The allelematch package for R: A detailed introduction and tutorial SUPPLEMENTARY DOCUMENTATION

P. Galpern[1], M. Manseau[1,2], P. Hettinga[3], K. Smith[4], and P. Wilson[4]

[1]*Natural Resources Institute, University of Manitoba*
[2]*Parks Canada*
[3]*Wildlife Resources Consulting Service Manitoba*
[4]*Natural Resources DNA Profiling and Forensic Centre, Trent University*

Contents

1 Introduction

This document is a supplement to the publication describing the allelematch package, and is also included as an R vignette. Here, we describe the operation of the package in more detail and illustrate its use in a tutorial format.

The allelematch package was developed to identify unique genotype profiles in situations where there are likely to be multiple samples of each individual present. Such conditions arise in wildlife conservation genetics where non-invasive sampling of hair and fecal material can produce large datasets containing an unknown number of animals. This package and the examples that follow are directed primarily towards this audience.

Finding unique individuals appears on the surface to be a trivial exercise; a matter of sorting genotypes into identical groups. When there is unlikely to be error in profiling and missing data is absent, identity analysis is straightforward and software for this purpose has long been available (Kalinowski et al., 2007; Peakall et al., 2006; Wilberg & Dreher, 2004). The task becomes much more complicated if data sets contain missing information at some loci, or are subject to even a modest amount of genotyping error. In these cases samples from the same individual may differ because of allele dropouts or false alleles, or they may be identical except for some missing information. These issues make it ambiguous which samples belong to which individual, or how many individuals are present.

This is the niche for allelematch: applications of genotype profiling where identity must be established in suboptimal circumstances. In conservation genetics these conditions are common. Collecting high quality samples can often be challenging, and budgets may emphasize extensive sampling to address conservation questions rather than expanding the number of genetic markers, or reprofiling for accuracy and the elimination of missing data.

Allelematch may also be useful in other applications of identity analysis; for example, in wildlife forensics, where no amount of missing data or genotyping error can be tolerated. It can be used to assist in laboratory quality control, i.e. to visually highlight genotyping errors in data sets where the same individual is profiled multiple times for confirmation.

2 Approach

Allelematch operates by comparing rows of a multilocus genotype matrix, where rows are samples, and columns for each locus give the names of the alleles that are present. For diploid codominant data, there are typically two columns for each locus. It finds the similarities between the samples using a metric which is a form of the Hamming distance (Hamming, 1950) modified to account for missing data. It then uses hierarchical clustering and a dynamic method for identifying clusters on a dendrogram. This step is done using the Dynamic Tree Cut package for R (Langfelder et al., 2008) which defines groups of similar samples at a specified threshold of dissimilarity.

2.1 Algorithm

Specifically, amUnique() the workhorse function of the package performs the following operations:

1. Determine a similarity score for each pair of samples.

   Let A be a matrix $[a_{ij}]_{m\times n}$ with elements that are allele names or missing data, where m is the number of samples, and n is the number of columns containing the allele names.

   Find the similarity score,

   $$s_{p,q} = \frac{N_{match_{p,q}}}{n} + \frac{N_{missing_{p,q}}}{2n}, \forall p, q \in \{1, 2, \ldots, m\} \tag{1}$$

   where $N_{match_{p,q}}$ is the number of elements (alleles or missing data) that match in rows $a_{p,*}$ and $a_{q,*}$, and $N_{missing_{p,q}}$ is the number of elements that are missing in either row $a_{p,*}$ or row $a_{q,*}$.

   A score of $s_{p,q} = 1$ means that the elements of both rows $a_{p,*}$ and $a_{q,*}$ are identical. Scores less than one have been penalized $\frac{1}{n}$ for each mismatching element, and penalized a smaller amount, $\frac{1}{2n}$, where an element is missing in one row but not missing in the other row.

2. Produce a symmetric dissimilarity matrix from the similarity scores, $s_{p,q}$.

   $$D_{m,m} = 1 - \begin{bmatrix} 1 & s_{1,2} & \cdots & \cdots & s_{1,m} \\ s_{2,1} & 1 & \cdots & \cdots & s_{2,m} \\ \vdots & \vdots & \ddots & & \vdots \\ \vdots & \vdots & & \ddots & s_{m-1,m} \\ s_{m,1} & s_{m,2} & \cdots & s_{m,m-1} & 1 \end{bmatrix}$$

3. Perform agglomerative hierarchical clustering on the dissimilarity matrix, D, using the complete linkage method.

4. Use a hybrid dynamic tree cutting method (Langfelder et al., 2008) to identify clusters, where a cut height parameter $\hat{d}$ gives the maximum dissimilarity between any two samples in a cluster. This parameter is related to the similarity score as

   $$\hat{d} = 1 - \hat{s}$$

5. Find the *consensus* sample for each cluster, if cluster contains more than one sample at this $\hat{d}$. Allelematch provides two supported options[1] for declaring *consensus* samples:

