sequoia

Reconstruction of multi-generational pedigrees from SNP data

Jisca Huisman (jisca.huisman @ gmail.com)

March 29, 2018

Contents

0.1 Quick-start example

An example pedigree and associated life history data are provided with the package, which can be used to try out the steps detailed here. This fictional pedigree consists of 5 generations with interconnected half-sib clusters (Pedigree II in [1]).

```
> 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)
> tail(Ped_HSg5)
> #
> # simulate genotype data for 200 SNPs
> Geno <- SimGeno(Ped = Ped_HSg5, nSnp = 200)
> #
> # run sequoia - duplicate check & parentage assignment only
> # (maximum number of sibship-clustering iterations = 0)
> ParOUT <- sequoia(GenoM = Geno,
+                   LifeHistData = LH_HSg5,
+                   MaxSibIter = 0)
> names(ParOUT)
> # [1] "Specs"  "AgePriors"  "LifeHist"  "PedigreePar"  "MaybeParent"
> # "TotLikParents"
> #
> # run sequoia - sibship clustering & grandparent assignment
> # use parents assigned above (in 'ParOUT$PedigreePar')
> SeqOUT <- sequoia(GenoM = Geno,
+                   SeqList = ParOUT,
+                   MaxSibIter = 5)
> #
> # compare the assigned real and dummy parents to the true pedigree
> chk <- PedCompare(Ped1 = Ped_HSg5, Ped2 = SeqOUT$Pedigree)
> chk$Counts
> #
> # save results
> save(SeqOUT, file="Sequoia_output_date.RData")
> writeSeq(SeqList = SeqOUT, GenoM = Geno, PedComp = chk,
+           folder = "Sequoia-OUT")
```

0.2 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 [1]. Explicit consideration of the likelihoods of alternative relationships before making an assignment reduces the number of false positives, compared to parentage assignment methods relying on the likelihood ratio parent-offspring versus unrelated only [4]. When genetic information is abundant, the heuristic, sequential approach used is considerably quicker than most alternative approaches, with little or 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.

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.

1 Input

1.1 Life history data

The life history data (`LifeHistData`) should be a dataframe with three columns:

- 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" in `?make.names`.
- Sex: 1 = female, 2 = male, other numbers or NA = unknown (except 4 = hermaphrodites [under development, for now possible in parentage assignment only])
- BY: 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 born prior to their putative offspring (e.g. parent's BY=2001 and offspring BY=2005, or BY=1 and BY=5 respectively). Negative numbers and NA's are interpreted as unknown.

The column names are ignored, and therefore the order of the columns is critical. Ideally this basic life history information is provided for all genotyped individuals, but this is not necessary. This dataframe may include many more individuals than the genotype data, or in a different order.

1.2 Genotype data

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.

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

The 0/1/2 format can for example be obtained using PLINK (`https://www.cog-genomics.org/plink2`) [3] in combination with `sequoia`'s `GenoConvert()`, as described below. `GenoConvert` can also convert Colony input files.

1.2.1 Real data - Selection of SNP markers

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 in low linkage disequilibrium (LD) with each other, and with

high minor allele frequencies (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. 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.

Creating a subset of SNPs can be done conveniently using PLINK, using for example in command prompt (or linux terminal) the command

```
plink --file mydata --maf 0.4 --indep 50 5 2
```

which on a windows machine is equivalent to running inside R

```
> system("cmd", input = "plink --file mydata --maf 0.4 --indep 50 5 2")
```

This will create a list of SNPs with a minor allele frequency of at least 0.4, 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 `https://www.cog-genomics.org/plink2/ld#indep`.

It is advised to 'tweak' the parameter values until a set with a few hundred SNPs (300–700) is created. It is possible to further select the subset based on the number of Mendelian errors in a preliminary pedigree, see `?SnpStats`.

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")
```

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

1.2.2 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 lifehistory 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`.

1.2.3 Very large datasets

When the number of individuals is very large, loading the genotype data into R will take up a lot of memory, and may even exceed R's memory limit and be impossible. 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 `https://github.com/JiscaH`. Using this requires a Fortran95 compiler, for example gfortran which comes with the linux-emulator 'Cygwin' for windows. The input consists of three text files: the lifehistory data; the genotype data with one column for IDs followed by one column per SNP (0/1/2/-9), and no header row; and the parameter settings, for which an example file is included with the code. These files can be generated using `writeSeq`, for example after running sequoia on a subset of the data. No manual for this has been written yet, please email jisca.huisman@gmail.com if you intend to use this and require help.

1.2.4 Simulating SNP data

When SNP data is not (yet) available, but an approximate pedigree is, it is possible to test `sequoia` on a simulated dataset. This may be useful to for example explore the number of markers required to reliably infer a particular pedigree structure. Alternatively, this can be used to estimated the pedigree-wide error rate of an inferred pedigree (see section 4.2).

The function `SimGeno()` lets the user specify the average proportion of missing genotypes per individual (`MisHQ`), the genotyping error rate (`ErHQ`), and the fraction of known parents (in the supposed 'true' pedigree) which have not been genotyped (`ParMis`). Moreover, the data can be made to contain a fraction of low-quality samples (`PropLQ`, with associated `MisLQ` and `ErLQ`), to assess whether inclusion of samples which did not pass stringent quality control would improve or hamper pedigree reconstruction.

1.3 Parameters

DummyPrefix The prefixes for dummy individuals (sham parental IDs assigned to sibship clusters) can be altered to avoid confusion with IDs of real individuals. Defaults to 'F' for females ('F0001', 'F0002', ...) and 'M' for males ('M0001', 'M0002', ...).

Err The genotyping error rate assumed, typically probably around 1E-4 to 1E-3. The error model is given in Table 1; other error structures could easily be implemented but are currently not user-settable.

MaxMismatch The maximum number of loci at which candidate parent and offspring are allowed to be opposite homozygotes, used to filter out highly unlikely pairs. Note that the actual upper limit used is `MaxOH = MaxMismatch + ceiling(Err * nSnp)`.

Table 1: Default probabilities used of observing genotype X, conditional on actual genotype x.

	X		
x	0	1	2
0	1-ϵ	ϵ	0
1	$\epsilon/2$	1-ϵ	$\epsilon/2$
2	0	ϵ	1-ϵ

MaxSibIter The number of iterations of sibship clustering. As this is by far the most time consuming step, and may take several hours for large datasets, it would be wise to first run with `MaxSibIter=0` so that only the much faster parentage assignment is performed, and inspect the output. If during sibship clustering the total likelihood asymptotes before `MaxSibIter` is reached, the algorithm is terminated and the results returned.

MaxSibshipSize Maximum number of offspring for a single individual. A generous safety margin is advised of at least twice the biologically plausible maximum.

Tassign Threshold log10-likelihood ratio (LLR) required for acceptance of a proposed relationship, relative to next most likely relationship. Must be zero or positive, with higher values resulting in more conservative assignments.

Tfilter Threshold LLR between a proposed relationship versus unrelated, to select candidate relatives. Typically negative, and more negative values may prevent filtering out of true relatives, but will increase computational time.

Complexity When it is known that the dataset contains only monogamous matings, the assignment rate can be improved by using the option `Complexity='mono'`. [under development . . .]

1.3.1 Re-use of previous output

The parameter values used as arguments when calling `sequoia` will be returned in the list element `Specs`. These settings can be re-used in a subsequent run, optionally afer changing them

