Tutorial For Using `related`

Last Updated: March 7, 2014

by

Tim Frasier
(timothy.frasier@smu.ca, www.frasierlab.ca)

Contents

1 Brief Description

`related` is an R package that allows users to estimate pairwise relatedness for pairs of individuals based on codominant molecular markers (microsatellites, SNPs, etc.), and also has simulation capabilities for comparing the performance of different estimators and for testing the resolution of a data set. Relatedness can be estimated using any of seven different methods, and can incorporate inbreeding and genotyping errors. The underlying code for implementing these relatedness estimators is Jinliang Wang's Fortran code for his COANCESTRY program (Wang 2011), and we are thankful to Dr. Wang for allowing us to use his code for this package.

The simulation function, which is new in `related` and is different than that implemented in COANCESTRY, allows users to use an allele frequency file to generate simulated individuals of known relatedness (parent-offspring, full–sibs, half–sibs, and unrelated). The rationale for providing this functionality are two-fold. First, it is well known that some relatedness estimators perform better than others, but this varies depending on the characteristics of the data set (Van de Casteele *et al.* 2001; Csilléry *et al.* 2006; Wang 2011). Thus, users can generate simulated individuals from their allele frequency file, conduct relatedness analyses on those individuals, and then assess which estimator will perform best with their data set.

Secondly, it is often difficult to know what sort of resolution to expect from a data set (e.g. how much differentiation to expect between full- and half-sibs?). Our simulation function provides users with a means to clearly test and visualize what sort of resolution is obtainable with their data.

2 Getting Data Into R In Correctly Formatted Form

2.1 Format

One source of headaches for population geneticists is the various formats of infiles that are required by different analysis programs. Therefore, we have tried to make the required format, and the formatting process, as easy and intuitive as possible.

The format of the genotype file required by `related` is simply one column of individual identifiers, followed by the genotype data, for which there are two columns for each locus (one for each allele). The general rules for the format of the genotype file are:

1. It should be a text file (not an Excel file);
2. It should be space– or tab–delimited;
3. Missing data must be represented by zeros (0); and
4. There should not be a row of column names in the genotype file.

The example genotype file provided with the package (GenotypeData) can be viewed as an example. The first few lines of that file are represented below. Briefly, each row contains data for one individual. The first column contains text representing individual identifiers for each individual. The second and third columns contain the two alleles for the first locus, columns 4 and 5 contain the two alleles for the second locus, and so on.

AA_n001aripo	241	241	150	150	125	131	...
AA_n002aripo	143	169	198	198	131	131	...
AA_n003aripo	152	169	178	202	131	131	...
...							

Note: if you want to analyze relatedness values based on pre-defined groups (e.g. compare relatedness within versus among groups), then the first two (2) characters of each individual ID should represent the labels for each group. If you look at the example infile (GenotypeData) you will see that this has been done, with the group identifiers being AA, AB, AC, and so on. See section 3.5 for more information.

2.2 Getting Data Into R

There are three ways with which you can interact with your genotype data within `related`:

1. Directly analyze the data via the text file;
2. Use our import function; and
3. Import and format it yourself.

2.2.1 Directly Analyze Text File

First, for calculating relatedness estimates, you can just provide `related` with the name of your genotype file, which needs to be in the format discussed in section 2.1. For this, the genotype file needs to be in R's working directory on your computer (i.e, it must be in the folder where R knows to look for it). Otherwise, you will need to provide the path to your file as well.

For example, if a user wants to get point estimates using Queller & Goodnight's estimator (more on the different relatedness functions later), and their data file is called 'GenotypeData.txt', then the appropriate command would be:

```
> outfile <- coancestry("GenotypeData.txt", quellergt=1)
```

The function for estimating relatedness is called `coancestry`, and here the first argument is the name of the genotype file, in quotes. The second argument indicates that we want point estimates of relatedness using the Queller & Goodnight estimator (more details on the different estimators, and associated options, are in section 3).