   (a) The *consensus* is the sample that is most similar to other samples in the cluster (calculated using Equation 1)

   (b) The *consensus* is the sample with the least amount of missing data

[1] Options to interpolate missing data when identifying the consensus are currently experimental

6. Declare as *singletons* samples which are the only member of their cluster.
7. Produce a new matrix $A' = [a'_{ij}]_{m' \times n}$ consisting of the *consensus* samples if they exist, and the *singleton* samples.
8. IF A' contains only *singletons* continue to step 9. ELSE let $A = A'$ and return to step 1.
9. Declare as *unique* all rows of the matrix A'.
10. Compare *unique* matrix A' to all original samples. For each row $a'_{q,*}$ of the *unique* matrix A', find all rows $a_{p,*}$ of the original matrix A that have a similarity score $s_{p,q} \geq \hat{s}$, where s is calculated in an analagous manner to Equation 1. For example,

$$s_{p,q} = \frac{N_{match_{p,q}}}{n} + \frac{N_{missing_{p,q}}}{2n}, \forall p \in \{1, 2, \ldots, m\}, q \in \{1, 2, \ldots, m'\} \tag{2}$$

11. Declare as *matches* rows $a'_{q,*}$ that have a score $s_{p,q} \geq \hat{s}$ for exactly one row $a_{p,*}$.
12. Declare as *multiple matches* rows $a'_{q,*}$ that have a score $s_{p,q} \geq \hat{s}$ for two or more rows $a_{p,*}$.
13. Declare as *unclassified* rows $a'_{q,*}$ that do not have a score $s_{p,q} \geq \hat{s}$.

2.2 Notes

- One iteration of clustering and tree cutting can often be insufficient to group all samples that are similar at the given criterion $\hat{d}$. The algorithm therefore proceeds recursively until no more clusters are formed (i.e. step 8).
- An optimal criterion $\hat{d}$ for the data set can often be found by examining the behaviour of *multiple matches* at different values of the $\hat{d}$ parameter. The tutorial in this document explores the selection of this value in detail.
- The R package described in the remainder of this document refers to the alleleMismatch parameter $\hat{m}$, the maximum number of alleles that can mismatch, rather than using the $\hat{d}$ parameter. These values are related by

$$\hat{m} = n\hat{d}$$

Example Using $\hat{m} = 3$ implies that samples can differ by up to 3 mismatching alleles, or the equivalent in missing alleles (6), to be declared the same *unique* individual.

In practice the differences among samples will be a result of both mismatching and missing data. Therefore in general $\hat{m}$ implies

$$N_{mismatch_{p,q}} + \frac{N_{missing_{p,q}}}{2} \leq \hat{m}$$

for samples in rows $a_{p,*}$ and $a_{q,*}$ to be declared the same *unique* individual.

We work with the alleleMismatch criterion, $\hat{m}$ rather than its equivalent $\hat{d}$ for heuristic reasons; it is more familiar to think in terms of the equivalent number of alleles that mismatch rather than in terms of a dissimilarity metric.

- To help confirm unique identity, the match probability P_{sib} (Wilberg & Dreher, 2004) is calculated given the allele frequencies in the *unique* set of samples. It is calculated by default for all samples that are either *unique*, or match a *unique* sample without allele mismatches, and assesses the probability that two samples could match at their non-missing loci because they are siblings rather than duplicates. Optionally, it can be calculated for all samples by assuming that mismatching alleles are missing.
- *Unclassified* samples are generally those where their highest similarity score s with any other sample is slightly less than the criterion $\hat{s}$. They therefore occupy a grey area in which they are not different enough to be declared *unique*, but not similar enough to any *unique* sample to be declared *matches*. In practice there are few *unclassified* samples. If the analysis protocol requires the inclusion of these samples, a robust solution is to reprofile them to eliminate missing data, which may be contributing to their uncertain status.

3 Tutorial

The tutorial is organized around example data sets. Five examples review the use of allelematch for identifying samples representing unique individuals in data sets where these individuals may be sampled multiple times.

The tutorials provide sample code, and the graphical output from R has been inserted as figures. However, the most important output from allelematch is in the form of HTML documents. These illustrate in colour how samples match and mismatch. It was not possible to include this output within this document, however they can be viewed in a web browser. If you are reading this document on the screen it should be possible to click on the links marked "output" to view the HTML produced in each case. Note that within one example multiple HTML files may be generated and these are numbered sequentially (e.g. example1_1.html, example1_2.html, etc.) The HTML files are included in a ZIP file with this PDF, and should have been extracted to the same file location as the PDF you are viewing.

It is also possible to reproduce these examples directly in R. All example data sets are provided with the package and the tutorials demonstrate how to access this data.