```
> load("Sequoia_output_date.RData")  # if it was saved to disk
> ParOUT$Specs
> #   NumberIndivGenotyped NumberSnps GenotypingErrorRate MaxMismatch
> # 1                  920        200               1e-04           3
> #   Tfilter Tassign nAgeClasses MaxSibshipSize MaxSibIter
> # 1      -2     0.5           6            100          0
```

```
> #   DummyPrefixFemale DummyPrefixMale Complexity FindMaybeRel CalcLLR
> # 1                 F               M       full          TRUE    TRUE
> ParOUT$Specs$DummyPrefixFemale <- "D-FEM"
> ParOUT$Specs$DummyPrefixMale <- "D-MALE"
> SeqOUTX <- sequoia(GenoM = Geno,
+                    SeqList = list(Specs = ParOUT$Specs),
+                    MaxSibIter = 10)
```

When `SeqList` is provided and contains an element named `Specs`, all other (default) parameter values are ignored, *except* `MaxSibIter`. It is also possible to re-use the entire output list,

```
> SeqOUT <- sequoia(SeqList = ParOUT)
```

which will use both `AgePriors` and `PedigreePar` in 'ParOUT', as detailed below.

2 Running Sequoia

Under the hood, `sequoia` consists of four sub-programs:

1. **Duplicates:** Check for duplicate entries in the genotype and life history data
2. **Agepriors:** Calculation of age-difference based prior probability ratios
3. **Parentage:** Parentage assignment (assign genotyped parents to genotyped focal individuals)
4. **Sibships:** Clustering of half- and full-siblings, grandparent assignment to singletons and sibships, and identification of avuncular relationships between sibships (jointly referred to as 'Sibships' for brevity)

these all return their output to a single list, with the elements listed in Table 4 and detailed in section 3.

2.1 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 a `MaxMismatch` mismatches between the genotypes, which may or may not have the same individual ID. Note that very inbred individuals may be nearly indistinguishable from their parent(s), especially when the number of SNPs is limited. Additionally, the genotype and life-history files are checked for duplicate IDs.

It 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 (but not always, see section 3.3).

2.2 Age difference based prior

Based on the species' age at first and last reproduction, some age differences between parent and offspring or between siblings are more likely than others, and some downright impossible. The age differences calculated from the birth years provided in `LifeHistData` are used as a secondary source of information, amongst others to help distinguish between half-siblings, grandparent–grand-offspring and full avuncular pairs.

The list element `AgePriors` contains 8 columns, and as many rows as the birth year range detected in the life history data. It initially only indicates whether a given relationship is biologically possible (1) or not (0) for a given age difference between individuals, for any species (e.g. parents and their offspring can never be exactly the same age). The first row is for individuals born in the same year, the second row for individuals born one year apart, etc. The columns are labelled for various relationship categories, with M = mother, P = father, MS = maternal sibling, PS = paternal sibling, MGM = maternal grandmother, PGF = paternal grandfather, MGF = maternal grandfather and paternal grandmother, and AU = avuncular (niece/nephew – aunt/uncle).

For example, the first value in the column 'MS' can be interpreted as 'if I were to pick two individuals born in the same year, and two individuals from my sample at random, how much more likely are the first pair to be maternal siblings, compared to the second pair?' Or to phrase it differently: 'Now that I learned that these individuals are born in the same year, does that make them more likely or less likely to be maternal siblings than before I knew this?' Values below 1 indicate less likely, and values above 1 more likely. For MS, PS and AU absolute age differences are used (with overlapping generations, nephews may be older than their aunts), while parents and grandparents are necessarily older than their (grand-)offspring (categories M, P, MGM, PGF and MGF).

These age-difference based priors are by default automatically updated after parent-age assignment, based on the empirical distribution of age differences between individuals and their assigned fathers and mothers. This update is prevented when `SeqList` is provided and contains an element `AgePriors` (see Table 2).

Table 2: Behaviour when 'AgePriors' and/or 'PedigreePar' are provided in 'SeqList'. –: not provided / not run; age prior categories are 'user'= user-provided, 'basic' = minimal restrictions, 'parents' = based on assigned parents

in SeqList		Age prior used	
AgePriors	PedigreePar	Parentage	Sibships
–	–	basic	parents
user	–	user	parents
–	Y	–	parents
user	Y	–	user

`AgePriors` can be altered to match the biological characteristics of the species, but the number of rows must not be decreased, and the column order kept as it is. If the number of rows is increased, `Specs['nAgeClasses']` should be updated to match the new number of rows.

M	P	MGM	PGF	MGF	FS	MS	PS	UA
0	0	0	0	0	1	1	1	0
1	1	0	0	0	0	0	0	1
0	0	1	1	1	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

(note that column order changed between v0.9 and v0.10, and column FS was added)

Table 3: Example age-difference prior, for non-overlapping generations

For example, for a species with strictly non-overlapping generations, one may wish to alter `AgePriors` to the matrix in Table 2.2, which can be done as follows

