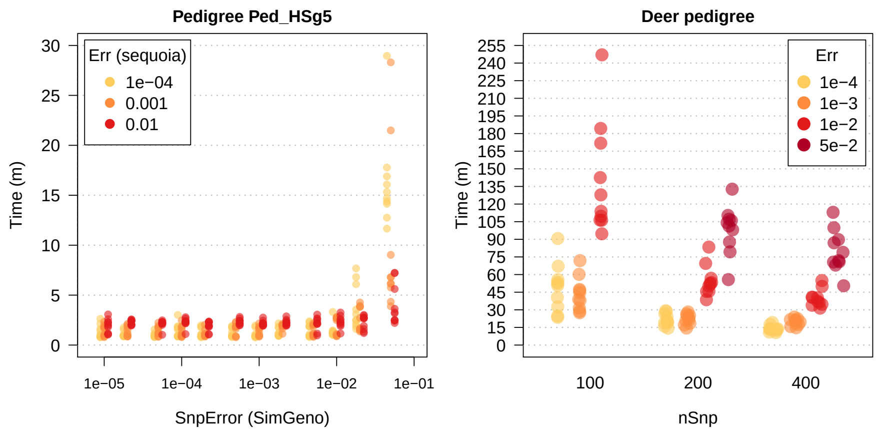


Figure 3: Runtimes are shorter with lower genotyping error rates and more SNPs

defaults are shown in the help file of `ErrToM()`. The slight differences between versions only have any consequences when the error rate is high (> 1).

From version 2.7 onwards, it is also possible to specify the genotyping error structure as a length 3 vector, with probabilities (between brackets = current defaults):

- hom|hom: an actual homozygote is observed as the other homozygote ($(E/2)^2$)

- het|hom: an actual homozygote is observed as heterozygote (E $times(1 - E/2)$)
- hom|het: an actual heterozygote is observed as homozygote ($E/2$)

During pedigree inference, the error rate is presumed equal across all SNPs. It is also assumed that none of the errors (or missingness) are due to heritable mutations.

3.3.6 Estimate

Function `SnpStats()` can estimate the genotyping error rate per SNP, conditional on a provided pedigree and error structure (`ErrFlavour`). These estimated error rates will not be as accurate as from duplicate samples, since a single error in an individual with many offspring will be counted many times, while errors in individuals without parents or offspring will not be counted at all.

3.4 Birth years & Ageprior

Providing individual birth years will typically greatly improve the completeness and speed of pedigree reconstruction. Age data is used to subset the potential relationships between each pair, and is required to orientate parent–offspring pairs the right way around. Birth year information is not strictly required for every individual: parent-parent-offspring trios can be assigned even in absence of any age data, as long as the sex of either parent is known.

'Birth year' should be interpreted loosely, as the time unit (week/month/year/decade/...) of birth/hatching/germination/... .

For individuals (or 'mystery samples') with unknown birth years, the probability that they were born in year y can be estimated using `CalcBYprobs()`. This relies on the (estimated) birth years of their parents and offspring, and the age distribution of parent-offspring pairs as specified in the `age prior`.

3.4.1 Definition & interpretation

The 'agepriors' as used by `sequoia()` is not an official term, nor a proper prior, but a set of age-difference based probability ratios which can be interpreted as

> If I were to pick two individuals with an age difference A, and two individuals at random, how much more likely are the first pair to have relationship R, compared to the second pair?

The value may vary from 0 (impossible), to 1 (just as likely), and typically not far beyond 10 (10x as likely). This quantification allows age differences to be used jointly with genetic information, which is mostly done in a conservative fashion (except when `UseAge='extra'`): only when the focal relationship is the most likely both when considering only genetic data, and when considering genetic + age data, is the assignment made. 'Most likely' applies to *from the set of possible relationships*, i.e. those with an age prior value > 0.

For example, in datasets with only one or two ageclasses, it is worth contemplating whether or not avuncular relationships should be considered as alternatives to siblings. The behaviour can be changed by explicitly declaring generations as `Discrete` (avuncular cannot have age difference of 0), and/or by setting `MaxAgeParent`.

Table 2: Probability of observed genotype (columns) conditional on actual genotype (rows) and per-locus error rate E

	aa	Aa	AA
aa	$(1 - E/2)^2$	$E \times (1 - E/2)$	$(E/2)^2$
Aa	$E/2$	$1 - E$	$E/2$
AA	$(E/2)^2$	$E \times (1 - E/2)$	$(1 - E/2)^2$

You can find extensive details about the ageprior in this separate vignette, including worked examples, mathematical framework, and full range of options.

3.4.2 Implementation

Parentage assignment is run with an ageprior only specifying that parents and offspring cannot be the same age. The resulting 'scaffold pedigree' is then used by `MakeAgeprior()` to estimate the dataset-specific age-difference distributions for each relationship (mother, father, full sibling, maternal and paternal half sibling), which are standardised by the age-difference distribution among all pairs. The resulting ageprior probability ratios are then used during full pedigree reconstruction (Figure 4).

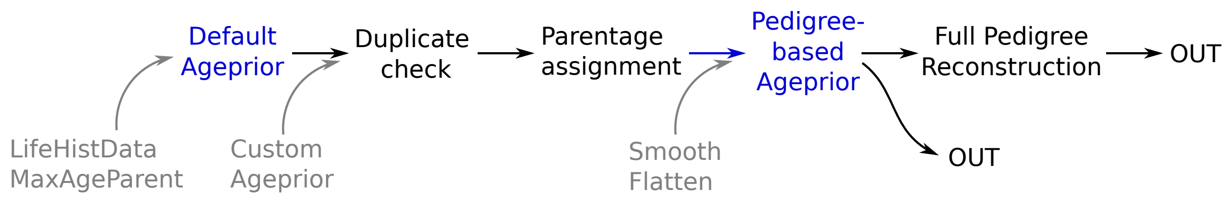


Figure 4: Ageprior pipeline overview

When running `sequoia()`, arguments can be passed to `MakeAgePrior()` via `args.AP`.

3.4.3 Smooth & Flatten

Often not all biologically possible age-differences are present in the scaffold pedigree. Therefore `MakeAgeprior()` by default applies a correction to allow for relatives to differ 1-2 years more in age than observed in its input pedigree, and to smooth out any dips or gaps in the age distributions. However, when such abrupt terminations or gaps are biologically meaningful, these corrections can and should be turned off (`Smooth=FALSE` and/or `Flatten=FALSE`). This is done automatically in case discrete generations are declared (`Discrete=TRUE`) or detected in the scaffold pedigree.

3.4.4 Age-based non-assignment

Any genetically identified parent-offspring pairs and other relatives which are deemed impossible based on their age difference and the age prior matrix, and thus not assigned, can be found by `GetMaybeRel()`. You may then correct or discard their birth years and/or update the ageprior to allow for their age difference (see Customising the ageprior), and re-run pedigree reconstruction.

3.5 Parent LLR

When parentage assignment or pedigree reconstruction is completed (when the total likelihood has asymptoted), a log10-likelihood ratio (LLR) is calculated for each assigned parent-offspring pair. This is

> **The ratio between the likelihood of being parent and offspring, versus the next-most-likely relationship between them, conditional on all other relationships in the pedigree.**

This differs from for example Cervus (Marshall et al. 1998), which returns the ratio between the likelihood that the assigned parent is the parent, versus that the next most likely candidate is the parent.

The LLRs for the dam and sire separately are calculated by temporarily dropping the co-parent (if any), and calculating the likelihoods assuming the co-parent is a random draw from the population. The LLR for the parent-pair is the minimum of the dam and sire LLR, each conditional on the co-parent. To retrieve the likelihoods underlying each ratio, use `CalcPairLL()`; specify column `dropPar1` for single-parent likelihoods.

Since this step can be quite time-consuming, `sequoia()` can be run with `CalcLLR=FALSE`, and `CalcOHLLR()` can be run separately later. That function can also calculate parental LLR's for any existing pedigree, including for any non-genotyped individuals with at least one genotyped offspring.

3.6 Confidence

The parental likelihood ratio does not necessarily indicate how likely it is that the assignment is correct. Pedigree-wide confidence probabilities can, among others, be estimated as follows

- Simulate genotype data according to the reconstructed (or an existing) pedigree, imposing realistic levels of missingness and genotyping errors (`SimGeno()`);
- Reconstruct a pedigree from the simulated data (`sequoia()`);
- Count the number of mismatches between the 'true' (input) pedigree and the pedigree reconstructed from the simulated data (`PedCompare()`).

When repeated at least 10–20 times, the proportion of assignments that is correct then provides an estimate of the confidence probability. This will be an optimistic estimate, as `SimGeno()` assumes all SNPs are independent, that parental genotypes are missing at random, and makes various other simplifying assumptions.

This process is conveniently wrapped in the function `EstConf()`, which calculates separate confidence levels for dams and sires (which may have very different sampling proportions), for genotyped and dummy parents, and for single parents and members of parent-pairs. The process can be rather time consuming, but from version 2.1 onwards it is possible to run on multiple computer cores in parallel.

3.7 Low assignment rate

Pedigree reconstruction with `sequoia()` regularly suffers from a low assignment rate, especially when only a small proportion of parents have been genotyped. This is fundamental to the method: when in doubt, MCMC approaches will return an assignment with a low posterior probability, but `sequoia()` will return no assignment at all.

In `sequoia()`, an incorrect assignment early on may cause a cascade of other incorrect assignments later in the run, and great care is taken to prevent this. On the flip side, when there is limited genetic or birth year information and/or very few parents have been genotyped, the cascade of *correct* assignments, where maternal assignments facilitate paternal assignments and v.v. (Figure 5), may never take off. For such situations, different programs with MCMC based approaches are more suitable.

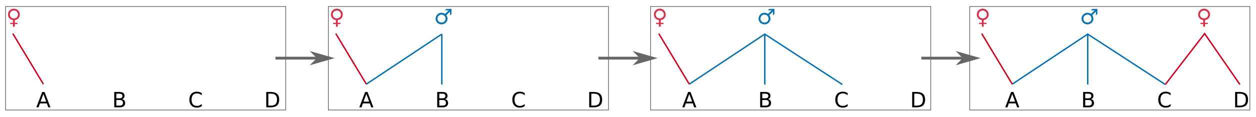


Figure 5: Sibship clustering of individuals A-D

3.7.1 Causes

There are four main causes a pair of true relatives are not assigned, which are not mutually exclusive:

1. There is more than one type of relationship possible between the pair, which cannot be distinguished with certainty (details below);
2. The type of relationship is clear (e.g. half siblings), but it is unclear whether they are maternal or paternal relatives;
3. It is clear the pair are mother and daughter (or grandmother – grand-granddaughter), but it is unclear which of the two is the (grand)mother, due to unknown birth year(s) and absence of a 'complementary' candidate father (and analogous for father-son);
4. There is more than one plausible candidate (dummy) (grand)parents of one sex, all of which are more likely to be a parent than otherwise related, even when considered jointly. This includes the situation where true offspring are among the candidate parents due to unknown birth years.

For more details, see FAQ.

Cause (1) has two common underlying causes, which again are not mutually exclusive:

1a. Half-siblings, grandparent–grand-offspring and full avuncular pairs are genetically indistinguishable, even with perfect genetic data. Assignment of half-sibling pairs is only possible when grand-parental and avuncular relationships can be excluded based on the age-difference of the pair, or when both individuals already have a parent assigned (in which case the likelihoods do differ). 1b. The SNP data is not informative enough to clearly distinguishing between parents and full siblings, between full and half siblings, between half siblings and cousins, etc.. The problem may be specific to a few pairs and due to the randomness of Mendelian segregation, or be widespread due to a low number of SNPs, low MAF, and/or high error rate.

3.7.2 Remedies

Sometimes it is possible to reduce the number of non-assigned parents in the pedigree by providing more life history data, removing SNPs with the highest error rates, or lowering the thresholds `Tfilter` and/or `Tassign`. The latter will reduce the probability of false negatives, but at the same time increase the probability for false positives. If inbreeding and double relationships are rare, using `Complx='simp'` may also improve assignment (see here).

Small adjustments may 'get the snowball rolling' (Figure 5) and greatly increase assignments. Once a sibling pair is assigned, adding additional siblings to the cluster becomes progressively easier as it becomes more and more obvious whether an individual is related to all individuals in the sibship, or to none.