2.2.2 Use Our Import Function

To try to make life easier, we have written a function that will read your data into an R data frame in a way that will allow `related` to access it. The function is called `readgenotypedata`. This will create a data frame with your genotype data, and will also create an allele frequency data frame that can be used for other analyses. For example, if your genotype data are in a file called "GenotypeData.txt", then this command would be:

```
> input <- readgenotypedata("GenotypeData.txt")
```

This will generate a few data frames, all of which are named as an extension to the filename that you specified ("input" in this case), and are read directly from the data (you don't have to enter any information):

`input$gdata`	The data frame containing your genotype data. The first column is character data, and the remaining columns are all integers. This is the same format as the genotype file shown in section 2.1.
`input$nloci`	An integer containing the number of loci used.
`input$nalleles`	A series of integers specifying the number of alleles at each locus.
`input$ninds`	An integer containing the number of individuals in the genotype file.
`input$freqs`	The allele frequency data, which will be useful in other analyses.

You could then conduct the same analysis as above (get point estimates using Queller & Goodnight's estimator) using the following command:

```
> outfile <- coancestry(input$gdata, quellergt=1)
```

Here, the first argument of the function refers to the data frame containing the genotype data, and the second argument indicates that point estimates using the Queller & Goodnight estimator are to be calculated. The output for this function call is assigned to an R data frame called 'outfile'.

2.2.3 Read in Data Manually

Some users that are familiar with R may prefer to do things their own way. This is possible too.

The general way to read data frame-like data into R is using the R function `read.table`. With this, you have to specify at least two characteristics of your data file: (1) whether or not there is header information (names for each of your fields/columns); and (2) what the different fields (columns) are delimited by (here they should be either tabs ("\t") or space (" "). However, if you do this, R will automatically assign the data type 'Factor' to the individual IDs, rather than character (all genotypes should be integers, which is correct). To change this, you have to add `stringsAsFactors = FALSE` as an argument to the `read.table` function. The full command would be:

```
> input <- read.table("GenotypeData.txt", header=FALSE, sep=" ", stringsAsFactors=
    FALSE)
```

You can then conduct analyses on this the same way as above (note that with this method you will not get all of the other associated data, such as number of loci, frequencies, etc.):

```
> outfile <- coancestry(input, quellergt=1)
```

3 Obtaining Estimates of Relatedness

The underlying code for estimating relatedness is the Fortran code for the program COANCESTRY (Wang 2011), and therefore all of the options for that program are available here. We will briefly describe them here, but users should refer to the COANCESTRY paper itself (Wang 2011), as well as the manual for that program for more details regarding the specifics of the calculations.

Briefly, `related` can estimate pairwise relatedness using any of seven possible estimators. These estimators, the command used to refer to them, and the appropriate reference, are listed below.

Command	Name and/or Reference
dyadml	dyadic likelihood estimator, Milligan (2003)
lynchli	Li *et al.* (1993)
lynchrd	Lynch and Ritland (1999)
quellergt	Queller and Goodnight (1989)
ritland	Ritland (1996)
trioml	triadic likelihood estimator, Wang (2007)
wang	Wang (2002)

3.1 Point Estimates of Relatedness

To obtain point estimates of relatedness, use the command names corresponding to the estimator(s) you want to use and set them equal to 1. The "1" tells the program to just calculate point estimates (rather than point estimates *and* 95% confidence intervals). You do not need to have an argument for all estimators: if they aren't listed as arguments to the function call, then it is assumed that you don't want to estimate them.

3.1.1 Non-likelihood Estimators

To obtain relatedness estimates using the 5 non-likelihood methods (`lynchli, lynchrd, quellergt, ritland, wang`), the command would be:

```
> output <- coancestry("GenotypeData.txt", lynchli=1, lynchrd=1, quellergt=1, ritland
    =1, wang=1)
```

There will be 5 resulting data frames, all of which are named as an extension to the data frame name for the output of the function call that you specified ("output" in this case). These data frames remain the same, and have the same structure, regardless of which estimators you choose to estimate. They are:

`output$relatedness`	A data frame containing all pairwise estimates of relatedness. This will always have 11 columns: (1) an integer for the pair number; (2) the ID for individual #1; (3) the ID for individual #2; (4) the group assignment (see section 3.5); (5 - 11) for the 7 relatedness estimators—contain values of 0 for estimators not chosen.
`output$delta7`	A data frame that contains the Δ_7 estimates for the relatedness estimators that use it (trioml, wang, lynchrd, dyadml). This data frame contains one row for each pairwise comparison, and 8 columns: (1) an integer for the pair number; (2) the ID for individual #1; (3) the ID for individuals #2; (4) the group assignment (See section 3.5); and (5 - 8) estimates of Δ_7 for the 4 relevant estimators (trioml, wang, lynchrd, dyadml), with values of 0 for estimators not chosen.
`output$delta8`	A data frame that contains the Δ_8 estimates for the relatedness estimators that use it (trioml, wang, lynchrd, dyadml). This data frame contains one row for each pairwise comparison, and 8 columns: (1) an integer for the pair number; (2) the ID for individual #1; (3) the ID for individuals #2; (4) the group assignment (See section 3.5); and (5 - 8) estimates of Δ_7 for the 4 relevant estimators (trioml, wang, lynchrd, dyadml), with values of 0 for estimators not chosen.
`output$inbreeding`	A data frame that contains the inbreeding estimates for each individual, as used in the relatedness estimators. Only four of the relatedness estimators can account for inbreeding: dyadml, lynchrd, ritland, trioml. This data frame contains one row for each individual, and 5 columns: (1) individual ID; (2-5) inbreeding estimates for the four relatedness estimators. Estimators not used will have a zero (0) in the corresponding column.

3.1.2 Likelihood Estimators

Note that the likelihood estimators are **much** slower than the others!

Obtaining point estimates for the likelihood estimators (dyadml and trioml) follows the same approach as with the non-likelihood estimators. However, the triadic likelihood estimator (**trioml**) requires that you specify the number of reference individuals to use for estimating relatedness. The default is 100, and therefore if you want to use 100 then you do not need to specify a value. However, if you want to change this (say to 150), then you would have to add the argument `trioml.num.reference = 150` to the function call.

For example, to obtain point estimates for both likelihood estimators, and the wang estimator, but use 150 reference individuals for the trioml estimator, the command would be:

```
> output <- coancestry("GenotypeData.txt", dyadml=1, trioml=1, wang=1, trioml.num.
    reference=150)
```

3.2 Point Estimates of Relatedness With 95% Confidence Intervals

`related` can estimate 95% confidence intervals for relatedness estimates based on bootstrapping over loci. This option can be selected by setting the appropriate relatedness estimator to `2`. For example, if you want to estimate relatedness using the Queller and Goodnight (1989), Lynch and Ritland (1999), and Wang (2002) estimators, but also want 95% confidence intervals for the Wang (2002) estimator, the command would be:

```
> output <- coancestry("GenotypeData.txt", quellergt=1, lynchrd=1, wang=2)
```

Or, if you want 95% confidence intervals for all three, then the command would be:

```
> output <- coancestry("GenotypeData.txt", quellergt=2, lynchrd=2, wang=2)
```

The default number of bootstrap replicates is set to 100. However, this can be changed using the `ci95.num.bootstrap` argument in the `coancestry` command. For example, if you want to change the number of bootstrap events to 500, the command would be:

```
> output <- coancestry("GenotypeData.txt", quellergt=2, lynchrd=2, wang=2, ci95.num.
    bootstrap=500)
```

In addition to the output data frames described in section 3.1.1, setting the estimator argument to 2 generates the additional output data frames, which are also named as an extension to the name that you specified ("output" in this case).

`output$relatedness.ci95` A data frame containing the low and high cut-off values for the 95% confidence interval associated with each chosen estimator. This will always have 18 columns: (1) an integer for the pair number; (2) the ID for individual #1; (3) the ID for individual #2; (4) the group assignment (see section 3.5); (5 - 18) for the high and low values associated with each of the 7 relatedness estimators—contain values of 0 for estimators not chosen.

`output$delta7.ci95` A data frame that contains the low and high cut-off values for the 95% confidence interval for the Δ_7 estimates associated with each chosen estimator that use it (trioml, wang, lynchrd, dyadml). This will always have 12 columns: (1) an integer for the pair number; (2) the ID for individual #1; (3) the ID for individual #2; (4) the group assignment (see section 3.5); (5 - 12) for the high and low values associated with each of the 7 relatedness estimators—contain values of 0 for estimators not chosen.