```
> AP <- as.matrix(SeqOUT1$AgePriors)
> AP[AP>0] <- 0
> AP[1,c("MS", "PS")] <- 1
> AP[2,c("M", "P", "UA")] <- 1
> AP[3,c("MGM", "PGF", "MGF")] <- 1
> SeqOUT2 <- sequoia(SeqList=list(Specs=SeqOUT1$Specs, AgePriors=AP),
+                    MaxSibIter = 0)
```

Note that any identified parent-offspring pairs which are not exactly 1 year / time unit apart will be returned in `MaybeParent` (section 3.3). It is possible to enforce the same age-difference prior on the sisbhip clustering as well, but only if parentage assignment and sibship clustering are run separately (see Table 2)

2.3 Parentage assignment

Assignment of genotyped parents to genotyped offspring is performed by default, unless earlier-assigned parents are provided in `SeqList$PedigreePar`.

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. Therefore, 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 or equal to the per-SNP genotyping error rate `Err` times the number of SNPs (rounded up to nearest whole number), plus the safety margin `MaxMismatch`,
- The likelihood ratio between being parent and offspring versus unrelated, not conditioning on any already assigned parents, must be equal to or greater than `Tfilter`,

- The likelihood ratio between the pair being parent and offspring versus being otherwise related must be equal to or greater than `Tassign`, to filters out siblings, grandparents and aunts/uncles,

and 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, the pair is returned in `MaybeParent` (section 3.3). If there are multiple candidate parents of the same sex, or some of unknown sex, the parent pair or single parent resulting in the highest likelihood is assigned. If a parent pair is identified but both sexes are unknown, such that it is unclear which is the father and which the mother, they are returned in `MaybeParentPairs`.

This heuristic sequential filtering approach makes parentage assignment quick, and for example takes less than a minute for an empirical dataset with 2 500 genotyped individuals on a laptop with an intel i7 2.3 GHz CPU and 8GB RAM

2.4 Sibship clustering & the rest

Full pedigree reconstruction, including sibship clustering amongst those individuals which have not been assigned two genotyped parents, is performed when `MaxSibIter`> 0. This 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. During this phase, all first and second degree links between individuals are attempted to be assigned, using the following steps in each iteration

- Find pairs of full- and half-siblings
- Cluster sibling pairs into sibships
- Find grandparent – grand-offspring pairs (round 3+)
- Merge existing sibships
- Replace dummy parents by genotyped individuals (round 2+)
- Add lone individuals to sibships (round 2+)
- Assign genotyped parents to genotyped individuals
- Assign grandparents to sibships (round 2+; grandparents may be dummy individuals as well as genotyped individuals)

The total likelihood (section 3.4) typically asymptotes within five to ten iterations, even for complex pedigrees. When an asymptote is reached before `MaxSibIter`, dependency on the age prior is increased (if `UseAge` = 'extra')and the algorithm continues until a new asymptote or `MaxSibIter` is reached. Then, parental likelihoods are calculated, a check is done for non-assigned potential relatives, and the algorithm is terminated. These last steps may take considerable time, and either or both can be skipped by specifying `CalcLLR` = FALSE and/or `FindMaybeRel` = FALSE.

Table 4: Output from Sequoia, returned within a named list.

Output	Description
AgePriors	Age-difference based prior probabilities
DummyIDs	Details per half-sib cluster
DupGenoID	Duplicated IDs in genotype data
DupGenotype	(near) Duplicated genotypes
DupLifeHistID	Duplicated IDs in life history data
LifeHist	sex and birth year data
MaybeParent	Non-assigned likely PO pairs
MaybeRel	Non-assigned likely relatives
NoLH	IDs in genotype data not present in life history data
Pedigree	Pedigree
PedigreePar	Scaffold pedigree
Specs	Parameter values
TotLikParents	Total likelihood during parentage
TotLikSib	Total likelihood during sib clustering

3 Output

Beside the inferred pedigree (section 3.1), `sequoia` also returns summary information of the dummy parents (section 3.2), any pairs of individuals which are likely to be relatives but could not be assigned as such (section 3.3), the total likelihood of the data after each iteration (section 3.4), and the input data and parameters (except the large genotype data) (full overview in Table 4).

3.1 PedigreePar & Pedigree

`PedigreePar` is the scaffold pedigree returned after assigning genotyped parents to genotyped offspring. `Pedigree` additionally includes dummy individuals, assigned to infered groups of half-siblings for which the shared parent is not genotyped. Note that dummy individuals are also assigned as the 'in-between' individual of identified grandparent – grand-offspring pairs. Dummy individuals are appended at the bottom of the pedigree with their assigned parents, i.e. the sibship's assigned grandparents, and by default have IDs 'F0001', 'F0002', ... for dams and 'M0001', 'M0002', ... for sires (sections 1.3 and 3.2).

The pedigrees columns are

- IDs of the individual, its assigned dam (mother) and sire (father),
- The log10 likelihood ratio (LLR) of the dam, sire and the parent pair; this is the ratio between the likelihood of the assigned parent being the parent, versus the most likely alternative type of being related to the focal individual (see Table 5),
- The number of loci at which the offspring and the assigned dam or sire are opposite homozygotes (`PedigreePar` only).

The parental LLRs are calculated at the very end, and are conditional on all other links in the reconstructed pedigree. The parent-pair LLR is relative to the most likely assignment of a single parent (or no parent). Note that this LLR differs from for example Cervus [2], which returns the natural log of the ratio between the probability that the assigned parent is the parent, versus that the next most likely candidate is the parent.

Some parents may have a very small or even negative single-parent LLR, but the LLR of the parent pair should ideally always be positive. For full sibling pairs and dummy-parents of dummy-individuals this is not always the case, due to some approximations used when calculating the parental LLR (which are not used during the assignment steps). It is however probably worthwhile to be cautious about assignments with low or negative LLRs, and for example compare with a previous pedigree (section 4.1) or the genomic relatedness (section 4.3).

Table 5: Pairwise relationships considered.

PO	Parent-offspring
FS	Full siblings
HS	Half siblings
GP	Grandparent – grand-offspring
FA	Full aunt/uncle – niece/nephew
HA	Half aunt/uncle – niece/nephew, or other 3rd degree relative
U	Unrelated

If some of the LLRs are very large negative or positive numbers, please send a bug report to `jisca.huisman@gmail.com` with a short description of your dataset — something probably went wrong.

3.2 DummyIDs

To each cluster of half-siblings a 'dummy' parent is assigned, denoted by increasing numbers, by default with prefix 'F' for females and 'M' for males (sections 1.3). `DummyIDs` is a dataframe with for each dummy individual

- the assigned dam and sire (the sibship's grandparent) and their associated LLRs, which can also be found in `Pedigree`
- its sex

- the estimated birth year, as a point estimate ('BY.est') and lower and upper bound of 95% probability interval ('BY.min' and 'BY.max'). These are based on the birthyears of the individuals in the sibship and of the sibship-grandparents, if any, in combination with `AgePriors`. This may help
- 'NumOff', the number of individuals in the sibship (= the dummy individuals number of offspring)
- the IDs of the individuals in the sibship, with column names 'O1', 'O2', ...

This information is intended to make it easier to associate dummy IDs to real IDs of observed but non-genotyped individuals (see also section 4.1).

3.3 MaybeParent & MaybeRel

`MaybeParent` countains probable or definite parent-offspring pairs which could not be assigned in `PedigreePar`, with columns

- ID1, ID2: identities of the pair
- Sex1, Sex2: sex of the individuals; 1=female, 2=male
- AgeDif: Age difference, positive numbers indicate that ID2 is older
- TopRel: Relationship with the highest likelihood, may be any of the abbreviations in Table 5, or '2nd' (undetermined type of second degree relative, see text). 'XX' indicates unclear, but more likely to be first or second degree relatives than unrelated.
- LLR: Log10 likelihood ratio (LLR) between the pair being related according to the most likely relationship (column 'TopRel') versus the next most likely relationship.
- OH: The number of loci at which the individuals are opposite homozygotes.

This dataframe includes cases where the pair is more likely to be parent-offspring than unrelated, but where it cannot be excluded that they are otherwise related ('LLR' between most likely and next most likely < `Tassign`), or were an alternative relationship is even more likely ('TopRel' not PO). Additionally, `MaybeParent` may include pairs which are most likely to be parent and offspring, but where lack of birth year information made it impossible to tell which of the two was the parent and which was the offspring ('AgeDif' = NA), or where lack of sex information of the older one made it impossible to tell whether this candidate parent is the mother or the father ('Sex2' = 3, see 'MaybeParentPairs' below).

`MaybeRel` includes pairs which are more likely to be first or second degree relatives than unrelated, but which could not be assigned in `Pedigree`. This includes for example half siblings where it is unclear whether they share a mother or a father. Distinguishing half siblings from grandparent–grand-offspring and full avuncular pairs is not straight forward either, and relies on either both individuals already having at least one parent assigned, or very strong support based on the age diference of the pair. When neither is the case, 'TopRel' indicates '2nd', and LLR is between being 2nd degree relatives versus the most likely of PO, FS, HA or U.

3.3.1 MaybeParentPairs

When the sex or birth year of many or all individuals is unknown, there will be cases where a particular individual (A) forms unassigned parent-offspring pairs with two or more other individuals (say B, C and D). Then, it is checked whether any of the candidate parents form a 'complementary' parent pair (B+C, C+D, B+D). These are returned in a similar format as the pedigree, but with headings 'parent1' and 'parent2' instead of 'dam' and 'sire'.

Use with caution, especially if both birth year and sex are unknown, as it seems that ocassionally actual offspring will form a likely parent pair, and the error rate is likely to be higher than for regular parent assignment.

3.4 TotLikParents & TotLikSib

These are vectors with the log10 of the approximate total likelihood of the pedigree, which is the probability of observing the genotype data, given the reconstructed pedigree, the allele frequencies of the SNPs, and the presumed genotyping error rate. The value at initiation (the first value in `TotLikParents`) is calculated assuming Hardy-Weinberg equilibrium in the sample. The subsequent value are at the end of each iteration of parentage assignment (`TotLikParents`) or sibshib clustering (`TotLikSib`, should be increasing across iterations, and asymptoting. If there is a large change in value between the second-last and last likelihood, consider running the algorithm for more iterations (increase `MaxSibIter`). One can do a visual check as follows:

```
> TLL <- c(SeqOUT$TotLikParents, SeqOUT$TotLikSib)
> xv <- c(paste("p", 1:length(SeqOUT$TotLikParents)-1),
+         paste("s", 1:length(SeqOUT$TotLikSib)-1))
> plot(TLL, type="b", xaxt="n", xlab="Round")
> axis(1, at=1:length(TLL), labels=xv)
```

The total likelihood is calculated assuming independent SNPs as

$$\mathcal{L} = \prod_{A=1}^{N} \prod_{l}^{L} \sum_{y} \sum_{z} P(A_l = X | DA_l = y, SA_l = z, \epsilon) P(DA_l = y) P(SA_l = z) \quad (1)$$

or the probability of observing individual A's genotype X at SNP l, given the true genotypes y and z of it assigned parents DA and SA, multiplied over all individuals and all SNPS. For example, if X is a heterozygote, the probability of this genotype is 1/2 if y is heterozygous and z a homozygote, 1 if y and z are opposite homozygotes, and 0 (or $\epsilon/2$ when allowing genotyping errors, Table 1) if y and z are identical homozygotes. This is summed over all possible parental genotypes, weighed by the probabilities that the parent have true genotype y and z. These probabilities are determined by the parent's observed genotypes and the genotyping error rate for genotyped parents, or according to Hardy-Weinberg proportions for non-assigned parents. For dummy parents, the probability depends on A's siblings and grandparents (see [1]).

3.5 Save output

There are various ways in which the output can be stored. This includes saving the seqoia list object, and optionally any other object, in an .RData file