There are functions in the package that may help identify whether there are un-assigned likely relatives in the dataset, and/or the probable cause of non-assignment of presumed relatives:

- `GetMaybeRel()` will identify any pairs that are likely to be 1st or 2nd degree relatives. It can do so as-is, or conditional on a (reconstructed) pedigree. It may also provide clues whether this is e.g. due to unknown birth years or an impossibility to tell the different 2nd degree relationships apart (TopRel = '2nd').
- `CalcPairLL()` will return for a specified set of pairs the likelihoods to be parent-offspring, sibling, ... (Table 1). The pairs may be the output from `GetMaybeRel()`, or e.g. pairs of litter mates. This may e.g. show that it is unclear whether the pair are full or half siblings, but they are probably either one or those. Or it may show the pair are probably either half-siblings or 3rd degree relatives (half-avuncular, cousins, great-grandparent). The latter may be 'declared' half-siblings if the breeding scheme or field data (e.g. being litter mates) makes 3rd degree relatives highly unlikely. You can specify a pedigree and/or ageprior to condition on, and can include a pair multiple times with different age difference or focal relationship to explore their effects on the likelihoods and thereby the LLR.
- use of mitochondrial haplotype sharing data to distinguish between maternal and paternal relatives. See the mtDNA vignette for details.

Table 3: Core functions with selected subset of their required (Y) and optional (+) input, and their output that can be re-used as input in other functions.

Function	GenoM	Pedigree	Pairs	AgePrior	SeqList	Reusable output	Plot function
sequoia	Y			+	+	Pedigree, SeqList	SeqListSummary
GetMaybeRel	Y			+	+	Pairs	PlotRelPairs
CalcOHLLR	Y	Y		+	+		SeqListSummary
CalcPairLL	Y	+	Y	+	+		PlotPairLL
MakeAgePrior		+				AgePrior	PlotAgePrior
SimGeno		Y				GenoM	SnpStats
PedCompare		YY					PlotPedComp
ComparePairs		Y+	+			Pairs	PlotRelPairs

4 Function overview

4.1 Input

- **GenoConvert**
  Read in genotype data from PLINK file, Colony file, or many user-specified formats, and return a matrix in sequoia's (or Colony) format. *Does not support vcf yet.*
- **LHConvert**
  Extract sex and birth year from PLINK file; optionally recode sex to 1=female, 2=male, check consistency with other LifeHistData, or combine family ID and individual ID into FID___IID.
- **CheckGeno**
  Check that the provided genotype matrix is in the correct format, and check for low call rate samples and SNPs
- **SnpStats**
  Calculate per-SNP allele frequency, missingness, and, if a pedigree provided, number of Mendelian errors.
- **CalcMaxMismatch**
  Calculate the maximum expected number of mismatches for duplicate samples, and Mendelian errors for parent-offspring pairs and parent-parent-offspring trios.

4.2 Simulate

- **SimGeno**
  Simulate genotype data for independent SNPs. Specify pedigree, founder MAF, call rate, proportion of non-genotyped parents, genotyping error & error model.
- **MkGenoErrors**
  Add genotyping errors and missingness to genotype data; more fine-scale control than with SimGeno.

4.3 Ageprior

- **MakeAgePrior**
  For various categories of pairwise relatives (R), calculate age-difference (A) based probability ratios $P(A|R)/P(A) = P(R|A)/P(R)$, or how much likelier a relationship is given the age difference. It applies corrections when the skeleton-pedigree contains few/no pairs with known age difference for some relationships.
- **PlotAgePrior**
  Visualise the age-difference based prior probability ratios as a heatmap.

4.4 Pedigree reconstruction

- **sequoia**
  Main function to run parentage assignment and full pedigree reconstruction, calls many of the other functions.
- **GetMaybeRel**
  Identify pairs of individuals likely to be related, but not assigned as such in the provided pedigree. Either search only for potential parent-offspring pairs, or for all 1st and 2nd degree relatives.

4.5 Pedigree check

These functions can be applied to any pedigree, not just pedigrees reconstructed by sequoia. Required input between brackets

- **SummarySeq** (1 pedigree)
  Graphical overview of the assignment rate, the proportion dummy parents, sibship sizes, parental LLR distributions, and Mendelian errors, + tables with pedigree summary statistics.
- **CalcOHLLR** (pedigree + genotypes)
  Count opposite homozygous (OH) loci between parent-offspring pairs and Mendelian errors (ME) between parent-parent-offspring trios, and calculate the parental log-likelihood ratios (LLR).
- **EstConf** (pedigree + genotypes)
  Estimate assignment error rate (false positives & false negatives). Using a reference pedigree, repeatedly simulate genotype data, run sequoia, and compare inferred to reference pedigree.
- **getAssignCat** (1 pedigree)
  Identify which individuals are genotyped, and which can potentially be substituted by a dummy individual. 'Dummifiable' are those non-genotyped individuals with at least 2 genotyped offspring, or at least 1 genotyped offspring and 1 genotyped parent.
- **PedCompare** (2 pedigrees)
  Compare 2 pedigrees, e.g. field and genetically inferred, or reference and inferred-from-simulated-data. Matches dummy parents to non-genotyped parents.
- **CalcRped** (1 pedigree) This is a wrapper for `kinship()` in package `kinship2`.

4.6 Pairwise relationships

- **CalcPairLL** (pairs + genotypes) For each pair, calculate the log10-likelihoods of being various different types of relative, or unrelated.
- **GetRelCat** (1 pedigree)
  Determine the relationship between individual X and all other individuals in the pedigree, going up to 1 or 2 generations back.
- **GetRelM** (pedigree or pairs) Generate a matrix with all pairwise relationships from a pedigree or dataframe with pairs
- **ComparePairs** (1 or 2 pedigrees)
  Compare, count and identify different types of relative pairs between two pedigrees, or within one pedigree. [2]
- **PlotRelPairs** (matrix) plot pairwise relationships between all individuals, as by Colony.

[2]The matrix returned by `DyadCompare` *(Deprecated)* is a subset of the matrix returned here using default settings.

4.7 Miscellaneous

- **PedPolish**
  Ensure all parents & all genotyped individuals are included, remove duplicates, rename columns, and replace 0 by NA or v.v. Can also generate 'filler' parents for software that requires individuals to have either 0 or 2 parents, never 1.
- **getGenerations** For each individual in a pedigree, count the number of generations since its most distant pedigree founder.
- **ErrToM**
  Generate a matrix with the probabilities of observed genotypes (columns) conditional on actual genotypes (rows), or return a function to generate such matrices. The error matrix can be used as input for `sequoia` and `CalcOHLLR`, the error function as input for `SimGeno`
- **writeSeq**
  Write the list with sequoia output in human-readable format, either as a folder with .txt files, or as a many-tabbed excel file. The latter uses R package `xlsx`, which requires java and can (therefore) be cumbersome to install.
- **writeColumns**
  write data.frame or matrix to a text file, using white space padding to keep columns aligned.
- **FindFamilies**
  Add a column with family IDs (FIDs) to a pedigree, with each number denoting a cluster of connected individuals.
- **PedStripFID**
  Reverse the joining of FID and IID in GenoConvert and LHConvert
- ** CalcBYprobs** Estimate the probability that individual i with unknown birth year is born in year y, based on the birthyears of its parents and offspring.

5 `sequoia()` input

5.1 Genotypes

The SNP data should be provided as a numeric matrix `GenoM` with one line per individual, and one column per SNP, with each SNP is coded as 0, 1, 2 copies of the reference allele, or missing (-9). The rownames should be the individual IDs, and column names are ignored.

```r
GenoM <- as.matrix(read.table("MyGenoData.txt", row.names=1, header=FALSE))
```

When the genotype data is in another format, such as Colony input files or PLINK's .ped or .raw files, `GenoConvert()` can be used.

5.1.1 Subset SNPs

Using tens of thousands of SNP markers for pedigree reconstruction is unnecessary, will slow down computation, and may even hamper inferences by their non-independence. Rather, a subset of SNPs with a decent genotyping call rate (e.g. > 0.9), in low linkage disequilibrium (LD) with each other, and with high minor allele frequencies (e.g. MAF > 0.3) ought to be selected first if more than a few hundred SNPs are available. The calculations assume independence of markers, and while low (background) levels of LD are unlikely to interfere with pedigree reconstruction, high levels may give spurious results. The recommended maximum number of SNPs therefore depends on genome size and chromosome number. Markers with a high MAF provide the most information, as although rare allele provide strong evidence when they are inherited, this does not balance out the rarity of such events.

It is advised to 'tweak' the filtering thresholds until a set with a few hundred SNPs (300–700) is created. To assist with this, the function `SnpStats()` gives for each SNP both the allele frequency and the missingness. In addition, when a pedigree is provided (e.g. an existing one, or from a preliminary parentage-only run), the number of Mendelian errors per SNP is calculated and used to estimate the genotyping error rate.

5.1.2 PLINK

Since the full dataset may be too large to easily load into R, creating a subset of SNPs can for example be done using PLINK:

```
plink --file mydata --geno 0.1 --maf 0.3 --indep 50 5 2
```

which on a windows machine is equivalent to running inside R

```
system("cmd", input = "plink --file mydata --maf 0.3 --indep 50 5 2")
```

This will create a list of SNPs with a missingness below 0.1, a minor allele frequency of at least 0.3, and which in a window of 50 SNPs, sliding by 5 SNPs per step, have a VIF of maximum 2. VIF, or variance inflation factor, is $1/(1-r^2)$. For further details, see the plink website.

The resulting list (`plink.prune.in`) can be used to create the genotype file used as input for Sequoia, with SNPs codes as 0, 1, 2, or NA, with the command

```
plink --file mydata --extract plink.prune.in --recodeA --out inputfile_for_sequoia
```

This will create a file with the extension .RAW, which can be converted to the required input format using

```
GenoM <- GenoConvert(InFile = "inputfile_for_sequoia.raw", InFormat="raw")
```

This function can also convert from two-columns-per-SNP format, as e.g. used by Colony.

5.1.2.1 Family IDs By default, the 'Family ID' (1st) column in the PLINK file is ignored, and IDs are extracted from the second column only. If the family IDs are essential to distinguish between individuals, use `GenoConvert()` with the flag `UseFID = TRUE` which will combine individual IDs and family IDs as FID__IID. Ensure the IDs in the life history file are in the same format, for example by using `LHConvert()`. The FID and IID can be split again in the resulting pedigree using `PedStripFID()`.

5.1.3 Low call rate

Samples with a very low genotyping success rate (call rate) can sometimes wrongly be assigned as parents to unrelated individuals, as `sequoia()` does not (yet) deal perfectly with these cases. In addition, at least in my experience with SNP arrays, a low sample call rate is often indicative of poor sample quality or a poor genotyping run, and associated with a high sample error rate. Samples with a call rate below 5% are automatically excluded, but it is strongly advised to create a subset where all individuals are genotyped for at least 50% of SNPs, and preferably for at least 80%.

In addition, SNPs with a call rate below 10% are excluded, as these contribute almost no information. Again, a stricter threshold is advised of at least 50% to minimise the risk of spurious results.

Checks for individuals and SNPs with low call rate, and monomorphic SNPs, are done automatically when calling `sequoia()` and various other functions, and can be done separately by calling `CheckGeno()`.

5.1.4 Very large datasets

When the number of individuals is very large, it may be impossible to load the genotype data into R as it will exceed R's memory limit. A stand-alone version of the algorithm underlying this R package does not suffer from this limitation, and is available as Fortran source code from github, where you can also find its manual. The standalone is not faster than the R package, as the bulk of the computations are done in Fortran regardless.

5.2 Birth years & sex

The relative age of individuals can greatly improve pedigree reconstruction (see Birth years & Ageprior), and sex information is necessary to determine whether a candidate parent is the mother or father.

5.2.1 LifeHistData

The life history data (`LifeHistData`) should be a dataframe with three or five columns (column names are ignored, order is important!):

- **ID**
  It is probably safest to stick to R's 'syntactically valid names', defined as "consists of letters, numbers and the dot or underline characters and starts with a letter, or the dot not followed by a number".
- **Sex**
  1 = female, 2 = male, 3=unknown, 4=hermaphrodites. All other numbers, letters, or NA = unknown
- **BirthYear**
  Year of birth/hatching/germination/... In species with more than one generation per year, a finer time scale than year of birth ought to be used (in round numbers), ensuring that parents are always 'born' in a time unit prior to their first offspring (e.g. parent's BirthYear=2001 ($t = 1$) and offspring BirthYear=2005 ($t = 5$)). Negative numbers and NA's are interpreted as unknown.
- **BY.min** (optional)
  Earliest year in which individual may have been born, if exact year is unknown. Ignored when BirthYear is non-missing.
- **BY.max** (optional)
  Latest year in which individual may have been born

Ideally this basic life history information is provided for all genotyped individuals, but this is not strictly necessary. This dataframe may be in a different order than the genotype data, and may include many more individuals (which are ignored).

5.2.2 Unknown birth years

The year of birth may be unknown for some individuals, and for those `sequoia()` cannot determine whether they are the parent or the offspring if a genetic parent-offspring pair is found (unless a 'complementary' co-parent is identified). Especially in wild populations this information can be unknown for a substantial part of the sample, but often a minimum and/or maximum possible birth year (`BY.min` and `BY.max`) can be determined. For example, `BY.max` is at the latest the first year in which an individual was observed.