3.1 Identifying unique individuals

Example 1 High quality data set

This data in this example are simulated[2] to represent a high quality data set that might result from a laboratory protocol where samples were run multiple times to confirm their identity. It has no genotyping error, a near-zero missing data load, and approximately 60% of the individuals have been artificially resampled more than once.

[2]Please see the publication associated with this package for details on how data were simulated.

In typical usage data would be imported into R in the most convenient manner. This may be the read.csv() function for example. In this and the following examples we load the data supplied with the package using the data() function.

```
> data(amExample1)
```

First, we create an amDataset object. This prepares the data set for use with other allelematch functions. The sample number will serve as the index. Missing data has been coded as "-99". For now, we'll exclude the column knownIndividual from our data set.

```
> example1 <- amDataset(amExample1, indexColumn = "sampleId",
+     ignoreColumn = "knownIndividual", missingCode = "-99")
```

The next step is to find the optimal criterion of dissimilarity to find the unique individuals. A convenient way to understand this criterion is in terms of the number of alleles that mismatch (alleleMismatch parameter; or $\hat{m}$ in section 2.2). This finds the number of unique individuals at a range of values for the parameter and attempts to declare an optimal value.

```
> amUniqueProfile(example1, doPlot = TRUE)
```

Figure 1 shows that an optimum for this data is allowing 2 alleles to mismatch (alleleMismatch=2). Note how multipleMatch, the number of samples that match more than one unique individual, is zero at the optimum. This tells us that every sample has been declared as a unique individual or as a match to a unique individual. This is described as a ZeroSecondMinimum profile, indicating that multipleMatch reaches zero at a parameter setting other than alleleMismatch=0. Simulations indicate that with this profile morphology the software can independently determine the alleleMismatch parameter required to identify the correct number of unique genotypes ($\pm 3\%$).[3]

The final step is to use the alleleMismatch criterion to identify which samples are unique and which samples are matches.

```
> uniqueExample1 <- amUnique(example1, alleleMismatch = 2)

allelematch:  assuming genotype columns are in pairs, representing 10 loci
```

This should be reviewed carefully before proceeding. The results can be viewed in an HTML format as follows. Click the link to view this output in a browser.

```
> summary(uniqueExample1, html = "example1_1.html")
```

In this example, 12 unique genotypes were identified among 20 samples. The alleleMismatch parameter we set at an earlier step allows up to two alleles to be different for samples to be declared identical. Examining the HTML output, there are no cases where mismatches occurred. The mismatching alleles would be highlighted in red. There is one case with missing data at one locus (unique genotype 12), and pink highlighting is used to indicate missing data. Recall how a missing allele in one genotype but not the other penalizes the score by $\frac{1}{2n}$ (by Equation 1). For this reason sample 19 (also unique genotype 12) matches itself with a score of 1, but matches sample 20 with a score of 0.95

[3]Please see the publication associated with this package for more information on how this was verified.