`output$delta8.ci95` A data frame that contains the low and high cut-off values for the 95% confidence interval for the Δ_8 estimates associated with each chosen estimator that use it (trioml, wang, lynchrd, dyadml). This will always have 12 columns: (1) an integer for the pair number; (2) the ID for individual #1; (3) the ID for individual #2; (4) the group assignment (see section 3.5); (5 - 12) for the high and low values associated with each of the 7 relatedness estimators—contain values of 0 for estimators not chosen.

`output$inbreeding.ci95` A data frame that contains the low and high cut-off values for the 95% confidence interval for the inbreeding estimates for each individual, as used in the relatedness estimators. Only four of the relatedness estimators can account for inbreeding: dyadml, lynchrd, ritland, trioml. This data frame contains one row for each individual, and 9 columns: (1) individual ID; (2-9) inbreeding estimates for the four relatedness estimators. Estimators not used will have a zero (0) in the corresponding column.

3.3 Including Genotyping Error Rates

To account for genotyping error rates, the `error.rates` argument can be added to the initial call of the function. A single value can be entered if the error rates are the same across all loci, or a different value can be entered for each locus. For example, to estimate relatedness using the Wang (2002) estimator, using an error rate of 0.01 for all loci, the command would be:

```
> output <- coancestry("GenotypeData.txt", error.rates=0.01, wang=2)
```

If different error rates are desired for different loci, then a vector can be made of the error rates for each locus, and then this vector can be referred to by the `error.rates` argument. An example is show below assuming genotypes at 10 loci. The "`c`" in the function below stands for "concatenate", which is an R command for concatenating values into a single variable.

```
> errors <- c(0.01, 0.02, 0.03, 0.04, 0.05, 0.06, 0.07, 0.08, 0.09, 0.10)
> output <- coancestry("GenotypeData.txt", error.rates=errors, wang=2)
```

Note that including error rate estimates does not change point estimates of relatedness, just the confidence intervals.

3.4 Accounting For Inbreeding

The two likelihood methods, lynchrd, and ritland can account for inbreeding in their estimates of relatedness. This option can be selected using the `allow.inbreeding` argument. The default for this is FALSE, but it can be changed to TRUE if you want to account for inbreeding in your estimates. For example, to estimate relatedness using the dyadml estimator, and allow for inbreeding, while also calculating confidence intervals, the command would be:

```
> output <- coancestry("GenotypeData.txt", allow.inbreeding=TRUE, ci95.num.bootstrap
    =100, dyadml=2)
```

Note that accounting for inbreeding greatly increases the time needed for the program to run.

3.5 Analyzing Group Structure

The original COANCESTRY program has functionality for testing whether or not relatedness within identified (pre-defined) groups is higher than expected. This functionality is not explicitly available in `related`, but users familiar with R should still be able to set up such analyses. In the "relatedness" output of the `coancestry` function, there will be a column called "group". This is automatically generated based on the individual IDs included in the input file. Briefly, the `coancestry` function interprets the first two (2) characters of each Individual ID as a group assignment. The "group" field/column in the relatedness output contains the combination of these two characters for each pair of individuals for which relatedness is being estimated. Thus, this field/column can be used to parse and analyze relatedness estimates within and among groups. If you do not have group information, then this field will be meaningless, and can obviously be ignored.

4 Conducting Simulations

Simulations can be conducted using the `familysim` function. This function requires two arguments: (1) the name of the allele frequency data; and (2) the number of individuals to simulate. Both the `readgenotypedata` and `coancestry` functions generate allele frequency data in a format that can be used for simulations. Only

one number should be provided for the number of individuals to simulate, and this represents the number of pairs to generate for each degree of relatedness. For example, entering `100` would generate 100 parent-offspring pairs, 100 full-sib pairs, 100 half-sib pairs, and 100 unrelated pairs.