Even very wide ranges may help in pedigree reconstruction: for example, Alpha and Beta are genetically identified to be parent and offspring, Alpha was first seen in 2011 as an adult, and Beta was known to be born 2012 (Table 4). Then Alpha is certainly the parent of Beta. Before version 2.0 there was no method to inform `sequoia` that Alpha could not have been born after 2012, and thus could not be an offspring of Beta. It was (and still is) possible to 'guestimate' birth years, but this can be time consuming and is potentially prone to errors. For example, if Alpha also forms a genetic parent-offspring pair with Gamma, then setting Alpha's birth year to 2010 will result in it being assigned as Gamma's offspring, while it may just as well be Gamma's parent.

To minimise such pedigree errors where parent and offspring are flipped the wrong way around, and the

Table 4: Example LifeHistData with the optional columns for minimum + maximum birth year

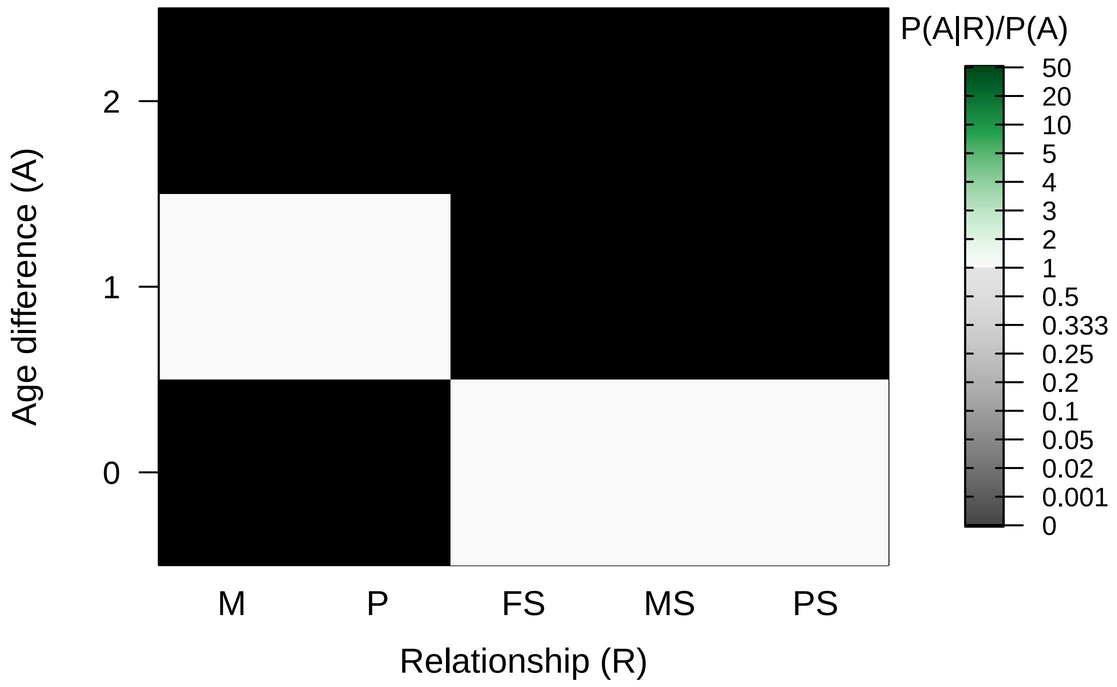
ID	Sex	BirthYear	BY.min	BY.max
Alpha	2	NA	NA	2010
Beta	2	2012	NA	NA
Gamma	1	NA	2008	2009

'secondary' errors derived from these cases, it is recommended to set the possible birth year range as wide as possible, at least during an initial (preliminary) round of parentage assignment. To see the pairs that are genetically parent and offspring, but for which it cannot be determined which of the two is the parent, use function `GetMaybeRel()`.

5.2.3 `args.AP` (Customising the ageprior)

When running `sequoia()`, you can pass arguments to `MakeAgePrior()` via `args.AP`. For example, when you are sure generations do not overlap, you can specify this via `MakeAgePrior()`'s argument `Discrete`:

```
SeqOUT <- sequoia(GenoM = SimGeno_example, LifeHistData = LH_HSg5, Module='par',
                  args.AP = list(Discrete = TRUE), quiet=TRUE)
PlotAgePrior(SeqOUT$AgePriors)
```



Or you can specify that the maximum age of (non-genotyped) parents exceeds the observed age range among SNP-genotyped parent-offspring pairs in `PedigreePar`:

```
SeqOUT <- sequoia(GenoM = Geno, LifeHistData = LH,
                  args.AP = list(MaxAgeParent = c(11, 9)),  # dams, sires
                  SeqList = ParOUT[c("Specs", "PedigreePar")])
```

Alternatively, if you have a field-pedigree or microsatellite-based pedigree that contains many more individuals than have been SNP-genotyped, an ageprior estimated from that old pedigree will be more informative than the one estimated from a limited number of SNP-genotyped parent-offspring pairs. This can be done as follows:

```
APfromOld <- MakeAgePrior(Pedigree = MyOldPedigree,
                          LifeHistData = LH,
                          Smooth = TRUE)
SeqOUT <- sequoia(GenoM = Geno,
                  LifeHistData = LH,
                  SeqList = list(AgePriors = APfromOld))
```

When argument `SeqList` contains an element `AgePriors`, it is used during both parentage assignment and sibship clustering. Thus, if you wish to use different agepriors during those different phases, you need to run them separately (see reuse).

For further information on the ageprior, please see the separate vignette on this topic. If you wish to manually edit the `AgePriors` matrix before using it as input in `sequoia()`, or have done so in the past, it is advised to look at the latest version of this document, as the implementation has changed in package version 2.0, and the documentation was clarified in version 2.1.

5.3 `Tfilter` & `Tassign`

These are threshold values for log10-likelihood ratios (LLRs).

`Tfilter` is the threshold LLR between a proposed relationship versus unrelated, to select candidate relatives. It is typically negative, and a more negative value may prevent filtering out of true relatives, but will increase computational time.

`Tassign` is the threshold used for acceptance of a proposed relationship, and is relative to next most likely relationship. It must be positive, with higher values resulting in more conservative assignments. Counter-intuitively a high `Tassign` can sometimes *increase* the number of wrong assignments, as a correct low-threshold assignment may prevent wrong assignments among close relatives (a.o. by revealing genotyping errors and changing the most-likely true genotype of an individual).

The default values of `Tfilter` and `Tassign` seem to work well in a wide range of datasets, but their interaction with genotyping error rate and other dataset characteristics has not been thoroughly explored.

5.4 `Complex`

The complexity of the mating system considered. One of:

- **`mono`** Only consider monogamous matings. This can be especially useful for small SNP panels that have insufficient power to distinguish reliably between full siblings and half-siblings, but may have ample power to distinguish between full siblings and third degree relatives (e.g. full cousins). Works well when full aunts/uncles and grandparents differ consistently more in age than full siblings, and do not have to be considered as alternatives to full sibling either.
- **`simp`** Consider polygamous matings, but ignore all inbred and double relationships (see below). Such relationships may still be assigned as side-effect. This again can be useful if the SNP panel has limited power, and the occurrence of such complex relationships is very rare.
- **`full`** The default, do consider all kinds of relationship combinations.

5.4.1 Unusual relationships

Pedigree inference is often applied in small, (semi-)closed populations, and is regularly done to check for inbreeding. In such cases, pairs of individuals may be related via more than one route. For example, maternal half-siblings may also be niece and aunt via the paternal side, and be mistaken for full-siblings. Or a pair may be both paternal half-siblings, and maternal full cousins. A range of such double relationships is considered explicitly (Table 5) to minimise such mistakes. If such a type is common in your population but not yet considered by `sequoia`, and seems to be causing problems, please send me an email as adding additional relationships is relatively straightforward.

Table 5: Double relationships between pairs of individuals. Abbreviations as before, and GGG=great-grandparent, F1C=full first cousins, H1C=half first cousins (parents are HS).

	PO	FS	HS	GP	FA	HA	GGG	F1C	H1C	U
PO	–	–	Y	Y		Y				Y
FS	–	–	–	–	–	Y		–	Y	Y
HS	Y	–	(FS)	Y	Y[2]	Y	Y[2]			Y
GP	Y	–	Y	[1]						Y
FA		–				Y				Y

	PO	FS	HS	GP	FA	HA	GGG	F1C	H1C	U
HA		Y	Y [2]							Y
GGG		–					[3]			Y
F1C										Y

–: impossible
Y: explicitly considered
empty: not (yet) explicitly considered (but may be inferred as 'side effect')
[1]: Can not be considered explicitly, as likelihood identical to PO
[2]: Including the special case were one is inbred
[3]: Can not be considered explicitly, as likelihood identical to GP

5.5 `Herm`

Hermaphrodites, one of:

- `no` Dioecious (separate sexes) species
- `A` Heed dam versus sire role of individuals, and assign a parent only if it is clear that it was the female or male parent of an individual. Requires a pedigree prior (e.g. the plant from which the seed was collected), or age difference between maternal versus paternal role (see Hermaphrodites).
- `B` No distinction is made between dam versus sire role; any parent is assigned in the first available 'slot', and no conclusions can be drawn from whether individuals are assigned as maternal, paternal or 'cross' half-siblings.

If any individuals in the life history data have sex=4, `Herm='no'` will automatically be changed into `Herm='A'`. Hermaphrodites may be mixed with known-sex (or known sex role) individuals.

5.6 `UseAge`

The strength of reliance on birth year information during full pedigree reconstruction.

In addition to providing no birth year information at all, there are three possible levels of reliance on birth years and ageprior during full pedigree reconstruction:

- `no` Only use the ageprior matrix to determine which age difference – relationship combinations are possible and which are not, i.e. simplifying the ageprior to a 0/1 (FALSE/TRUE) matrix.
- `yes` (default) Use age information in a restrictive way: only accept a proposed assignment if both the purely genetic LLR (LLR_{SNP}) and the sum of LLR_{SNP} and log10 of the ageprior's probability ratios (LLR_{age}) pass T_{assign} (Note that the ageprior matrix is not on a log scale, and that $log(0) = -Inf$, and $log(1) = 0$). Thus, relationships that are genetically likely but unlikely based on the age difference, will not be assigned.
- `extra` Use age information in a permissive way: accept a proposed assignment if $LLR_{SNP} + LLR_{age}$ passes T_{assign}, even if LLR_{SNP} on its own does not. Thus, a pair for which genetically the relationship is ambivalent (e.g. 'either one of HS, GP, FA'), but whose age difference makes one of these relationships (much) more likely, are now also assigned (see example). The risk of a cascading avalanche of assignment errors is reduced by first running the algorithm as for `UseAge = 'yes'`, and only once the total likelihood has plateaued switching to `UseAge = 'extra'` (and continuing until the total likelihood plateaus again).

Below an example of how to calculate LLR_{SNP} and LLR_{age}, for a grandparent – grand-offspring pair from the griffin example pedigree, with an age difference of 4 years. This pair would not be assigned with `UseAge = 'yes'`, as purely genetically the likelihoods for the pair being HS, GP or FA are identical (unless both individuals already have at least 1 parent assigned), but would get assigned with `UseAge = 'extra'`, as in this fictional population 4-year-older paternal grandmothers are about 10x as common as 4-year-older paternal aunts (difference of 1 unit on log10-scale).

Table 6: Parameter 'UseAge' (see text for definitions)

	no	yes	extra
$LR_{age} > 0$	Y	Y	Y
$LLR_{SNP} > T_{assign}$	Y	Y	
$LLR_{SNP} + LLR_{age} > T_{assign}$		Y	Y

Table 7: LLR example for a grandparent - grand-offspring pair

	PO	FS	HS	GP	FA	HA	U
SNP	-408.9	-344	-333.8	-333.8	-333.8	-333.4	-336.3
Age	-0.4	-3	-3.0	0.3	-0.3	-0.3	0.0
SNP + Age	-409.2	-347	-336.8	-333.5	-334.1	-333.7	-336.3

Such assignments which rely strongly on the age difference of a pair, in combination with the estimated age distribution of relative pairs, are often not desirable, and are therefore not made by default (`UseAge = 'yes'`).

```
data(Ped_griffin, SeqOUT_griffin, package="sequoia")
GenoX <- SimGeno(Ped_griffin, nSnp = 400, ParMis=0)
LLR_SNP <- CalcPairLL(Pairs = data.frame(ID1="i122_2007_M", ID2="i042_2003_F"),
                      GenoM = GenoX, Plot=FALSE)
```

```
## i Not conditioning on any pedigree
```

```
LLR_Age <- GetLLRAge(SeqOUT_griffin$AgePriorExtra, agedif=4, patmat=2)
knitr::kable(rbind(SNP = LLR_SNP[,colnames(LLR_Age)],
                   Age = LLR_Age,
                   "SNP + Age" = LLR_SNP[,colnames(LLR_Age)] +
                     LLR_Age),
             digits=1, booktabs=TRUE,
             caption = "LLR example for a grandparent - grand-offspring pair")
```

5.7 SeqList

Parameter settings and input data from one `sequoia()` run can be re-used in a subsequent `sequoia()` run, and as input by various other functions (Table 3). This makes it easier to have consistent parameter values and input data across different pedigree inference runs and different functions.