```
> save(SeqList, LHdata, Geno, file="Sequoia_output_date.RData")
```

which can be read back into R at a later point

```
> load("Sequoia_output_date.RData")
> # 'SeqList' and 'LHdata' 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 (since v0.10) using `writeSeq`:

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

which also creates a README file, to remind one that this was created by sequoia and the date. This can be used for any notes or comments, and any R scripts could be saved in the same folder.

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

```
> 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(xlsx)
> write.xlsx(Geno, file = "Sequoia_OUT.xlsx", sheetName="Genotypes",
+       col.names=FALSE, row.names=TRUE, append=TRUE, showNA=FALSE)
```

The option `append=TRUE` ensures that the sheet is appended to the file, rather than the file overwritten.

4 Output check

4.1 Comparison with previous pedigree

Often times, a (part) pedigree is already available to which one wants to compare the results, for example consisting of maternal links, deduced from observations in the field. The function `PedCompare()` performs such comparisons, and takes as arguments the 'true' pedigree as `Ped1`, and the newly inferred pedigree as `Ped2`:

```
> compareOUT <- PedCompare(Ped1 = Ped_HSg5, Ped2 = SeqOUT$Pedigree)
```

Where the output list consists of `Counts`, a summary of the number of matches and mismatches between the two pedigrees, as well as `MergedPed`, a side-by-side comparison, and `ConsensusPed`, an amalgamation of the two. `PedCompare()` does its best to align any dummy parents in the inferred pedigree 2, to non-genotyped individuals in pedigree 1.

Counts An array printed as two 7x5 matrices, one for dams and one for sires. When checking the results from parentage assignment only, only the rows 'GG' (Genotyped focal - Genotyped parent) are relevant:

```
> compareOUT2 <- PedCompare(Ped1 = Ped_HSg5, Ped2 = ParOUT$Pedigree)
> compareOUT2$Counts["GG",,]
> #          dam sire
> # Total    130  170
> # Match    128  166
> # Mismatch   0    0
> # P1only     2    4
> # P2only     0    0
```

Further details, amongst others on what counts as a 'Match' versus 'Mismatch' in the case of dummy parents is provided in the help file (`?PedCompare`).

MergedPed This side-by-side comparison of the two pedigrees allows one to inspect any mismatches and discrepancies between the two pedigrees. In addition to the parents in Ped1 ('dam.1' and 'sire.1') and Ped2 ('dam.2' and 'sire.2'), it includes three columns ('id.r', 'dam.r', and 'sire.r') where dummy IDs in Pedigree 2 are replaced by the most likely non-genotyped individual from Pedigree 1. The value 'nomatch' in these columns indicates that there is no no-genotyped individual for which more than half of its offspring according to Ped1 has been assigned this dummy in Ped2. Note that this does include cases where a true sibship of say five individuals was split into one of three and one of two; the one of three is considered a match, and the smaller a mismatch — even though it can be argued the inferred pedigree does not contain any incorrect links.

ConcensusPed Here the merged pedigree is collapsed, with Pedigree 2 (here `Sequoia` assignments) taking priority over Pedigree 1, and dummy parents being replaced where known (using 'id.r', 'dam.r', and 'sire.r'). The columns 'dam.cat' and 'sire.cat' indicate with a 2-letter code whether the focal individual and the assigned parent were genotyped (G), a dummy individual in Pedigree 2 (D), a dummy individual replaced by a best-match non-genotyped individual from Pedigree 1 (R) or ungenotyped (U, and thus taken from Pedigree 1 only).