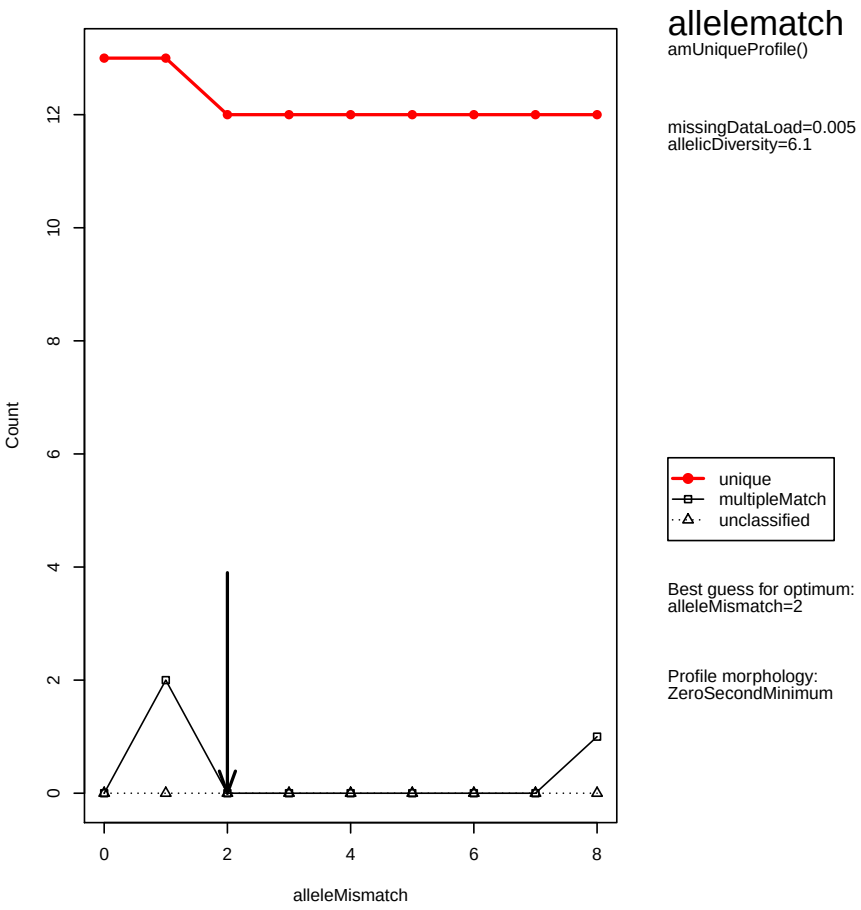

Figure 1: High quality data set

The same results can also be saved in a CSV spreadsheet format. This format lacks some of the supplementary information as well as the colour formatting that can be helpful for reviewing the results.

```
> summary(uniqueExample1, csv = "example1_1.csv")
```

An analysis-ready dataset can also be prepared from the CSV output. We encourage caution with this step. Typically the output of the analysis must be reviewed in detail before an analysis-ready data set can be used uncritically. For example, checking that P_{sib} probabilities are all below a threshold value is a minimum requirement as allelematch does not use this as a criterion.

```
> summary(uniqueExample1, csv = "example1_2.csv", uniqueOnly = TRUE)
```

When the multiple sampling was simulated in this data set we kept track of which individual each sample came from in the knownIndividual column. For typical non-invasive sampling applications, however, such information would not be known. We will repeat the analysis using this column as the meta-data to demonstrate that allelematch performed correctly.

```
> example1chk <- amDataset(amExample1, indexColumn = "sampleId",
+     metaDataColumn = "knownIndividual", missingCode = "-99")
> uniqueExample1chk <- amUnique(example1chk, alleleMismatch = 2)

allelematch:  assuming genotype columns are in pairs, representing 10 loci

> summary(uniqueExample1chk, html = "example1_2.html")
```

Click to view the output in HTML. Notice how the individual identifier (a three letter code) is consistent between the genotype declared unique and the genotypes that match it. Also note how each unique identifier appears only once in the list of unique genotypes, indicating that the analysis has not overestimated the number of individuals.

Example 2 Good quality data set

```
> data(amExample2)
```

The data in this example have also been simulated[4], this time to reflect the qualities of good quality data set, where genotyping error and missing data exist, but these can be confidently handled by allelematch without manual intervention. At each locus a random 4% of heterozygotes lost their second allele to simulate an allele dropout, and a random 4% of samples at each locus had alleles set to missing.

Create an amDataset object. The knownIndividual column is again kept as meta-data for instructional purposes.

```
> example2 <- amDataset(amExample2, indexColumn = "sampleId",
+     metaDataColumn = "knownIndividual", missingCode = "-99")
```

Find the optimal alleleMismatch parameter for this data set.

```
> amUniqueProfile(example2, doPlot = TRUE)
```

[4]Please see the publication associated with this package for details on how data were simulated.