The parameter values used as arguments when calling `sequoia()` are returned in the list element `Specs`. This 1-row dataframe may be edited (with caution!) before re-use. Other elements of the `sequoia()` output list (`SeqList`) that are used as input by one or more functions are `LifeHist`, `AgePriors`, and `PedigreePar`/`Pedigree`. Details on which elements are used is provided in the help file of each function. `SeqList[["Specs"]]` nearly always take precedent over similarly-named input parameters, with exception of e.g. `sequoia()`'s `Module` argument. The other `SeqList` elements generally also take precedent, but not always with a warning. If you wish to use a mixture of `SeqList` elements and 'new' data, it is safest to provide only the minimal subset of `SeqList`, and specify the other data as separate input (or as 'fake' `SeqList` elements).

For example, if you wish to change the maximum sibship size, but do not want to re-run parentage assignment:

```
ParOUT$Specs$MaxSibshipSize <- 150
SeqOUTX <- sequoia(GenoM = Geno,
                  SeqList = ParOUT[c("Specs", "PedigreePar", "LifeHist")],
                  Module = "ped")
```

6 Running sequoia()

First all input is checked, and by default (see `Module`) this is followed by assignment of genotyped parents to genotyped offspring, and then by full pedigree reconstruction.

6.1 console output

You can toggle the amount of runtime information provided with parameter `quiet`, with options `TRUE`, `FALSE` (default), and `verbose` (extra details). With `verbose`, you see output similar to this:

```
SeqOUT <- sequoia(GenoM = Geno_griffin,  LifeHistData = LH_griffin, Plot=FALSE, quiet='verbose')
```

```
## i Checking input data ...
## v Genotype matrix looks OK! There are  142  individuals and  400  SNPs.
##
## -- Among genotyped individuals: ___
## i There are 66 females, 76 males, 0 of unknown sex, and 0 hermaphrodites.
## i Exact birth years are from 2001 to 2010
## ___
## i Calling `MakeAgePrior()` ...
## i Ageprior: Flat 0/1, overlapping generations, MaxAgeParent = 10,10
##
## ~~~ Duplicate check ~~~
## v No potential duplicates found
##
## ~~~ Parentage assignment ~~~
##     Time |  R |       Step |   progress |  dams | sires |   Total LL
## -------- | -- | ---------- | ---------- | ----- | ----- | ----------
## 14:45:50 |  0 | count OH   | .......... |
## 14:45:50 |  0 | initial    |            |     0 |     0 |   -23820.2
## 14:45:50 |  1 | parents    | .......... |    65 |    79 |   -19499.2
## 14:45:51 |  2 | parents    | .......... |    65 |    79 |   -19499.2
## 14:45:52 | 99 | est byears |            |
## 14:45:52 | 99 | calc LLR   | .......... |
## v assigned 65 dams and 79 sires to 142 individuals
## i Ageprior: Pedigree-based, overlapping generations, smoothed, MaxAgeParent = 5,5
##
## ~~~ Full pedigree reconstruction ~~~
##     Time |  R |       Step |   progress |  dams | sires |   Total LL
## -------- | -- | ---------- | ---------- | ----- | ----- | ----------
## 14:45:52 |  0 | count OH   | .......... |
## 14:45:52 |  0 | initial    |            |    65 |    79 |   -19499.2
## 14:45:52 |  0 | ped check  | .......... |    65 |    79 |   -19499.2
## 14:45:52 |  1 | find pairs | .......... |    65 |    79 |   -19499.2
## 14:45:53 |  1 | clustering | .......... |    94 |    88 |   -18761.7
## 14:45:53 |  1 | merging    | .......... |    94 |    88 |   -18761.7
## 14:45:53 |  1 | P of sibs  | .......... |    94 |    88 |   -18761.7
## 14:45:53 |  1 | find/check | .......... |    94 |    88 |   -18766.8
## 14:45:58 |  2 | find pairs | .......... |    94 |    88 |   -18766.8
## 14:45:58 |  2 | clustering | .......... |    94 |    88 |   -18766.8
## 14:45:58 |  2 | merging    | .......... |    94 |    88 |   -18766.8
## 14:45:58 |  2 | P of sibs  | .......... |    94 |    88 |   -18766.8
## 14:45:58 |  2 | GP Hsibs   | .......... |    94 |    88 |   -18654.9
## 14:45:59 |  2 | find/check | .......... |    94 |    88 |   -18644.4
(...)
```

```
## 14:46:08 |  5 | P of sibs  | .......... |   107 |   101 |   -18164.4
## 14:46:08 |  5 | GP Hsibs   | .......... |   107 |   101 |   -18164.4
## 14:46:08 |  5 | GP Fsibs   | .......... |   107 |   101 |   -18164.4
## 14:46:08 |  5 | find/check | .......... |   106 |   101 |   -18181.4
## 14:46:09 | 99 | est byears |            |
## 14:46:09 | 99 | calc LLR   | .......... |
## v assigned 123 dams and 122 sires to 142 + 28 individuals (real + dummy)
## i You can use `SummarySeq()` for pedigree details, and `EstConf()` for confidence estimates
## i Possibly not all relatives were assigned, consider running `GetMaybeRel()`
## conditional on this pedigree to check
```

where `R` is the round number - this is not like MCMC iterations, but rather each round builds upon the previous.

6.1.1 step

`step` describes the various elements within each full pedigree reconstruction round, as well as the preparation & closing steps:

- **count OH** (R=0): Count number of opposing homozygous loci between all pairs of individuals
- **initial** (R=0): Initial status; for `par` this assumes all individuals are unrelated, for `ped` this is the parents assigned during the `par` module.
- **parents** (`Module='par'`): Assignment of genotyped parents to genotyped offspring
- **est byears** (R=99): Estimate birth year ranges for individuals with unknown birth year in the input `LifeHistData`, as well as for dummy individuals (when `Module='ped'`).
- **calc LLR** (R=99): Calculate for each assigned parent & parent-pair the likelihood ratio to be parent-offspring with the focal individual vs otherwise related, conditional on the rest of the pedigree.
- **ped check** (R=0): Check input pedigree: are all assignments genetically likely (LLR > `T_assign`) and are age differences compliant with the provided ageprior matrix? Unlikely and impossible assignments are dropped.

For `Module='ped'` (some are skipped during the 1st and 2nd round):

- **find pairs**: find pairs of likely full and half siblings
- **clustering**: cluster these pairs into full-sibling and maternal and paternal half-sibling clusters
- **GP pairs**: assign grandparent-grandoffspring pairs (both individuals genotyped)
- **merging**: check if any maternal (paternal) half-sibships should be merged
- **P of sibs**: check if the dummy parent of any half-sibship can be replaced by a real, genotyped individual (e.g. one with unknown sex or birth year, or poorly genotyped)
- **GP Hsibs**: assign grandparents (real & dummy) to half-sibships
- **GP Fsibs**: assign grandparents (real & dummy) to full-sibships
- **find/check**: for each individual, see if a (dummy) parent can now be assigned. If a parent was assigned this round, double check it.

6.1.2 dams, sires

The number of dams & sires assigned to genotyped individuals at the end of the step. For brevity, the number of parents assigned to dummy individuals (i.e. grandparents of sibships) is not included. Note that the final message (‘assigned 123 dams and 122 sires to 142 + 28 individuals (real + dummy)’) *does* include parents assigned to dummy individuals.

6.1.3 Total LL

The total likelihood (last column) should go steadily from the initial value (how well are the genetic data explained as random, independent draws from a population to HWE) towards zero (all genetic data perfectly explained by the reconstructed pedigree). It is normal for it to not change during some steps (no change made to the pedigree), and it can decrease a bit during the 'find/check' step (not confident in an earlier made assignment after all, conditional on how the rest of the pedigree looks now).

However, if you see the likelihood zigzag up & down again and the program keeps on running, something is wrong. It can be that there are several mutually exclusive pedigree configurations with nearly equal likelihood, and the algorithm gets stuck cycling between them. This can especially happen with poor quality/quantity of genetic data. Sometimes it can get unstuck by changing input parameter values (`Err`, `T_assign`) or by providing additional information (YearLast in LifeHistData, more informative ageprior). Sometimes it may simply mean `sequoia` is not the right tool for your dataset, and you require an MCMC based program (which, I think, are not yet available for complex multi-generational pedigrees. . .).

6.2 `Module`

Most datasets initially contain imperfections. Pedigree reconstruction can be a useful tool in quality control, to find outliers with many Mendelian errors among e.g. known mother–offspring pairs (sample mislabelling?) or among SNPs (poor quality SNPs to be excluded).

To speed up iterative quality control, it is possible to run `sequoia()` only up to a certain point, use the output thus far to, as necessary, exclude SNPs, exclude/relabel samples, or adjust input parameters, and re-run again from the start or resume from the last point.

There are three possible intermediate stopping points, plus the final end point, each named after the last '`Module`' [3] that is run:

1. **`'pre'`**: Input check
   check that the genotype data, life history data and input parameters are in a valid format, create `'Specs'` element of output list
2. **`'dup'`**: check for duplicates
   Check for (nearly) identical genotypes, and for duplicated IDs in the genotype and life history data
3. **`'par'`**: parentage assignment
   Assign genotyped parents to genotyped individuals. Includes call to `MakeAgePrior()` to estimate `AgePriors` based on the just-assigned parents.
4. **`'ped'`**: full pedigree reconstruction
   Cluster half- and full-siblings and assign each cluster a dummy-parent; assign grandparents to sibships and singletons.

6.3 Input check

SNP datasets are typically too large to easily spot any problems by simply looking at the data in a spreadsheet. There are many tools available to deal with genotype data and do proper quality control; the function `CheckGeno()` merely checks that the data are in the correct format for `sequoia`, and that there are no SNPs or individuals with excessively many missing values.

The function `SnpStats()` can be used to count the number of Mendelian errors per SNP. It is not automatically called by `sequoia()`, but excluding SNPs with exceptionally high genotyping error rates is recommended to improve accuracy of the inferred pedigree.

6.4 Check for duplicates

The data may contain positive controls, as well as other intentional and unintentional duplicated samples, with or without life-history information. `sequoia()` searches the data for (near) identical genotypes, allowing for `MaxMismatchDUP` mismatches between the genotypes, which may or may not have the same individual ID. The threshold value depends on the presumed genotyping error rate, allele frequencies and number of SNPs, and is calculated by `CalcMaxMismatch()`.

[3] `Module` is the successor of the rather confusing `MaxSibIter` that was used up to version 2.0).

Not all pairs flagged as potential duplicate genotypes are necessarily actual duplicates: inbred individuals may be nearly indistinguishable from their parent(s), especially when the number of SNPs is limited.

This check will also return a vector of individuals included in the genotype data, but not in the life history data (`NoLH`). This is merely a service to the user; individuals without life history information can often be successfully included in the pedigree.

6.5 Parentage assignment

The number of pairs to be checked if they are parent and offspring is very large for even moderate numbers of individuals, e.g. 5,000 pairs for 100 individuals, and 2 million for 2,000 individuals. To speed up computation, three 'sieves' are applied sequentially to find candidate parent-offspring pairs, with decreasing 'mesh size'

- The number of SNPs at which the pair are opposing homozygotes must be less than `MaxMismatchOH`;
- The log10-likelihood ratio (LLR) between being parent and offspring versus unrelated, not conditioning on any already assigned parents, must be equal to or greater than `Tfilter`;
- The LLR between the pair being parent and offspring versus being otherwise related, calculated conditional on all already assigned relatives, must be equal to or greater than `Tassign`. This step filters out siblings, grandparents and aunts/uncles.

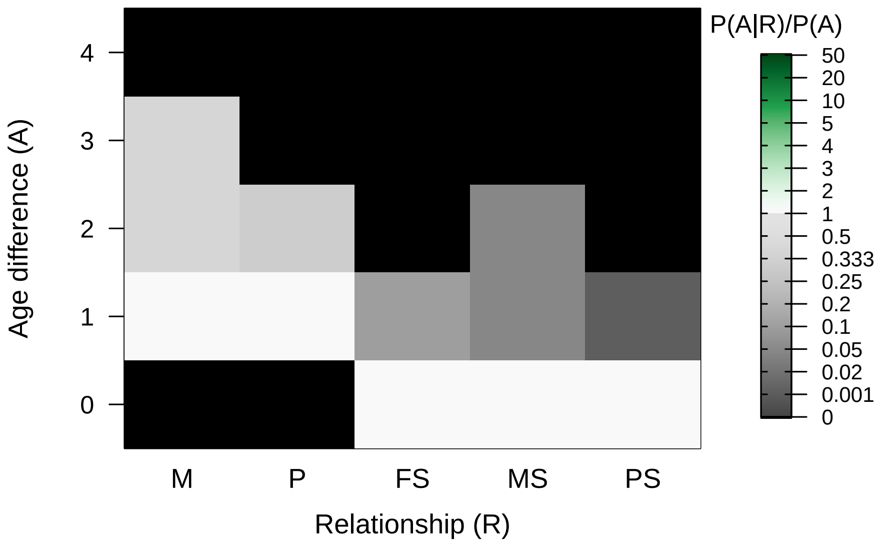
The older of the pair is assigned as parent of the younger. If it is unclear which is the older, or if it is unclear whether the parent is the mother or the father, no assignment is made, but these pairs can be found with `GetMaybeRel()` (see section Likely relatives). If there are multiple candidate parents of one or both (or unknown) sex(es), the parent pair or single parent resulting in the highest likelihood is assigned.