As an example, the commands for reading genotype data into R, and generating 100 simulated pairs of individuals of each degree of relatedness would be:

```
> input <- readgenotypedata("GenotypeData.txt")
> sim <- familysim(input$freqs, 100)
```

The result is a data frame of simulated genotypes that can be analyzed using the `coancestry` function. For example, to analyze these simulated data using the Wang (2002) estimator the code would be:

```
> output <- coancestry(sim, wang=1)
```

One issue with the way this works is that `coancestry` will calculate relatedness for **all** pairwise comparisons of the simulated individuals, whereas we are only interested in the relatedness values for the specific pairs that we generated of known relatedness. For example, for parent–offspring we are only interested in the relatedness of individuals 1 & 2, 3 & 4, 5 & 6, and so on. We don't care about the relatedness of 1 & 3, 1 & 4, etc. To deal with this, we have created a function called `cleanuprvals` that will remove the unwanted relatedness values. This function takes two arguments: (1) the name of the data frame containing relatedness values that will be "trimmed"; and (2) the number of individuals of each pair that were simulated. Thus, for this example the command would be:

```
> simrel <- cleanuprvals(output$relatedness, 100)
```

Now, you can analyze the relatedness values of these simulated individuals of known relatedness in whatever way is most informative for you and best suits your goals. For example, you may want to plot the distribution of relatedness values to see how well you can discriminate between individuals of different relatedness. There are two primary ways to do this: with box plots, or with density graphs. We'll walk through both of them.

4.1 Box Plots

Prior to plotting the data, it is helpful to clean the data up further. What we will do here is make new data frames consisting of two types of data: labels for each degree of relatedness, and the relatedness values. The Wang (2002) estimator is in the 6th column of the results file, so we can place that into a new list using the following command:

```
> relvalues <- simrel[, 6]
```

Using this sort of syntax in R, you can select specific rows and columns that you want to select from a given data frame. The row selections are indicated before the comma, and the column selections are indicated after the comma. If you don't include a number, this means "select all". Therefore, our command above selects all rows, and only the 6th column of the simrel data frame.

Now we need to add a label indicating the degree of relatedness of each pair. We can do this with the following commands:

```
> label1 <- rep("PO", 100)
> label2 <- rep("Full", 100)
> label3 <- rep("Half", 100)
> label4 <- rep("Unrelated", 100)
```

These commands make use of the "repeat" command in R, which is abbreviated to `rep`, which will generate a list containing the specified value repeated the specified number of times. So the first command is making a list, called "label1" that contains the text "PO" repeated 100 times, and so on. It is very important that these are made in this order, because this corresponds to the order of relatedness of the simulated pairs. We can now combine these into one long list using the following command:

```
> labels <- c(label1, label2, label3, label4)
```

The `c` in the above command stands for "concatenate", and so the above command is concatenating the four lists into one long list.

To plot a box plot, you have to tell R what list of data you want to group the data by, and this is called the `factor`. Here, we want to group the data by the labels indicating the degree of relatedness. We can specify which field to use for grouping values by with the `as.factor` function. The command is below (results in **Figure 1**).

```
> plot(as.factor(labels), relvalues, ylab="Relatedness Value", xlab="Relatedness")
```

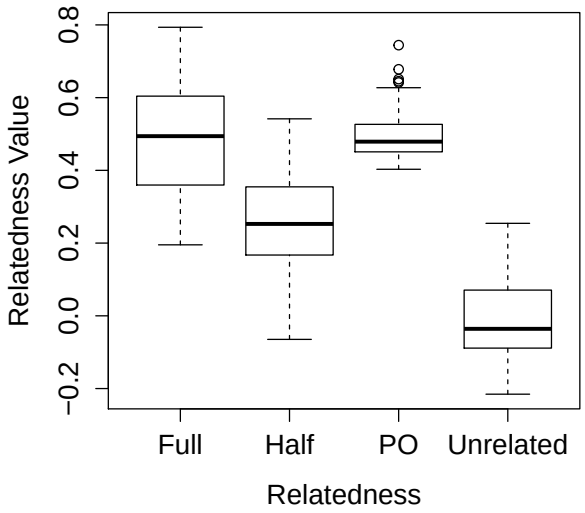

Figure 1: Box plot of relatedness values for simulated pairs of known relatedness using the `plot` function.