Example To increase the chance of mismatches, we simulate a genotype dataset with few SNPs, and pretend 20% of birth years and genders are unknown. The specific numbers will differ between simulated datasets, but the output structure will be the same.

```
> data(LH_HSg5, Ped_HSg5)
> GM <- SimGeno(Ped = Ped_HSg5, nSnp = 200, ErHQ = 1e-3)
> #
> LH <- LH_HSg5
> LH$BY[sample.int(nrow(LH), round(nrow(LH)*0.2))] <- NA
> LH$Sex[sample.int(nrow(LH), round(nrow(LH)*0.2))] <- NA
> #
> # run sequoia, with max 5 iterations of full pedigree reconstruction
> SeqX <- sequoia(GenoM = GM,  LifeHistData = LH, MaxSibIter = 5)
> #
> #check the number of mismatches in the full pedigree
> comp <- PedCompare(Ped1 = Ped_HSg5, Ped2 = SeqX$Pedigree)
> comp$Counts
> # , , dam
> #
> #    Total Match Mismatch P1only P2only
> # GG   529   522        4      3      0
> # GD   367   363        4      0      0
> # GT   892   885        4      3      0
> # DG    39    39        0      0      0
> # DD    29    29        0      0      0
> # DT    68    68        0      0      0
> # TT   961   953        4      3      1
> #
> # , , sire
> #
> #    Total Match Mismatch P1only P2only
> # GG   550   549        1      0      0
> # GD   343   337        5      1      0
> # GT   892   886        5      1      0
> # DG    38    38        0      0      0
> # DD    30    30        0      0      0
> # DT    68    68        0      0      0
> # TT   960   954        5      1      0
```

The errors are `Mismatch + P2only`, while `P1only` are the non-assigned parents

```
> # error rate:
> (4+1+5+0)/(2*960)
> #[1] 0.005208333
>
> # correct assignment rate
> (953+954)/(2*960)
> #[1] 0.9932292
```

We can investigate the mismatches further (in Rstudio, you can also use `View(comp$Mismatch)`):

```
> comp$Mismatch
>  #     id  dam.1 sire.1 dam.2 sire.2   id.r  dam.r  sire.r Cat Parent
>  # b05019 a04004 b04002 F0003  M0004 b05019 a04001  b04002  GG    dam
>  # b05018 a04004 b04002 F0003  M0004 b05018 a04001  b04002  GG    dam
>  # a05017 a04004 b04002 F0003  M0004 a05017 a04001  b04002  GG    dam
>  # b05020 a04004 b04002 F0003  M0004 b05020 a04001  b04002  GG    dam
>  # b05164 a04053 b04048 F0047  M0031 b05164 a04053 nomatch  GG   sire
>  # a05090 a04053 b04164 F0047  M0031 a05090 a04053 nomatch  GD   sire
>  # b05092 a04053 b04164 F0047  M0031 b05092 a04053 nomatch  GD   sire
>  # a05091 a04053 b04164 F0047  M0031 a05091 a04053 nomatch  GD   sire
>  # a04004 a03173 b03044 F0031  M0009 a04004 a03173  b03093  GD   sire
```

and split the mismatches by the three errors

dam a04004 vs F0003 The offspring of dam a04004 and sire b04002 in pedigree 1 are assigned the correct sire in pedigree 2, but apparently the wrong dam (F0003). We can gather some information about this dummy dam

```
> SeqX$DummyIDs[SeqX$DummyIDs$id=="F0003", ]
> #      id   dam  sire LLRdam LLRsire LLRpair sex BY.est BY.min BY.max NumOff     O1
> # 3 F0003 F0031 M0007   9.34   10.79    4.11   1      5      5      5     12 b05019
> #       O2     O3     O4     O5     O6     O7     O8     O9    O10    O11    O12
> # 3 a05017 b05173 a05174 b05175 a05176 b05037 b05038 b05040 a05039 b05020 b05018
```

based on its offspring (b05019, a05017, ...), `PedCompare` judges that this dummy female most likely is the non-genotyped individual a04001 (column dam.r in `comp$Mismatch`). A closer look at the true pedigree shows that this female is a full sibling of the true dam a04004

```
> Ped_HSg5[Ped_HSg5$id %in% c("a04001", "a04004", "b04002"), ]
> #         id    dam   sire
> # 617 a04001 a03173 b03044
> # 618 b04002 a03173 b03044
> # 620 a04004 a03173 b03044
```

Moreover, b05019 and its siblings are the result of a full-sib mating, further complicating the assignment.

sire b04165 vs M0031 We can have a look at the offspring assigned to dummy male M0031:

```
> PedM <- comp$MergedPed  # just to save typing
> #
> PedM[which(PedM$sire.2=="M0031"), ]
> #         id  dam.1 sire.1 dam.2 sire.2 id.r  dam.r  sire.r
> # 877 a05090 a04053 b04164 F0047  M0031 <NA> a04053 nomatch
> # 878 b05164 a04053 b04048 F0047  M0031 <NA> a04053 nomatch
> # 879 b05092 a04053 b04164 F0047  M0031 <NA> a04053 nomatch
> # 880 a05091 a04053 b04164 F0047  M0031 <NA> a04053 nomatch
```

and see that all but one (a05164, second row) share the same true sire b04164.

We can have a look if b040164 has more true offspring

```
> PedM[which(PedM$sire.1=="b04164"), ]
> #         id  dam.1 sire.1  dam.2 sire.2 id.r  dam.r  sire.r
> # 846 b05175 a04001 b04164  F0003  M0028 <NA> a04001  b04164
> # 847 a05166 a04122 b04164 a04122  M0028 <NA>   <NA>  b04164
> # 848 a05176 a04001 b04164  F0003  M0028 <NA> a04001  b04164
> # 849 a05089 a04053 b04164  F0047  M0028 <NA> a04053  b04164
> # 850 a05167 a04122 b04164 a04122  M0028 <NA>   <NA>  b04164
> # 851 a05174 a04001 b04164  F0003  M0028 <NA> a04001  b04164
> # 852 b05173 a04001 b04164  F0003  M0028 <NA> a04001  b04164
> # 853 b05165 a04122 b04164 a04122  M0028 <NA>   <NA>  b04164
> # 877 a05090 a04053 b04164  F0047  M0031 <NA> a04053 nomatch
> # 879 b05092 a04053 b04164  F0047  M0031 <NA> a04053 nomatch
> # 880 a05091 a04053 b04164  F0047  M0031 <NA> a04053 nomatch
> # 897 a05168 a04122 b04164   <NA>   <NA> <NA>   <NA>    <NA>
```

and see that his offspring are split across two sibships, M0028 and M0031, resulting in an Mismatch count equal to the size of the smaller of the two halves (here 3). One offspring (a05169) is not assigned a dam or sire in pedigree 2, contributing to the 'P1only' count.