The heuristic sequential filtering approach makes parentage assignment quick, and usually takes only a few minutes, especially when setting `CalcLLR=FALSE` (do not re-calculate LLR parent-offspring vs next-most-likely relationship for all assigned parents, based on the final pedigree).

6.6 AgePrior

The assigned parents are used to update the ageprior, before returning the results to the user of continuing with full pedigree reconstruction. Any arguments to `MakeAgePrior()` can be passed via `args.AP`. For example, you can specify the maximum age of dams and sires:

```
SeqOUT.B <- sequoia(GenoM = SimGeno_example, Err = 0.005,
                    LifeHistData = LH_HSg5, Module="par", Plot=FALSE, quiet=TRUE,
                    args.AP = list(MaxAgeParent = c(3,2), Smooth=FALSE))
PlotAgePrior(SeqOUT.B$AgePriors)
```



This way, even if sampled parents are all age 1, siblings sharing an unsampled parent may still have an age difference of 1 or 2 years, and grandparent – grand-offspring pairs may have an age difference of 2-6 years.

6.7 Full reconstruction

During full pedigree reconstruction, assignment of all first and second degree links between all individuals is attempted, using the following steps within each iteration

- Find pairs of likely full- and half-siblings, using filtering steps with decreasing 'mesh size' similar to parentage assignment
- Cluster potential sibling pairs into sibships
- Find and assign grandparent – grand-offspring pairs (iteration 3+)
- Merge existing sibships
- Replace dummy parents by genotyped individuals
- For individuals without parent(s), find candidate real and dummy parents, and assign parent(-pairs)
- For sibships without grandparent(s), find candidate real and dummy grandparents, and assign grandparent(-pairs) (iteration 2+)

The order of these steps, and the skipping of some steps in the first iteration(s), has been established by trial & error to minimise false positive assignments while maximising correct assignments across a wide range of datasets. When running `sequoia()`, you can keep track of the progress through the various steps by setting `quiet = 'verbose'`.

Full pedigree reconstruction may take from a few seconds to several hours, depending on the number of individuals without an already assigned parent, the proportion of individuals with unknown sex or birth year, the number of sibships that is being clustered and their degree of interconnection, and the number of SNPs and their error rate. Generally computation is faster if the data is more complete and more accurate.

6.7.1 Dummy Individuals

The 'dummy' parent assigned to each cluster of half-siblings is denoted by a 4-digit number with prefix `F` for females (`F0001`, `F0002`, . . .) and `M` for males (`M0001`, `M0002`, . . .). In the output, dummy individuals are appended at the bottom of the pedigree with their assigned parents, i.e. the sibship's grandparents. In addition, all information for each dummy individual is given in dataframe `DummyIDs`, including its parents (again), its offspring, and the estimated birth year, as a point estimate (`BY.est`) and lower and upper bound of the 95% probability interval. This information is intended to make it easier to match dummy IDs to real IDs of observed but non-genotyped individuals (see also Compare pedigrees).

6.7.2 Total likelihood

The total likelihood provides a measure of how well the observed genotype data is explained by the currently inferred pedigree, the allele frequencies of the SNPs, the presumed genotyping error rate, and the assumption that founder genotypes were drawn from a genepool in Hardy-Weinberg equilibrium. It is always a negative number, and closer to zero indicates a better fit. When an asymptote is reached, typically in 5–10 iterations, either the algorithm is concluded (the default) or dependency on the age prior is increased (if `UseAge = 'extra'`) and the algorithm continues until a new asymptote is reached.

6.8 Save output

There are various ways in which the output can be stored. For a single R object you can use `saveRDS` & `readRDS`:

```
saveRDS(SeqList, 'Sequoia_output_date.RDS')
# and later:
SeqList_B <- readRDS('Sequoia_output_date.RDS')
```

For multiple objects you can use `save` & `load`:

```
save(SeqList, LHdata, Geno, file="Sequoia_output_date.RData")
# and later:
load("Sequoia_output_date.RData")
# 'SeqList', 'LHdata' & Geno will appear in R environment
```

The advantage is that all data is stored and can easily be manipulated when recalled. The disadvantage is that the file is not human-readable, and (to my knowledge) can only be opened by R.

Alternatively, the various dataframes and list elements can each be written to a text file in a designated folder. This can be done using `write.table` or `write.csv`, or using the wrapper `writeSeq()`:

```
writeSeq(SeqList, GenoM = Geno, folder=paste("Sequoia_OUT", Sys.Date()))
```

The same function can also write the dataframes and list elements to an excel file (.xls or .xlsx), each to a separate sheet, using package `openxlsx`:

```
writeSeq(SeqList, OutFormat="xls", file="Sequoia_OUT.xlsx")
```

Note that ‘GenoM’ is ignored, as a very large genotype matrix may result in a file that is too large for excel to open. If you have a genotype matrix of modest size, you can add it to the same excel file:

```
library(openxlsx)
wb <- loadWorkbook("MyFile.xlsx")
addWorksheet(wb, sheetName = "Genotypes")
writeData(wb, sheet = "Genotypes", GenoM, rowNames=TRUE)
saveWorkbook(wb, "MyFile.xlsx", overwrite=TRUE, returnValue=TRUE)
```

7 Output check

7.1 Pedigree stats & plots

Various summary statistics can be calculated and displayed using `SummarySeq()`, such as the number and kind of assigned parents (Figure 6), family sizes (Figure 7), and the distributions of Mendelian errors. Its output also includes a table analogous to `pedigreeStats` by R package `pedantics` (which was archived on CRAN in December 2019).

There is a range of software available to plot pedigrees in various styles, such as R package `kinship2` or the program PedigreeViewer. Be aware that many use a different column order (id - sire - dam, rather than id - dam - sire), use 0 to denote missing parents rather than NA, and/or do not allow individuals to have a single parent (they must have 0 or 2 parents). The last two issues can be resolved with `Sequoia`’s function `PedPolish()`.

7.1.1 Negative LLR’s

The parent LLR’s are calculated based on genetic data only, without any consideration of age. Consequently, they are often close to zero for dummy – dummy parent-offspring pairs, as genetically the parent may just as well be the offspring. Such pairs will not have been assigned unless there is a compatible co-parent (`LLRpair` should always be positive) or if there is overwhelming age-based evidence to decide which one is the parent.

When a parental LLR is negative without an accompanying co-parent, or when `LLRpair` is negative, this is reason to cautious about the assignment and for example compare it to a previous pedigree or inspect the genomic relatedness. In contrast to MCMC type approaches, `sequoia`’s algorithm has limited scope to undo previously made assignments, and may get stuck at suboptimal solutions.

Extremely negative LLRs (< -100) are typically caused by one or more of the alternative relationships considered, now conditional on the rest of the pedigree, being very convoluted and neither being caught as ‘highly improbable and not implemented’ nor being implemented properly (see Table weird relationships for

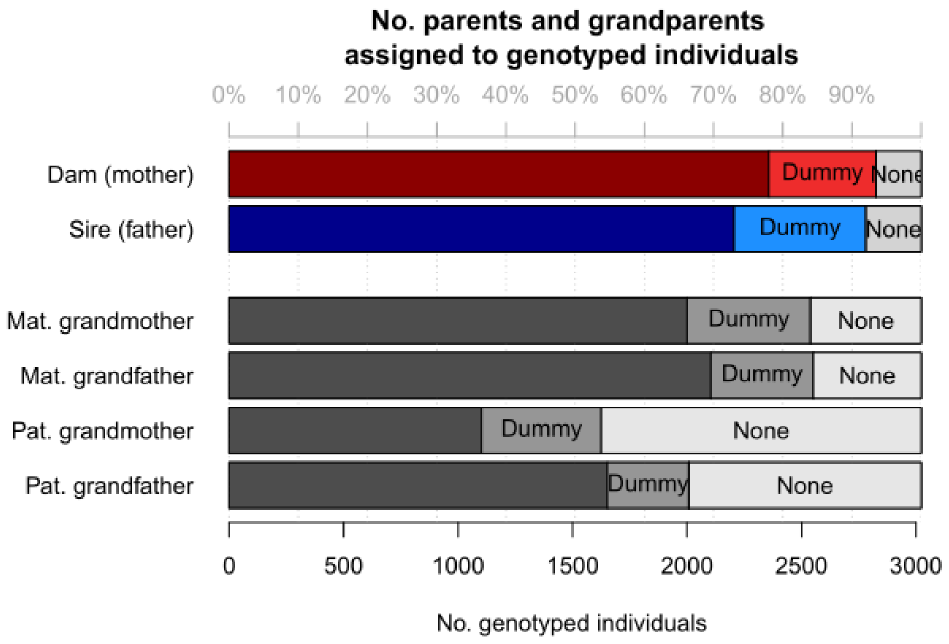


Figure 6: Number of parents assigned to genotyped individuals, split by category

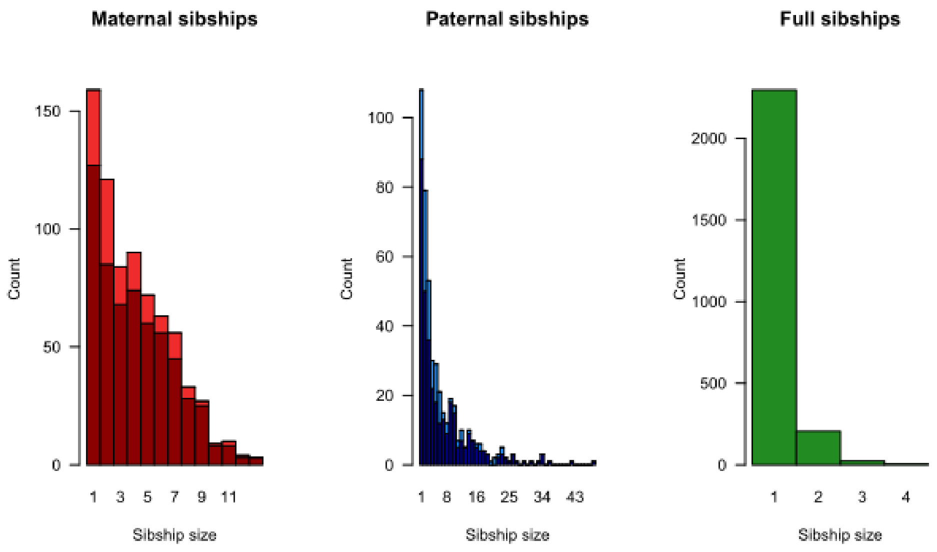


Figure 7: Family sizes, split by genotyped (dark) or dummy (light) parent

double relationships that are implemented). In the unlikely event that the parent-offspring relationship itself is not implemented properly you will see an error value of 444 or 777.

7.1.2 Dyad plot

Per popular request, from version 2.1 sequoia includes `PlotRelPairs()` to produce pairwise plots, similar to Colony's plot of half- and full-sibling dyads. It plots the output from `GetRelM()`, where you can specify if you want to trace each individual's pedigree 1 generation back (as in Figure 8) or 2 (include grandparents & aunts/uncles), and whether or not you want to differentiate between paternal and maternal relatives. It includes some basic functionality to subset individuals.