This looks good, but I prefer the plots that are generated with the **ggplot2** package, is installed automatically when you load `related`. To generate a similar plot, use the following commands (see **Figure 2**):

```
> qplot(as.factor(labels), relvalues, geom="boxplot", ylab="Relatedness Values", xlab="
    Relatedness")
```

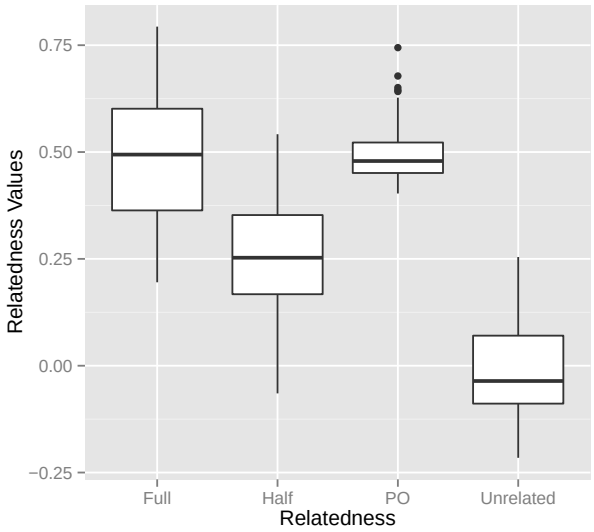

Figure 2: Box plot of relatedness values for simulated pairs of known relatedness using ggplot2.

4.2 Density Plots

Another informative way to assess the overlap between relatedness values of individuals of different relatedness is to create density plots representing histograms of the relatedness values. This is also easiest to do with the `ggplot2` package.

We will need to do a few things to get the data ready for plotting. First, let's rename the list of labels, because this name will be used in the legend of our graph.

```
> Relationship <- labels
```

Next, we need to combine the `Relationship` and `relvalues` vectors into a single data frame. We can do this with the `cbind` function in R, which is for binding columns together. We also want to make sure the resulting object is saved as a data frame.

```
> newdata <- as.data.frame(cbind(Relationship, relvalues))
```

The above command codes the `relvalue` data in the wrong format, and we need to correct that.

```
> newdata$relvalues <- as.numeric(as.character(newdata$relvalues))
```

Now we can plot a density plot of the relatedness values for each type of relationship (see **Figure 3**).

```
> qplot(relvalues, ..density.., data=newdata, geom="density", colour=as.factor(
    Relationship), xlab="Relatedness Value", ylab="Density")
```

4.3 Comparing Relatedness Estimators

Prior to deciding which relatedness estimator to use for your analysis, it is a good idea to test the performance of the different estimators on simulated data sets with the same locus characteristics as your own. We

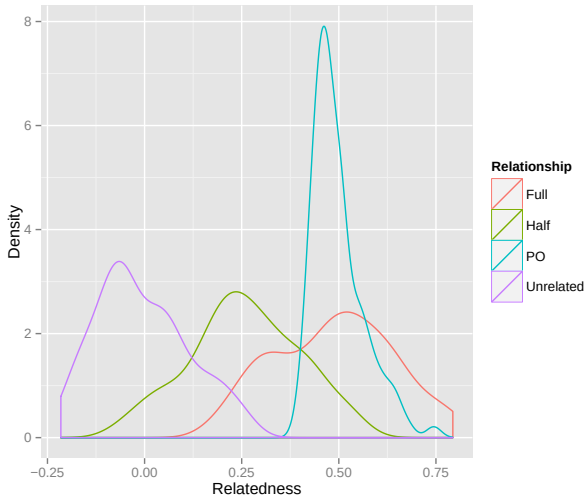

Figure 3: Density plot of relatedness values for simulated pairs of known relatedness using ggplot2.

have created a function that will automate this process for you, called `compareestimators`. This function takes two arguments: (1) the name of the data frame containing your genotype data; and (2) the number of simulated pairs of individuals you want for each type of relationship. This function will generate simulated genotypes of known relatedness, calculate relatedness values using 4 of the moment-based estimators, and then plot the data for your visualization. The Ritland (1996) estimator is not included in this comparison because we found that the variation around these estimates were vastly larger than for the other estimators, which made the required scales for the estimators very different, and therefore compromised the usefulness of the subsequent graphs.