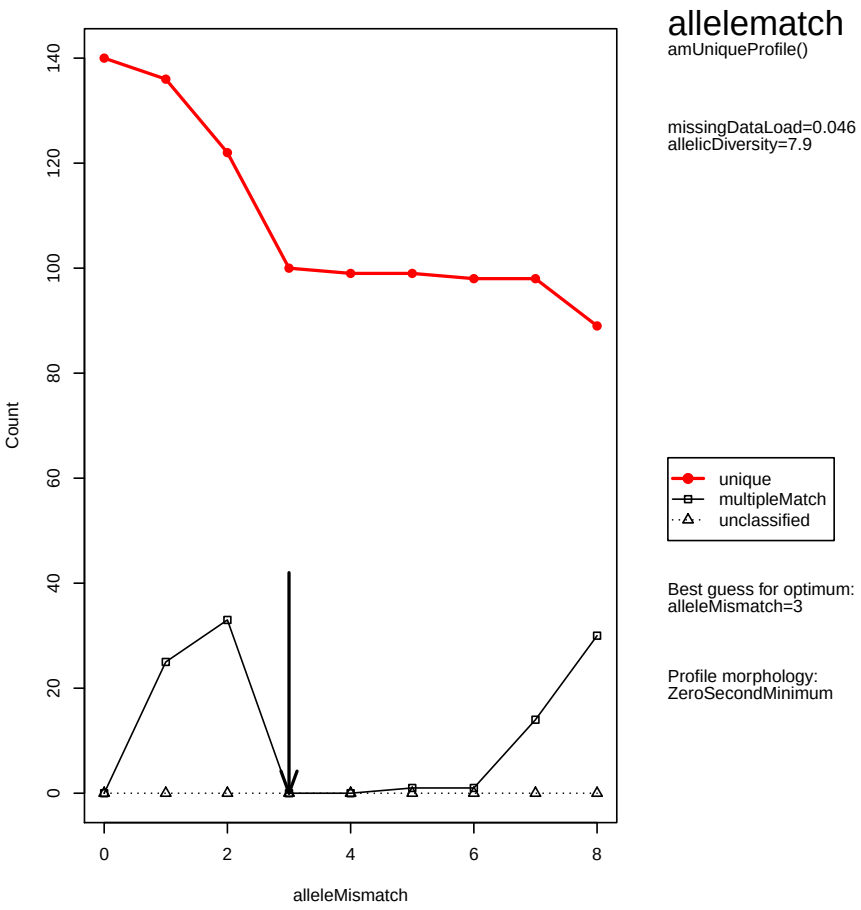

Figure 2: Good quality data set

Figure 2 demonstrates a ZeroSecondMinimum profile suggesting that the identified alleleMismatch parameter can be applied with confidence.

Conduct the unique analysis with the recommended alleleMismatch setting, and review the HTML output Click to view this output.

```
> uniqueExample2 <- amUnique(example2, alleleMismatch = 3)
```

```
allelematch:  assuming genotype columns are in pairs, representing 10 loci
```

```
> summary(uniqueExample2, html = "example2_1.html")
```

Allelematch identified 100 unique genotypes from these 148 samples, and did so with no errors (as illustrated by the knownIndividual identifier). There are a number of examples of allelematch correctly matching samples despite allele mismatches (the mismatching alleles are highlighted in red).

P_{sib} is not given in these mismatching cases by default because partial matching samples are not "identical". However, if we treat mismatching alleles as if they were missing, P_{sib} can be calculated for the non-missing loci as follows. Click to view this output.

```
> uniqueExample2 <- amUnique(example2, alleleMismatch = 3,
+     doPsib = "all")
```

```
allelematch:  assuming genotype columns are in pairs, representing 10 loci
```

```
> summary(uniqueExample2, html = "example2_2.html")
```

It is important to stress again that allelematch does not identify unique genotypes using the P_{sib} criterion, but rather presents this value to allow the user to assess the probability that: (1) matching samples represent siblings of unique genotypes rather than duplicate samples of the same individual; and (2) unique genotypes represent siblings of other unique genotypes.

Example 3 Marginal quality data set

```
> data(amExample3)
```

The data in this example have been simulated[5] to represent a data set of marginal quality where the use of allelematch combined with careful manual review of the results is required to achieve a confident assessment of the unique genotypes. At each locus a random 4% of heterozygotes lost their second allele to simulate an allele dropout, and a random 10% of samples at each locus had alleles set to missing.

Create an amDataset object. Once again we retain the knownIndividual for instructional purposes.

```
> example3 <- amDataset(amExample3, indexColumn = "sampleId",
+     metaDataColumn = "knownIndividual", missingCode = "-99")
```

Find the optimal alleleMismatch parameter for this data set.

```
> amUniqueProfile(example3, doPlot = TRUE)
```

[5]Please see the publication associated with this package for details on how data were simulated.