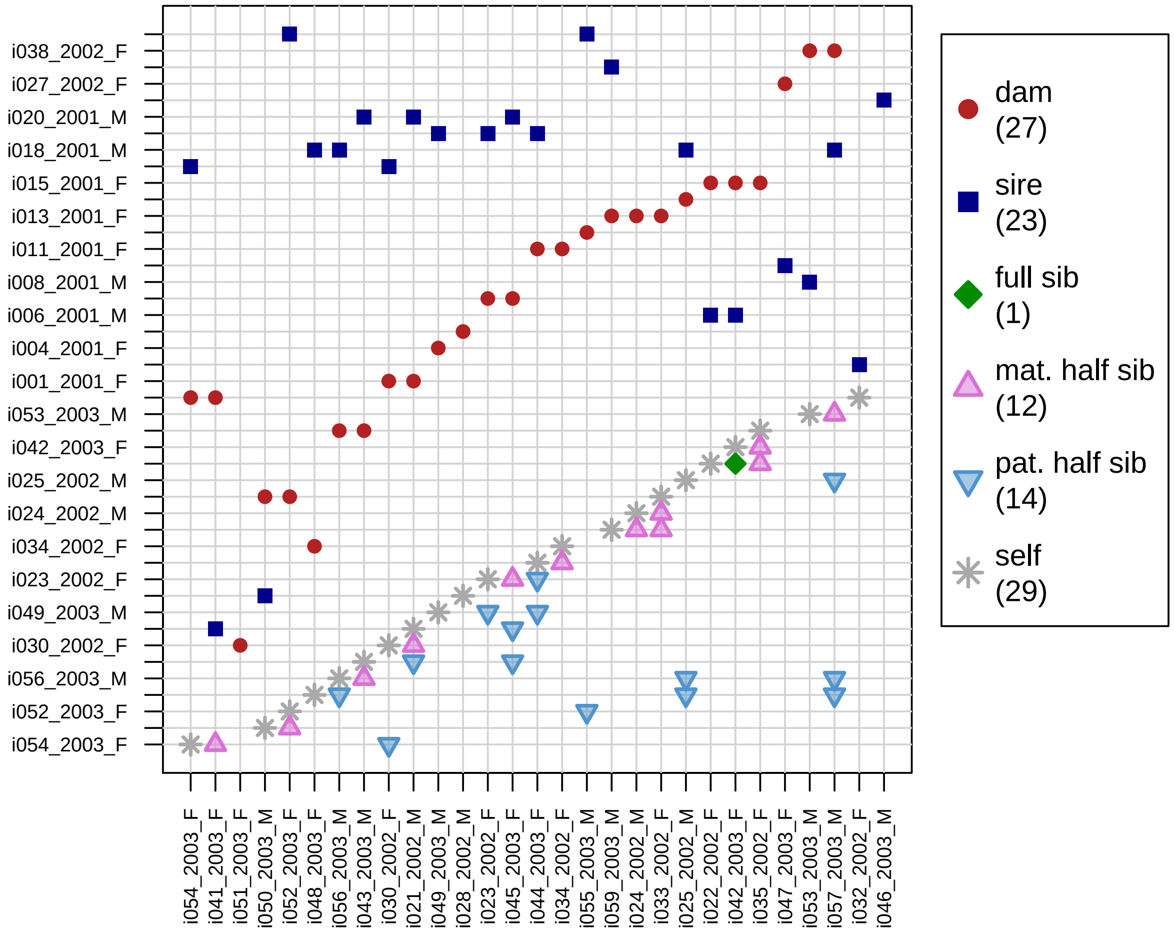


Figure 8: Example plot from PlotRelPairs()

7.2 Likely relatives

During parentage assignment, occasionally pairs are encountered which genetically are more likely to be parent-offspring than unrelated, but which can not be assigned because the sex or age difference is unknown or incompatible, or because the LLR is just below the assignment threshold (e.g. about equally likely PO and FS). Similarly, during pedigree reconstruction not all pairs of almost-certainly-relatives can be assigned, e.g. because half-siblings, full avuncular and grandparent–grand-offspring cannot always be distinguished. Distinguishing different kinds of second degree relatives relies on either both individuals already having at least one parent assigned, or two out of three being impossible due to the age difference of the pair. When neither is the case, the output indicates '2nd' as most likely relationship, and the LLR is between being 2nd degree relatives versus the most likely of PO (parent-offspring), FS (full siblings), HA (3rd degree relative) or U (unrelated).

Table 8: Example comparison between Pedigree1 (left) and Pedigree2 (right)

id	dam	sire	id	dam	sire
Alpha			Alpha		
Beta			-		
Abigail	Alpha	Zorro	Abigail	Alpha	Zorro
Aster	Alpha	Yann	Aster	Alpha	M0004
Blossom	Beta		Blossom	F0007	
Bob	Beta	Zorro	Bob	F0007	Zorro
-			F0007	Alpha	

These pairs can be identified by function `GetMaybeRel()` with as argument the genotype data and optionally any pedigree or a sequoia output list. It will also identify parent-parent-offspring trios which could not be assigned because the sex for both parents is unknown.

7.3 Compare pedigrees

Often a (part) pedigree is already available to which one wants to compare the newly inferred pedigree, for example to field-observed mothers or a microsatellite-based pedigree. `PedCompare()` compares for each individual the assigned parents between the two pedigrees (match/mismatch), while `ComparePairs()` compares for each pair their relationship between the two pedigrees (e.g. full sibs vs half sibs). Both pedigrees may have genotyped as well as non-genotyped individuals.

Most of the terminology in this vignette and the helpfiles is based on `Ped1` being the 'true' pedigree (the one from which data is simulated) and `Ped2` is the inferred pedigree (from the simulated data), but both functions work for any two pedigrees. From version 2.0 onwards you can indicate in `PedCompare()` that both pedigrees may be imperfect (`Symmetrical = TRUE`).

7.3.1 Dummy matching

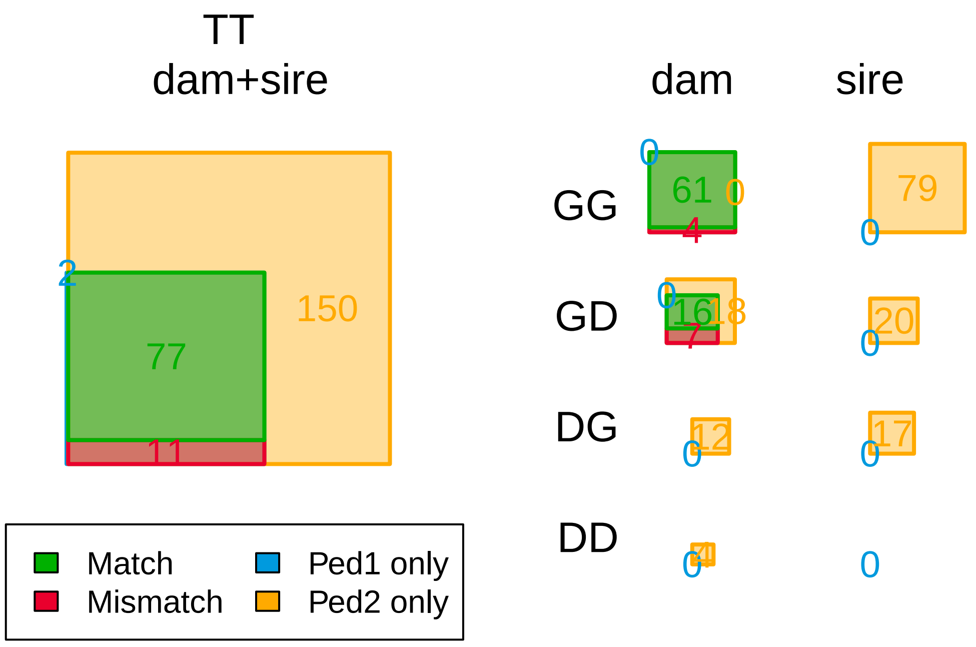
Comparing two pedigrees becomes complicated when dummy IDs are involved – When are non-identical parental IDs between the two pedigrees actually referring to the same individual? For example, in Table 8 the mother of Blossom and Bob is Beta in Pedigree1, and dummy female F0007 in Pedigree 2. It seems fair to assume that F0007 thus refers to the non-genotyped female Beta. And since Alpha was assigned as maternal grandmother to Blossom and Bob, we can also conclude that Alpha must be the mother of Beta!

Matching and replacing dummy IDs this way extends the number of individuals with both pedigree and phenotypic data, which will among others increase power of any downstream quantitative genetic analyses.

More specifically, `PedCompare()` considers it a 'match' if the inferred sibship (in Pedigree2) which contains the most offspring of a non-genotyped parent (in Pedigree1), consists for more than half of this individual's offspring. If for example a cluster of 5 siblings in Pedigree1 is split into two clusters in Pedigree2, the larger sibship (of say 3 individuals) is considered a match, and the smaller a mismatch (thus 2 mismatches) — even though it could be argued the inferred Pedigree2 does not contain any incorrect links. For this same example, `ComparePairs()` would tell that the $(5 * 4)/2 = 10$ halfsib pairs in Pedigree1 are $3 + 1$ halfsib pairs in Pedigree2, and 6 unrelated pairs. `ComparePairs()` does not attempt any matching of dummy individuals.

7.3.2 Example

```r
PCG  <- PedCompare(Ped1 = cbind(FieldMums_griffin,
                                sire = NA),
                   Ped2 = SeqOUT_griffin$Pedigree)
```



```
# pedigrees side-by-side (subset of columns because no field-observed sires here)
PCG$MergedPed[127:133, c("id", "dam.1", "dam.2", "dam.r", "id.dam.cat", "dam.class")]
```

```
##                id       dam.1       dam.2    dam.r id.dam.cat dam.class
## 127 i148_2008_F GreenYellow       F0005 nomatch         GD  Mismatch
## 128 i149_2008_F i126_2007_F i126_2007_F     <NA>         GG     Match
## 129 i151_2008_F i108_2006_F i108_2006_F     <NA>         GG     Match
## 130 i153_2008_F        <NA>        <NA>     <NA>         GX         _
## 131 i154_2008_F i137_2007_F i137_2007_F     <NA>         GG     Match
## 132 i157_2008_F        <NA>        <NA>     <NA>         GX         _
## 133 i158_2008_M     BlueRed       F0001  BlueRed         GD     Match
```

```
#  Non-genotyped field mums have a two-colour code (e.g. 'BlueRed')

PCG$MergedPed[c(137,138, 6, 128,129,7), c("id", "id.r", "dam.1", "dam.2", "dam.r")]
```

```
##                id        id.r       dam.1       dam.2    dam.r
## 137 i165_2009_F i165_2009_F    PinkBlue       F0002 PinkBlue
## 138 i166_2009_F i166_2009_F    PinkBlue       F0002 PinkBlue
## 6           F0006   RedOrange        <NA>        <NA>     <NA>
## 128 i149_2008_F i149_2008_F i126_2007_F i126_2007_F     <NA>
## 129 i151_2008_F i151_2008_F i108_2006_F i108_2006_F     <NA>
## 7           F0007  YellowPink        <NA>        <NA>     <NA>
```

```
# column 'id': ids common to both pedigrees, plus those only in Pedigree2
# column 'id.r': 'consensus' ids, plus those only occurring in Pedigree1

# dummy individuals from Ped2 with their best-matching non-genotyped individual in Ped1
head(PCG$DummyMatch[, -c(3:5)])
```

```
##    id.2        id.1 off.Match off.Mismatch off.P1only off.P2only
## 1 F0001     BlueRed         4            4          0          3
## 2 F0002    PinkBlue         3            0          0          0
## 3 F0003 GreenYellow         3            2          0          0
## 4 F0004     nomatch         0            0          0          2
## 5 F0005     nomatch         0            2           0           0
## 6 F0006   RedOrange         3            0           0           0
```

```
# 'nomatch' in the `id.1` column: none of the siblings in Ped2 had a field-observed
# mother, or there is a mismatch.

# Total number of matches & mismatches:
PCG$Counts
```

```
## , , parent = dam
##
##     class
## cat  Total Match Mismatch P1only P2only
##   GG    65    61        4      0      0
##   GD    41    16        7      0     18
##   GT   107    77       11      1     18
##   DG    12     0        0      0     12
##   DD     4     0        0      0      4
##   DT    16     0        0      0     16
##   TT   124    77       11      2     34
##
## , , parent = sire
##
##     class
## cat  Total Match Mismatch P1only P2only
##   GG    79     0        0      0     79
##   GD    20     0        0      0     20
##   GT    99     0        0      0     99
##   DG    17     0        0      0     17
##   DD     0     0        0      0      0
##   DT    17     0        0      0     17
##   TT   116     0        0      0    116
```

```
# dim1: category: genotyped/dummy/total
# dim2: classification: total (could-have-had-parent)/ match/ mismatch/
#       P1only (no parent in Ped2)/ P2only
# dim3: dam/sire
```

See here for a detailed walk-through of this example, and suggestions on how to trace the underlying cause of discrepancies between the newly-inferred genetic pedigree and an existing field pedigree.

In short, the main aim of checking the discrepancies is to figure out the underlying cause of each case:

- pedigree reconstruction error,
- wet lab error (sample swap/mislabeling), or
- field 'error' (e.g. mistaken/unknown identity),

so that the records and/or pedigree can be corrected. `PedCompare()` only points you to where the discrepancies are; field records, lab records and common sense are needed to determine the most likely cause (or determine that the cause is unknowable).