For example, using the example file the commands would be:

```
> input <- readgenotypedata("GenotypeData.txt")
> compareestimators(input, 100)
```

The output will be a graph of the data, organized by relationship type and estimator (see **Figure 4**). Note that it will take a while to do all of the analyses and show the graph. Be patient!

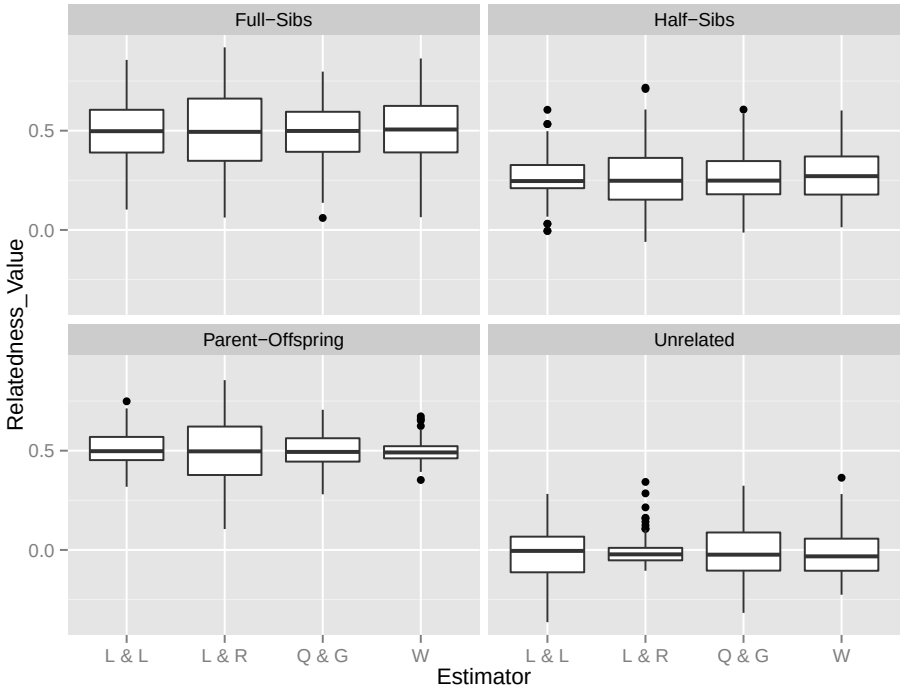

Figure 4: Plot comparing the relatedness estimates using 4 different estimators, and simulated individuals of known relatedness. "L & L" is the lynchli estimator, "L & R" is the lynchrd estimator, "Q & G" is the quellergt estimator, and "W" is the wang estimator.

References

Csilléry K, Johnson T, Beraldi D, Clutton-Brock T, Coltman D, Hansson B, *et al.* (2006). Performance of marker-based relatedness estimators in natural populations of outbred vertebrates. *Genetics* **173**: 2091–2101.

Li CC, Weeks DE, Chakravarti A (1993). Similarity of DNA fingerprints due to chance and relatedness. *Human Heredity* **43**: 45–52.

Lynch M, Ritland K (1999). Estimation of pairwise relatedness with molecular markers. *Genetics* **152**: 1753–1766.

Milligan BG (2003). Maximum-likelihood estimation of relatedness. *Genetics* **163**: 1153–1167.

Queller DC, Goodnight KF (1989). Estimating relatedness using molecular markers. *Evolution* **43**: 258–275.

Ritland K (1996). Estimators for pairwise relatedness and inbreeding coefficients. *Genetical research* **67**: 175–186.

Van de Casteele T, Galbusera P, Matthysen E (2001). A comparison of microsatellite-based pairwise relatedness estimators. *Molecular Ecology* **10**: 1539–1549.

Wang J (2002). An estimator for pairwise relatedness using molecular markers. *Genetics* **160**: 1203–1215.

Wang J (2007). Triadic IBD coefficients and applications to estimating pairwise relatedness. *Genetical research* **89**: 135–153.

Wang J (2011). COANCESTRY: a program for simulating, estimating and analysing relatedness and inbreeding coefficients. *Molecular Ecology Resources* **11**: 141–145.