Both the split and the non-assignment are most likely side effects of the mis-assignment of b04164 as full sibling rather than maternal half-sibling of a05090, b05092 and a05091, resulting in a mis-estimation of the most likely genotype of the non-genotyped shared father.

a04004 This individual was assigned M0009 as father (sire.2), which corresponds to non-genotyped male b03093 (sire.r), while its true father (sire.1) is b03044.

```
> PedM[which(PedM$dam.1=="a03173"), ]
> #         id  dam.1 sire.1 dam.2 sire.2   id.r  dam.r sire.r
> # 619 a04003 a03173 b03044 F0031  M0007   <NA> a03173 b03044
> # 636 b04080 a03173 b03093 F0031  M0009   <NA> a03173 b03093
> # 639 b04079 a03173 b03093 F0031  M0009   <NA> a03173 b03093
> # 640 a04004 a03173 b03044 F0031  M0009   <NA> a03173 b03093 <--
```

```
> # 643 a04078 a03173 b03093 F0031  M0009   <NA> a03173 b03093
> # 645 a04077 a03173 b03093 F0031  M0009   <NA> a03173 b03093
> # 968  M0004 a03173 b03044 F0031  M0007 b04002 a03173 b03044
> # 970  F0003 a03173 b03044 F0031  M0007 a04001 a03173 b03044
```

Thus, a04004's mother mated with both b03044 and b03093, and a04004 got clustered with the wrong full sibling group (but the correct maternal half-siblings).

4.1.1 Dyads

If you only care if pairs of individuals are 'full sibs', 'half sibs' or 'other', you can use `dyadcompare`

```
> DyadCompare(Ped_HSg5, SeqX$PedigreePar)
> #     RC.2
> # RC.1     FS     HS      U
> #   FS    561    325    500
> #   HS      0   2131   2579
> #   U       0      0 416644
```

which here shows that no unrelated individuals (row U) are wrongly assigned as full (column FS) or half (HS) siblings, while many full sib pairs were left unassigned.

4.1.2 Colony

To compare Colony output with an existing pedigree, use:

```
> BestConfig <- read.table("Colony/file/file.BestConfig",
+                          header=T, sep="", comment.char="")
> PedCompare(PedFile1 = "ExistingPedigree.txt",
+            Ped2 = BestConfig)
```

4.2 Estimating confidence probabilities

The provided likelihood ratio between the assigned parent being the parent versus otherwise related to the focal individual, does not necessarily indicate how likely it is that the assignment is correct. Pedigree-wide confidence probabilities can, amongst others, be estimated by

- simulating genotype data according to the reconstructed (or an existing) pedigree, imposing realistic levels of missingness and genotyping errors;
- reconstructing a pedigree from these simulated data;
- counting the number of mismatches between the 'true' pedigree, used as input for the simulated data, and the pedigree reconstructed from the simulated data.

When repeated at least 10–20 times, the mean error count divided by the total number of pedigree links provides an estimate of one minus the the confidence probability. Note that this can be rather time consuming, and will give an anti-conservative estimate as the current simulations assume all SNPs are independent.

Since version 0.10, this process is conveniently wrapped in the function `EstConf`.

```
> data(SimGeno_example, LH_HSg5, package="sequoia")
> SeqOUT <- sequoia(GenoM = SimGeno_example[, 1:100],
+                    LifeHistData = LH_HSg5, MaxSibIter = 5)
> ConfPr <- EstConf(Ped = SeqOUT$PedigreePar,
+                   LifeHistData = LH_HSg5,
+                   Specs = SeqOUT$Specs, Full = TRUE,
+                   nSim = 3, ParMis = 0.4)
> # , , mean
> #      GG    GD    GT  DG  DD  DT    TT
> # dam   1 0.950 0.980 NaN NaN NaN 0.980
> # sire  1 0.986 0.995 NaN NaN NaN 0.995
> #
> # , , min
> #      GG    GD    GT  DG  DD  DT    TT
> # dam   1 0.896 0.957 NaN NaN NaN 0.957
> # sire  1 0.957 0.986 NaN NaN NaN 0.986
>
```

The second set of confidence probabilities ('min') is calculated using the maximum number of errors in a simulation, rather than the average number.

To add confidence probability to the pedigree based on real data, assuming that replacement of dummies by IDs of non-genotyped individuals is free from error,

```
> PedC <- PedCompare(Ped1 = Ped_HSg5,
+                    Ped2 = SeqOUT$Pedigree)$ConsensusPed
> ConfProb <- cbind(ConfPr[,,"mean"],
+                   "U" = NA,  # Ungenotyped, parent taken from Ped1
+                   "X" = NA)  # no parent in either pedigree
> PedC$dam.cat2 <- PedC$dam.cat
> PedC$dam.cat2[PedC$dam.cat == "GR"] <- "GD"
> PedC$dam.cat2[PedC$dam.cat == "RG"] <- "DG"
> PedC$dam.cat2[PedC$dam.cat %in% c("DD", "DR", "RD", "RR")] <- "DD"
> PedC$dam.prob <- ConfProb["dam", as.character(PedC$dam.cat2)]
>
> # and analogously for sires.
```

4.3 Comparison pedigree-based and genomic relatedness

In absence of a previous pedigree, or when it is not obvious whether the previous or newly inferred pedigree is correct, one can compare the pairwise relatedness estimated from the pedigrees 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, `http://cnsgenomics.com/software/gcta/#MakingaGRM`, while pedigree relatedness can be calculated for example using the R package `pedantics`. Genomic relatedness will vary around the pedigree-based relatedness even for a perfect pedigree due to Mendelian variance, but outliers suggest pedigree errors.