Fairly common (but still rare) pedigree reconstruction errors are

- one sibship is erroneously split into two,
- two sibships are erroneously merged into one,
- an aunt/uncle is assigned as sibship-grandparent,
- the parent-offspring relationship between to dummy individuals, or between two genotyped individuals with unknown birth years, is flipped the wrong way around.

```
# ~~ Mismatches ~~
PCG$MergedPed[which(PCG$MergedPed$dam.class == "Mismatch"),
              c("id", "dam.1", "dam.2", "id.dam.cat")]
```

```
##             id       dam.1       dam.2 id.dam.cat
## 97  i108_2006_F   GreenBlue i081_2005_F         GG
## 108 i122_2007_M   GreenBlue i081_2005_F         GG
## 109 i123_2007_F OrangeGreen       F0001         GD
## 110 i124_2007_M   GreenBlue i081_2005_F         GG
## 114 i130_2007_F OrangeGreen       F0001         GD
## 115 i131_2007_F OrangeGreen       F0001         GD
## 117 i133_2007_F OrangeGreen       F0001         GD
## 122 i142_2008_M   GreenBlue i081_2005_F         GG
## 126 i147_2008_F  YellowBlue       F0003         GD
## 127 i148_2008_F GreenYellow       F0005         GD
## 134 i159_2008_F  YellowBlue       F0005         GD
```

```
PedM <- PCG$MergedPed[, c("id", "dam.1", "dam.2")]   # short-hand to minimise typing

# who are the mismatching individual's siblings according to Ped1 & Ped2?
PedM[which(PedM$dam.1 == "GreenBlue"), ]
```

```
##             id     dam.1       dam.2
## 97  i108_2006_F GreenBlue i081_2005_F
## 108 i122_2007_M GreenBlue i081_2005_F
## 110 i124_2007_M GreenBlue i081_2005_F
## 122 i142_2008_M GreenBlue i081_2005_F
```

```
PedM[which(PedM$dam.2 == "i081_2005_F"), ]
```

```
##             id     dam.1       dam.2
## 97  i108_2006_F GreenBlue i081_2005_F
## 108 i122_2007_M GreenBlue i081_2005_F
## 110 i124_2007_M GreenBlue i081_2005_F
## 122 i142_2008_M GreenBlue i081_2005_F
```

```
# who are their maternal grandparents?
PedM[which(PedM$id.1 == "GreenBlue"), ]
```

```
## [1] id    dam.1 dam.2
## <0 rows> (or 0-length row.names)
```

```
PedM[which(PedM$id.2 == "i081_2005_F"), ]
```

```
## [1] id    dam.1 dam.2
## <0 rows> (or 0-length row.names)
```

7.3.3 Colony

To compare Colony output with an existing pedigree, use:

```
BestConfig <- read.table("Colony/file/file.BestConfig",
                         header=T, sep="", comment.char="")
PC <- PedCompare(Ped1 = ExistingPedigree,
                 Ped2 = BestConfig)
```

7.4 Relatedness

7.4.1 Pedigree relatedness

Another way to compare two pedigrees is to compare the pedigree-based relatednesses calculated from each. In R, pedigree relatedness can for example be calculated by the R package `kinship2` – relatedness is defined as twice the kinship coefficient.

`kinship2` requires a pedigree with columns id - dadid - momid - sex, and requires that individuals either have two parents, or zero. The latter can be achieved with `sequoia::PedPolish(, FillParents=TRUE)`. From `sequoia` version 2.1, `CalcRPed()` will reformat the pedigree, call `kinship2::kinship()`, and return relatedness coefficients as a matrix or dataframe.

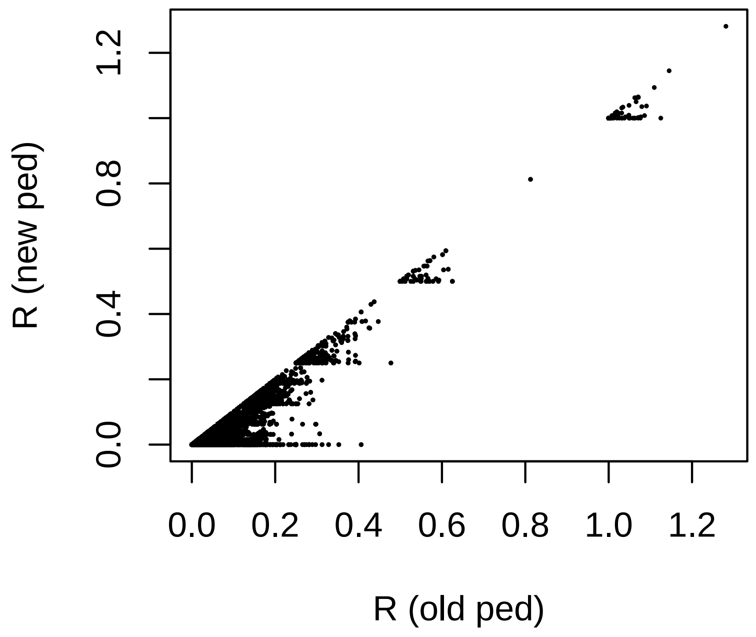
The same can be done for a second pedigree, and the two dataframes merged. As the number of pairs p becomes very large even for moderate numbers of individuals n ($p = n \times (n-1)/2$, so e.g. 2 million pairs for 2 thousand individuals), it is a good idea to use package `data.table` to assist with merging (merge in seconds vs. many minutes or not at all):

```r
Rped.old <- CalcRped(Ped_griffin, OUT="DF")
Rped.new <- CalcRped(SeqOUT_griffin$Pedigree, OUT="DF")
library(data.table)
Rped.both <- merge(data.table(Rped.old, key=c("IID1", "IID2")),
                   data.table(Rped.new, key=c("IID1", "IID2")),
                   all=TRUE, suffixes=c(".old", ".new"))

cor(Rped.both$R.ped.new, Rped.both$R.ped.old, use="pairwise.complete")
```

```
## [1] 0.9537727
```

```r
plot(Rped.both$R.ped.old, Rped.both$R.ped.new, pch=16, cex=0.3,
     xlab = 'R (old ped)', ylab = 'R (new ped)')
```



7.4.2 Genomic relatedness

In absence of a previous pedigree, or when it is not obvious whether the previous or newly inferred pedigree is correct, one could compare the pairwise relatedness estimated from the pedigree(s) to a measure of genomic relatedness, estimated directly from the complete SNP data – which may be many more SNPs than used for pedigree reconstruction. Genomic relatedness can be estimated for example using GCTA. Genomic relatedness will vary around the pedigree-based relatedness even for a perfect pedigree due to Mendelian variance, but outliers suggest pedigree errors.

For example:

```r
# read in output from GCTA
Rel.snp <- read.table("GT.grm.gz")
Rel.id <- read.table("GT.grm.id", stringsAsFactors=FALSE)
Rel.snp[,1] <- as.character(factor(Rel.snp[,1], labels=Rel.id[,2]))
Rel.snp[,2] <- as.character(factor(Rel.snp[,2], labels=Rel.id[,2]))
names(Rel.snp) <- c("IID1", "IID2", "nSNPS", "R.GRM")
Rel.snp <- Rel.snp[Rel.snp$IID1 != Rel.snp$IID2,]  # between-indiv only
#
# combine with pedigree relatedness
Rel.both <- merge(data.table(Rel.snp[,c(1,2,4)], key=c("IID1", "IID2")),
                  data.table(Rped.both, key=c("IID1", "IID2")), all.x=TRUE)
Rel.both <- as.data.frame(Rel.both)  # turn back into regular dataframe
rm(Rel.snp, Rped.both, Rped.new, Rped.old)   # clean up: remove large dataframes
#
round(cor(Rel.both[, c("R.GRM","R.ped.new", "R.ped.old")],
          use="pairwise.complete"), 3)
#
# scatterplot doesn't work well with many thousand points
# >> use heatmap-like alternative, e.g. hexbinplot
hexbin::hexbinplot(Rel.both$R.GRM ~ Rel.both$R.ped.new,
                   xbins=100, aspect=1,
                   xlim=c(-.05,.9), ylim=c(-.2, .9),
                   xlab="Pedigree relatedness", ylab="Genomic relatedness",
                   trans=log10, inv=function(x) 10^x,
                   colorcut=seq(0,1,length=14), maxcnt=10^6.5,
                   colramp = function(n) {grDevices::hcl.colors(n, palette='Berlin')})
#
# if you want to add e.g. a diagonal line to that plot:
hb <- hexbin::hexbin(Rel.both$R.GRM ~ Rel.both$R.ped.new,
                     xbins=100, xbnds=c(-.05, .9), ybnds=c(-.2, .9),
             xlab="Pedigree relatedness", ylab="Genomic relatedness")
hbp <- hexbin::plot(hb,
                    trans=log10, inv=function(x) 10^x,
                    colorcut=seq(0,1,length=14), maxcnt=10^6.5,
                    colramp = function(n) {grDevices::hcl.colors(n, palette='Berlin')}
)
hexbin::hexVP.abline(hbp$plot.vp, a=0, b=1)
```

```r
knitr::include_graphics("hexbin.png")
```

7.5 Cluster families

Certain analyses, such as the Mendelian error check in PLINK, are done on a family-by-family basis. `FindFamilies()` takes a pedigree as input and clusters the individuals in as few families as possible, by repeatedly searching all ancestors and all descendants of each individual and ensuring those all have the same family ID.

This function does not take separate FID and IID columns in the input pedigree, rather these need to be joined together before running `FindFamilies()` (e.g. with `paste()`), and then split afterwards using

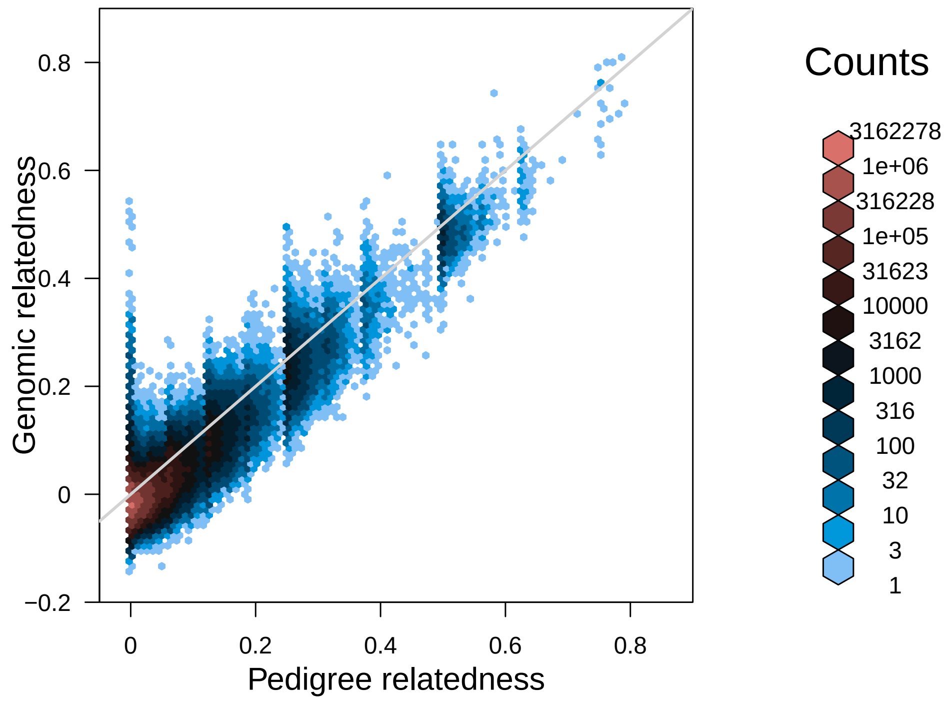


Figure 9: Relatedness hexbinplot example

`PedStripFID()`.

8 Hermaphrodites

Hermaphrodites have been re-implemented in versions 2.2 and 2.3, and now also full pedigree reconstruction should work as expected. Selfing is fully supported, also when a selfing parent or its selfed offspring are non-genotyped, and the aim is to infer their genotyped relatives.

With hermaphrodites it is not self-evident whether an identified parent should be assigned as dam or as sire. Sex-linked markers are not implemented in `sequoia`, and sequential hermaphroditism (male until age t, and female after, or vice versa) is not standard.

Two alternative approaches are available, by specifying parameter `Herm='A'` or `Herm='B'`.

8.1 Herm 'A': Pedigree prior

Chose this option if you are interested in whether a parent contributed the male versus female gamete to an offspring. This distinction can only be made if you have some non-hermaphroditic parents, or if you provide a 'pedigree prior' with e.g. candidate dams being the plant from which the seed was collected. A genetically identified parent will be assigned when it matches a candidate parent, or when it forms a complementary parent-pair with such an assigned parent. Candidate parent–offspring pairs that are not a genetic match will never be assigned, and non-genotyped candidate parents are ignored.

In absence of a pedigree prior and/or non-hermaphrodite parents, only instances of selfing can be assigned.