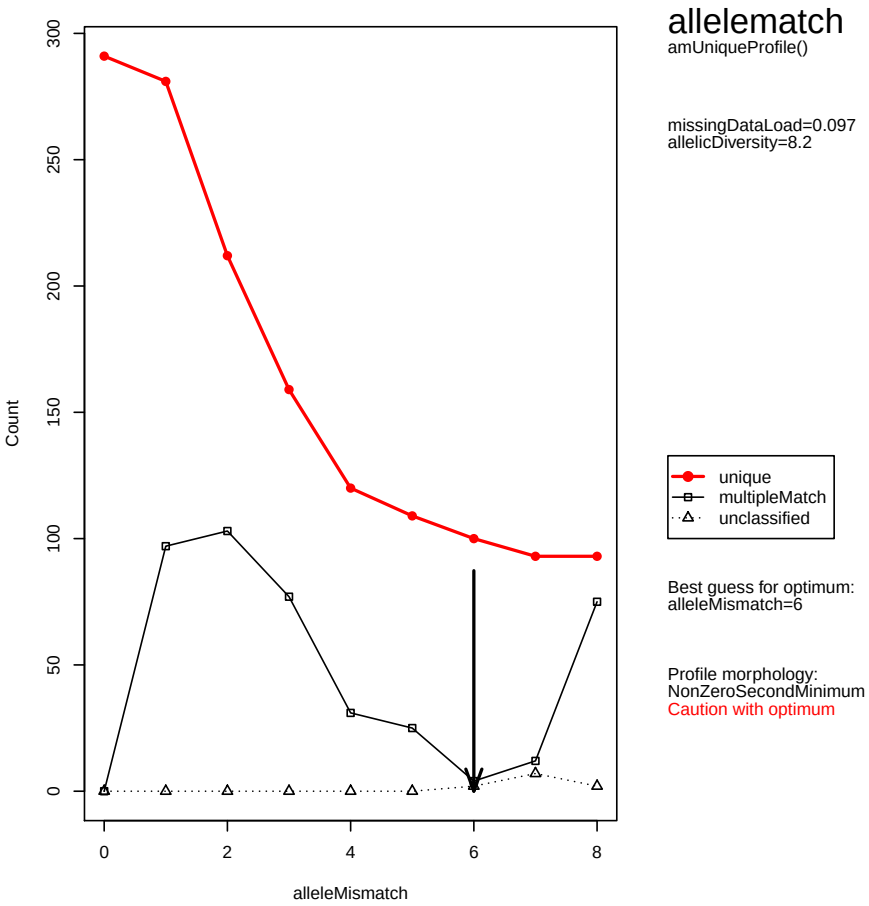

Figure 3: Marginal quality data set

Figure 3 demonstrates a NonZeroSecondMinimum profile, which is the first sign that we are dealing with a marginal data set. When this sort of profile is produced amUniqueProfile() can be somewhat error-prone in finding the optimal value for the alleleMismatch parameter, and our first concern is making sure that it chose the best value. We are looking for the second minimum in the multipleMatch variable, (the first minimum being at alleleMismatch=0), and the function appears to have identified this correctly at alleleMismatch=6.

Conduct the unique analysis with the recommended alleleMismatch setting, and review the HTML output. Click to view the output.

```
> uniqueExample3 <- amUnique(example3, alleleMismatch = 6)

allelematch:  assuming genotype columns are in pairs, representing 10 loci

> summary(uniqueExample3, html = "example3_1.html")
```

Two new issues appear in this example that did not exist for the previous two data sets. The header of the HTML output informs us that there are 2 unclassified samples and 4 multipleMatch samples. We will consider these in turn.

Unclassified samples are those which just exceed the criterion of similarity to be declared a match, but as an artifact of the dynamic tree cutting algorithm are not sufficiently different to be declared unique (i.e. to form their own clusters). Often this is because of missing data rather than mismatching data. If the analysis protocol does not permit their exclusion, the most robust approach is to reprofile these samples at their missing loci. However, this may not always be possible. Here we demonstrate an approach to simplify the manual classification of these samples.

The task is to determine whether these unclassified samples should be declared a unique genotype or as a match to an existing unique genotype. We use a function which conducts a pairwise analysis, comparing all the rows in data set of the first argument with all rows in the data set of the second argument and returning those that exceed a similarity score (i.e. by Equation 2). In this case the first data set consists of the two unclassified samples in the previous step, and the second all the samples declared as unique in the previous step. These data sets can be accessed from the uniqueExample3 object produced by amUnique().

```
> unclassifiedExample3 <- amPairwise(uniqueExample3$unclassified,
+     uniqueExample3$unique, alleleMismatch = 7)
```

Note how we used the next highest criterion, alleleMismatch=7, because we want to see if these samples are only slightly more different than existing unique genotypes. We can then examine this in HTML format.

```
> summary(unclassifiedExample3, html = "example3_2.html")
```

Using this output we must now make a judgement call about what types of evidence are sufficient for declaring differences among individuals. Here the rows with yellow highlighting are the unclassified samples and the rows without highlighting are unique genotypes. For the first unclassified sample it is mostly missing data and one mismatch that is driving the borderline status. If the five missing loci were reprofiled, it is likely that these two rows would match at a much lower criterion, and therefore sample 208

should be declared a match of sample 204. Thanks to the knownIndividual information retained from when that data were simulated we can confirm this supposition. Indeed, the two samples did come from the same individual, ACN. The second unclassified sample achieves its borderline status with four missing loci, but it also differs from its closest unique genotype because of three mismatching alleles. In this case it is more likely that sample 251 is a unique genotype. Again, this conclusion is supported by the knownIndividual column with samples representing individuals ADA and ABN.

MultipleMatch samples are those that match more than one unique genotype. These occur when samples do not all sort into clearly defined groups representing unique genotypes. As the profiling (e.g. Figure 3) shows, the numbers of these uncertain samples varies with the alleleMismatch criterion. If we set the criterion too low, unique genotypes will not be sufficiently different and samples will appear to match multiple unique individuals. If we set a high criterion it might not matter as long as unique genotypes are well differentiated (e.g. Figure 1; alleleMismatch=3 to alleleMismatch=7). However in cases where there is low allelic diversity and heterozygosity in the data set there could be insufficient information to distinguish unique genotypes when the criterion is set too high, and once again unique genotypes may be improperly determined resulting in samples that will match multiple unique genotypes.