As the number of pairs p becomes very large even for moderate numbers of individuals n ($p = n \times (n-1)/2$), additional packages are required to assist with merging (`data.table`) and plotting (`hexbinplot`). For example:

```
> 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("IID2", "IID1", "SNPS", "R.SNP")
> Rel.snp <- Rel.snp[Rel.snp$IID1 != Rel.snp$IID2,]
> #
> library(pedantics)
> PedStats <- pedigreeStats(SeqOUT$Pedigree[,1:3], graphicalReport=FALSE,
+                           includeA=TRUE)
> Rel.ped <- as.data.frame.table(PedStats$Amatrix)
> names(Rel.ped) <- c("IID1", "IID2", "R.seq")
> #
> library(data.table)
> Rel.snp <- data.table(Rel.snp, key=c("IID1", "IID2"))
> Rel.ped <- data.table(Rel.ped, key=c("IID1", "IID2"))
> Rel.gt <- merge(Rel.snp[,c(1,2,4)], Rel.ped, all.x=TRUE)
> Rel.gt <- as.data.frame(Rel.gt)
> rm(PedStats, Rel.snp, Rel.ped)
> #
> round(cor(Rel.gt[, 3:4], use="pairwise.complete"),4)
> #
> library(hexbin)
> ColF <- function(n) rev(rainbow(n, start=0, end=4/6,
+                                 s=seq(.9,.6,length.out=n),v=.8))
> hexbinplot(Rel.gt$R.SNP~Rel.gt$R.ped, xbins=100, aspect=1, maxcnt=10^6.5,
+            trans=log10,inv=function(x) 10^x, colorcut=seq(0,1,length=14),
+            xlab="Pedigree relatedness", ylab="Genomic relatedness",
+            xlim=c(-.1,.9), ylim=c(-.1, .9), colramp=ColF, colorkey = TRUE)
```

5 Other

5.1 Unusual relationships

Pedigree inference is often applied in small, (semi-)closed populations, and regularly to test 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. A range of such double relationships is considered explicitly (Table 6) 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 an email to `jisca.huisman@gmail.com` as adding additional relationships is relatively straightforward.

Table 6: Double relationships between pairs of individuals; – = impossible, Y = explicitly considered, empty = not (yet) explicitly considered (but possible to be inferred in two steps). 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
FS	–	–	–	–	–	Y		–	Y	Y
HS	Y	–	(FS)	Y	Y	Y[2]				Y
GP	Y	–	Y	[1]						Y
FA		–				Y				Y
HA		Y	Y[2]							Y
F1C										Y
GGG		–					[3]			Y

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

5.2 Hermaphrodites

For *parentage assignment only*, hermaphrodites can be specified in `LifeHistData` with sex '4', which then 'under the hood' are threated as two individuals with opposite sex, and identical genotypes. When running as usual, and all candidate parents are hermaphrodites, no parents will be assigned at all, as the configuration where A is the dam and B the sire is as likely as B being the dam and A the sire. Likely parent-offspring pairs can be found in 'MaybeParent', and parent-parent-offspring trios in 'MaybeTrio'.

To chose between A being the dam or the sire, additional information is required on the probable dam. This can e.g. the plant from which the seed was collected. This 'prior' information is not (yet) fully integrated, but rather, only when two configurations are equally likely, is the one that matches the pedigree prior chosen. Parent–offspring pairs in the pedigree prior that are not genetically a match will never be assigned.

```
> cdam <- read.table("Candidate_dams.txt", header=TRUE, stringsAsFactors=FALSE)
> # cdam has columns 'id' and 'dam', and does not need entries for all ids
> seq.euc2 <- sequoia(GenoM = Geno, LifeHistData = LH_X, MaxSibIter=0,
+                     SeqList = list(PedigreePar = cbind(cdam, sire=NA)))
```

5.3 Cluster families

Certain analyses, such as the Mendelian error check in PLINK, are done on a family-by-family basis. The function `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`, and then split afterwards using `PedStripFID`.

5.4 Pedigree stats & plots

A table with basic pedigree summary statistics can for example be generated using `pedigreeStats` from library pedantics. This package can also draw the pedigree, using `drawPedigree`.

There is a range of software available to plot pedigrees in various styles, such as for example PedigreeViewer. Be aware that many use a different column order (id - sire - dam, instead of id - dam - sire used here), and often use '0' to denote missing parents, rather than NA.

References

[1] J Huisman. Pedigree reconstruction using snp data: parentage assignment, sibship clustering, and beyond. *Molecular Ecology Resources*, 17(5):1009-1024.

[2] T C Marshall, J B K E Slate, L E B Kruuk, and J M Pemberton. Statistical confidence for likelihood-based paternity inference in natural populations. *Molecular ecology*, 7(5):639–655, 1998.

[3] S Purcell, B Neale, K Todd-Brown, L Thomas, M A R Ferreira, D Bender, J Maller, P Sklar, PIW De Bakker, MJ Daly, et al. PLINK: a tool set for whole-genome association and population-based linkage analyses. *The American Journal of Human Genetics*, 81(3):559–575, 2007.

[4] E A Thompson and T R Meagher. Parental and sib likelihoods in genealogy reconstruction. *Biometrics*, pages 585–600, 1987.