The 'pedigree prior' can only be used during parentage assignment, by specifying `SeqList[["PedigreePar"]]` as input *and* setting `Module='par'`:

```
cand.dams <- read.table("Candidate_dams.txt", header=TRUE,
                        stringsAsFactors=FALSE)
# cand.dam has columns 'id' and 'dam', and does not need entries for all ids
cand.par <- cbind(cand.dams, sire=NA)

par.herm <- sequoia(GenoM = Geno,
                    LifeHistData = LH,
                    SeqList = list(PedigreePar = cand.par),
                    Module='par')

# re-use all settings (including the newly assigned parents):
seq.herm <- sequoia(GenoM = Geno,
                    LifeHistData = LH,
                    SeqList = par.herm,
                    Module='ped')
```

8.2 Herm 'B': Ignore sex role

Use `Herm='B'` if it is irrelevant whether the parent was the maternal or paternal parent. With this setting single parents are assigned as dams, and in parent-pairs dam vs sire is mostly determined by their order in the genotype file. No conclusions can be drawn from whether individuals are assigned as maternal, paternal or 'cross' half-siblings.

8.3 Hermaphrodite dummies

Consequently, with `Herm='B'`, it is arbitrary whether half-siblings share a dummy dam or a dummy sire. If one or more individuals in a sibship are the product of selfing, the maternal and paternal halves will be linked, and additional half-siblings will be arbitrarily assigned to either half. Such dummy individuals get dummy prefix `'H'` in the output. An alternative 3rd `DummyPrefix` can be specified in (most?) relevant functions.

In `SeqList[["DummyIDs"]]`, the offspring-as-dam and offspring-as-sire are shown separately, with the same id ('H0xxx') and Sex=1 vs Sex=2 respectively. Selfed offspring will occur in both rows. Note that estimated birth years and other aspects may differ somewhat between the maternal and paternal halves, as during pedigree reconstruction they are represented as two 'linked' dummy individuals of opposite sex.

8.4 Duplicate check

The duplicate check may return a different number of putative duplicate pairs when `Herm` is A or B then when `Herm='no'`, as various relationships involving selfing are now also considered as alternatives.

8.5 Assignment errors

`PedCompare()` is not useful in combination with `Herm='B'`, as any dam assigned as sire and v.v. will show up as error. Use instead `ComparePairs(, patmat=FALSE)`, which will also pick up half-siblings for which the mother of i is the father of j.

One type of assignment error is unavoidable: if the true parent is unsampled and the product of selfing, a sampled grandparent will be indistinguishable from a parent. Similarly, when the true parent is sampled, but both the parent and the grandparent are the product of selfing, they are genetically nearly identical and often erroneously interchanged when birth year information is absent.

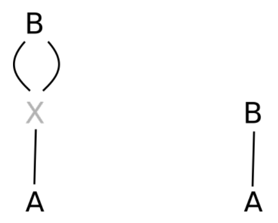


Figure 10: Two configurations with identical likelihoods

8.6 Parthenogenesis

There are several types of parthenogenesis or asexual reproduction (https://en.wikipedia.org/wiki/Parthenogenesis). From a single-SNP perspective, some resemble selfing (joining of two gametes produced by the same individual), and females can be represented as hermaphrodites in sequoia. Parthenogenesis were genetically identical copies or haploids are produced, are not implemented.

9 FAQ

9.1 Error about/due to input data

For smaller datasets (say up to 1000 individuals) it may be useful to read in the data with `read.table()` (using `header=TRUE` or `FALSE`, as appropriate, and typically `row.names=1`), and inspect the data with `table(as.matrix(mydata))` for odd entries that weren't caught by `CheckGeno()` to give an informative error message. For very large datasets, specialist tools may be required to get the data in a standardised format.

9.2 Low assignment rate

See also key points.

In many cases, assignment rate can be boosted without increasing the number of SNPs: by assuming a different genotyping error rate, providing sex or birth year information on more individuals, or fine-tuning the ageprior. Sometimes *reducing* the number of SNPs may increase assignment rate, by removing those with the highest error rate and/or SNPs in close LD.

9.2.1 Not enough SNPs

As opposed to most other pedigree assignment programs, Sequoia does not rely on MCMC to explore many different pedigree possibilities, but instead sequentially assigns highly likely relationships, and expands the pedigree step by step. For a relationship to be highly likely, a substantial number of SNPs is necessary, larger than for MCMC methods: at least 100-200 for parentage assignment, or full sibling clustering in a monogamous population, and about 400-500 otherwise.

9.2.2 Too many SNPs

More is not always better: with tens of thousands of SNPs many will unavoidably be in high LD (unless you work on a species with a very large genome and very high N_e), and the signal will get lost in the noise. Using more than 1000 SNPs is never necessary, and for a typical vertebrate genome there is little benefit of using more than 500 good SNPs.

This unfortunately also means that sequoia is unsuitable for species with a very small genome.

If you have tens of thousands of SNPs, the easiest way to down sample is to pick SNPs with the lowest missingness and/or the highest MAF. In theory filtering also on LD is even better, but average LD will always be lower in any subset of 500(-ish) SNPs than in the full set.

9.2.3 Genotyping errors

A few SNPs with a high (apparent) error rate may throw off pedigree reconstruction if their erroneous signal is not sufficiently offset by a large number of more accurate SNPs. Such SNPs can be identified with `SnpStats()`, using an existing pedigree or one from an preliminary round of parentage assignment. If there are clear outliers with regards to estimated error rate, it may be worth exploring pedigree reconstruction without these SNPs.

9.2.4 Lacking sex information

Currently Sequoia cannot handle sex-linked markers, and therefore cannot distinguish between maternal versus paternal relatives. Sometimes the sex of an individual can be inferred, if it forms a complementary parent-pair with an individual of known sex. This is also the manner in which the sex of dummy parents is determined; half-sibships sharing a parent of unknown sex are not currently implemented.

Pairs of relatives for which it is unclear whether they are maternal or paternal relatives, can be identified with `GetMaybeRel()`.

9.2.5 Insufficient age information

A parent will only be assigned if it is known to be older than the individual with which it genetically forms a parent–offspring pair, or if a complementary co-parent is identified. Purely genetically, it is impossible to tell who is the parent in a parent–offspring pair. But once an individual has been assigned parents, any remaining individuals with which it forms a parent-offspring pair must be its offspring, and will be assigned as such. Thus, a high proportion of sampled parents may somewhat compensate for unknown birth years.

Providing minimum and maximum possible birth years can substantially increase assignment rate. The risk is that when intervals are taken too narrowly, parent–offspring pairs are flipped the wrong way around, which in turn may lead to other wrong assignments. This may especially be a problem in long-lived species which start breeding at an early age.

Age information will also help to distinguish between the three different types of second degree relatives (half siblings, grandparent – grand-offspring and full avuncular (aunt/uncle – niece/nephew). These are genetically indistinguishable, unless both individuals of the pair already have a parent assigned. In most species, there is limited overlap between the age-difference of half siblings versus grandparents, and only partial overlap of either with the age distribution of avuncular relationships.

9.2.6 Uninformative age prior

The ageprior used for full pedigree reconstruction is estimated from the birth year differences of parent-offspring pairs assigned during the initial parentage assignment. When only few parents are assigned, or when many birth years are unknown, this ageprior may not be very informative. If a large pedigree is available from the same or a similar population (e.g. based on observations and microsatellite paternity assignments), it can be useful to estimate the agepriors from that pedigree. For details, see the age vignette

9.2.7 Mating system

When the population has a complex mating system, with overlapping generations and many double relatives, a large number of SNPs is needed to distinguish between various plausible alternatives. When the power of the SNP panel is insufficient to make the distinction, no assignment will be made.

When inbreeding and complex relationships (e.g. paternal half-sibling as well as maternal half-aunt) are rare, ignoring these typically increases assignment rate (`Complex="simp"`). Similarly, when polygamy is rare and not of particular interest, assuming a monogamous mating system typically increases assignment rate (`Complex="mono"`). However, these choices will risk erroneous assignments when complex relationships or polygamy, respectively, do occur.

9.3 Death year

The year of death forms an upper limit to when an individual could have reproduced, and can from version 2.5 onwards be used as input via the `Year.last` column in `LifeHistData`. Note that for many species, the last possible year in which offspring can be born may be after the year of death (e.g. males in many mammal species), or the year preceding the year of death (if a individual died between January 1st and the breeding season).

If it is unclear whether an individual died or emigrated, please do not provide a `Year.last` – emigrants can still be candidate parents, as they may reside just outside the study area boundaries, or return briefly and unseen during the breeding season. is unknown.

9.4 Accuracy

The real, true pedigree is normally unknown, but there are a few ways to infer the accuracy of specific pedigree links, or to estimate the average accuracy of assignments.

9.4.1 Field pedigree

If the genetically assigned mother matches the mother caring for the individual, or the plant from which the seed was collected, there is little reason to doubt the assignment. Similarly, when the genetically assigned father matches (one of) the observed mates of the mother, the assignment is most likely correct.

In rare other cases, the genetically assigned parent is impossible, for example because the assigned parent was not alive at the time of birth (for mothers) or conception (for fathers). The blame for such an erroneous assignment may be the pedigree reconstruction software (due to genotyping errors, or a bug in the code), but may also be due to sample mislabelling in the lab, or a case of mistaken identity in the field.

Tracking down the underlying cause of discrepancies between the field and genetic pedigree will be time consuming, but can be a valuable part of data quality control.

9.4.2 Genomic relatedness

When many thousands of SNPs are typed, it is possible to calculate the genomic relatedness (R_{grm}) between all pairs of individuals (see Relatedness). Due to the random nature of Mendelian inheritance there is always considerable scatter of genomic relatedness around pedigree relatedness, but when the pedigree relatedness is considerably higher (say $R_{ped} - R_{grm} > 0.2$), this is often indicative of a pedigree error. Note however that most estimators of R_{grm} assume a large, panmictic, non-inbred population, and deviations from these assumptions may contribute to differences between R_{ped} and R_{grm}.

9.4.3 Data simulation

Simulations as performed by 'EstConf()' do not tell which assignments may be incorrect, but do give an estimate of the overall number of incorrect assignments. The simulations are done presuming the inferred pedigree (or an existing pedigree) is the true pedigree, i.e. for a pedigree that is (hopefully) very close to the actual true pedigree. It relies on several simplifying assumptions, and will therefore always be an optimistic estimate of the actual accuracy.

9.5 Add new individuals

When new individuals have been genotyped, such as a new cohort of offspring, it is best to re-run the pedigree reconstruction for all genotyped individuals. This ensures that older siblings of the new offspring are identified, as well as any grandparents. A possible exception is when all candidate parents have been genotyped, although even then inclusion of their parents (the new individuals' grandparents) may correct for any genotyping errors they may have.

9.6 Results differ from COLONY

Sequoia will typically produce a more conservative pedigree than COLONY, for various reasons:

- COLONY only considers whether individuals are full siblings, half siblings, or unrelated, while `sequoia` also considers full- or half- avuncular (aunts and uncles and their nieces and nephews), and grand-parental. These alternative relationships are impossible among individuals born in the same year when generations do not overlap (`args.AP=list(Discrete=TRUE)`). If all samples are from a single birth year, this is automatically assumed.
- `sequoia` also considers double relationships, e.g. paternal half-siblings that are also maternal half-cousins, with a relatedness inbetween full and half siblings. These may be erroneously assigned as full siblings by COLONY when present, but their consideration may cause true full siblings to be only assigned as half-siblings, especially when there is only a moderate amount of genetic data. To ignore double and inbred relationships, use `Complex='simp'`.
- `sequoia` considers third degree relationships, such as full cousins. Again, these may result in false positives by COLONY, while their consideration may result in false negatives by `sequoia`. This can not be turned off.
- When no parents of known sex can be assigned, there is no way in `sequoia` to differentiate between maternal and paternal half-siblings, and none are assigned (although they may be identified with `GetMaybeRel()`. In COLONY, candidate mothers and fathers can be specified, but this option is not available in `sequoia` (yet?).
- When a relationship is uncertain, COLONY will include it in the output with a low confidence probability. `sequoia` only makes assignments of which it is very sure, as the sequential nature of the algorithm may otherwise result in a snowball effect of incorrect assignments.

You can get `sequoia`'s 'opinion' on a COLONY reconstructed pedigree (or any other pedigree) with `CalcOHLLR()`, which for every assigned parent gives the log-likelihood ratio between being parent-offspring with the focal individual, versus otherwise related. You may also use `CalcPairLL()` for a specific set of pairs of individuals to calculate their likelihoods to be parent-offspring, full siblings, … , and investigate how these likelihoods change when changing the ageprior or with `Complex='simp'`.

References

Huisman, Jisca. 2017. "Pedigree Reconstruction from SNP Data: Parentage Assignment, Sibship Clustering and Beyond." *Molecular Ecology Resources* 17 (5): 1009–24.

Marshall, T C, J B K E Slate, L E B Kruuk, and J M Pemberton. 1998. "Statistical Confidence for Likelihood-Based Paternity Inference in Natural Populations." *Molecular Ecology* 7 (5): 639–55.

Thompson, E A, and T R Meagher. 1987. "Parental and Sib Likelihoods in Genealogy Reconstruction." *Biometrics*, 585–600.