We could examine the earlier output from the unique analysis to resolve these samples (the questionable genotypes are flagged CHECK_UNIQUE and the samples that cause this are flagged MULTIPLE_MATCH). However it is simpler to again use a pairwise approach to bring together just the relevant samples. Here we compare the four multipleMatch samples against the unique genotypes. The criterion is not changed this time from the original unique analysis.

```
> multipleMatchExample3 <- amPairwise(uniqueExample3$multipleMatch,
+     uniqueExample3$unique, alleleMismatch = 6)
> summary(multipleMatchExample3, html = "example3_3.html")
```

Examining the HTML output, sample 112 (yellow highlighting) matches two unique genotypes (unhighlighted). Sample 110 is very likely the same unique genotype as 112 because it differs mostly because of missing data, while sample 260 differs by both four mismatching alleles and missing data. In this case sample 112 can be declared a match of sample 110 and a false match of sample 260. Using the same argumentation sample 162 is likely a match of 161, and sample 218 a match of 213. This has resolved three of the four multipleMatch cases.

Sample 183 differs from two unique genotypes (samples 182 and 181) chiefly because of missing data. In this case, we should consider two unique genotype rows to be the same as the multipleMatch sample; we could think of sample 183 as an intermediate or "missing link" between two incorrectly desginated unique genotypes. As a result we must remove one of the unique genotypes. Because it has the least amount of missing data, let's declare sample 181 the unique genotype, and sample 183 and 182 matches of this unique genotype. Once again, we turn to the knownIndividual column for confirmation that we made the correct decisions for this simulated data. Indeed the correct decision was to reduce the total number of unique genotypes by one (individual ACF appears in two unique genotypes).

Reviewing the unclassified samples added sample 251 to the unique genotype list, and reviewing the multipleMatch samples removed sample 183 from the unique genotype list. Finally, we produce a CSV file of the original unique analysis and use spreadsheet software to make the necessary changes in classification.

```
> summary(uniqueExample3, csv = "example3_1.csv")
```

Example 4 Low quality data set

```
> data(amExample4)
```

For this example we have simulated[6] a low quality data set where uncertainty created by genotyping error and missing data, combined with a lack of information in the form of allelic diversity across loci will result in a low confidence assessment of the unique genotypes. At each locus a random 6% of heterozygotes lost their second allele to simulate an allele dropout, and a random 20% of samples at each locus had alleles set to missing.

Create an amDataset object. Again we retain the knownIndividual for instructional reasons.

```
> example4 <- amDataset(amExample4, indexColumn = "sampleId",
+     metaDataColumn = "knownIndividual", missingCode = "-99")
```

Find the optimal alleleMismatch parameter for this data set.

```
> amUniqueProfile(example4, doPlot = TRUE)
```

Figure 4 demonstrates a NoSecondMinimum profile, which is a sign that allelematch cannot make a confident assessment of the unique genotypes within the range of the alleleMismatch criterion examined (by default this is 0% to 40% of allele columns mismatching). We can also, therefore, disregard the optimal alleleMismatch criterion.

For illustration let's see what would happen if we used this criterion uncritically and proceeded with analysis.

```
> uniqueExample4 <- amUnique(example4, alleleMismatch = 1)

allelematch:  assuming genotype columns are in pairs, representing 10 loci

> summary(uniqueExample4, html = "example4_1.html")
```

Thanks to the knownIndividual column we find in the output that at this low criterion virtually all samples are declared unique incorrectly.

We know from our simulation that there should be exactly 100 unique genotypes in this data set. In practice, of course, this would seldom be known. But it may be possible to have a ballpark estimate for the number of unique genotypes. Let's set our ballpark estimate at 100, and look at Figure 4 to find what setting of alleleMismatch we would need to return approximately 100 samples. This appears to be alleleMismatch=6.

```
> uniqueExample4ballpark <- amUnique(example4, alleleMismatch = 6)

allelematch:  assuming genotype columns are in pairs, representing 10 loci
```

[6] Please see the publication associated with this package for details on how data were simulated.

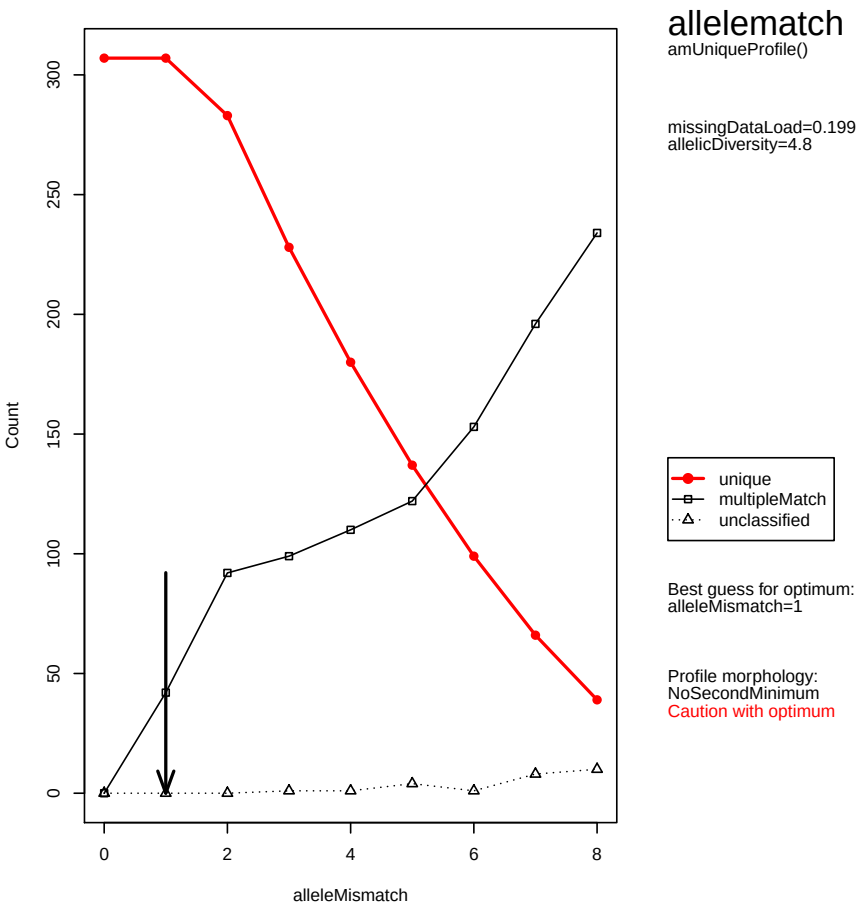

Figure 4: Low quality data set

```
> summary(uniqueExample4ballpark, html = "example4_2.html")
```

There are 99 individuals in this output, but from the knownIndividual column we can see that there are a number of individuals that have been declared unique genotypes more than once, and others that should have been declared unique have been overlooked. Given uncertainty and the absence of information in the data set, allelematch is unable to identify the unique genotypes with confidence. Arguably, a human tasked with the same job would face the same challenge.

In some analysis protocols, however, a high confidence assessment of all the unique genotypes may not be required. For example, a subsample of the unique genotypes in a population may be needed for a multi-population analysis. One approach is simply to remove samples with more than a threshold amount of missing data in the hope that this reduces ambiguity in the data set. Another approach is to set a high alleleMismatch criterion. For example:

```
> uniqueExample4high <- amUnique(example4, alleleMismatch = 8)

allelematch:  assuming genotype columns are in pairs, representing 10 loci

> summary(uniqueExample4high, html = "example4_3.html")
```

Looking at the knownIndividual column for the unique genotypes in the output we can see we have produced a set of truly unique genotypes using this very high alleleMismatch criterion, although these will be biased towards individuals who are unrelated.

Example 5 Wildlife data set

```
> data(amExample5)
```

In this final example we use real data from the non-invasive sampling of a wildlife population. The data have been anonymized by changing sampling details. A single column giving the gender is also available and we show how this can be used as an extra locus. Missing data is also more common at some loci than at others, with a total load of about 10%.

First we create the amDataset object, using the samplingData column as meta-data. Note that we have previously concatenated several sampling data columns into one because allelematch is limited to only one meta-data column and we require this information for downstream analyses. (Concatenation of the raw data set could be done using the paste() function, for example.) Also note how the index column, sampleId can contain any valid ASCII character, and is not limited to numerals. Allele columns are also not limited to numerals, as is the case with the gender column.

```
> head(levels(amExample5$samplingData))

[1] "SiouxLookout-Feb-2004" "SiouxLookout-Feb-2005"
[3] "SiouxLookout-Jan-2004" "SiouxLookout-Jan-2005"
[5] "SiouxLookout-Jan-2006" "SiouxLookout-Jan-2007"

> head(levels(amExample5$sampleId))

[1] "1001.ON.CA" "1002.ON.CA" "1003.ON.CA" "1004.ON.CA"
[5] "1005.ON.CA" "1006.ON.CA"
```

```
> head(levels(amExample5$gender))

[1] "-99" "F"   "M"

> example5 <- amDataset(amExample5, indexColumn = "sampleId",
+     metaDataColumn = "samplingData", missingCode = "-99")
```

Allelematch operates largely without reference to loci, instead working by matching elements of corresponding row vectors. Locus information is only required for the sibling P(ID) calculation. In previous examples we have not specified which allele columns were associated with each locus because there were an even number of columns. Allelematch correctly assumed that it was reading codominant diploid data, and therefore that columns were paired for each locus.

In this data set we have one unpaired column gender and ten additional loci in paired columns.

```
> names(amExample5)

 [1] "sampleId"     "samplingData" "gender"       "LOC1a"
 [5] "LOC1b"        "LOC2a"        "LOC2b"        "LOC3a"
 [9] "LOC3b"        "LOC4a"        "LOC4b"        "LOC5a"
[13] "LOC5b"        "LOC6a"        "LOC6b"        "LOC7a"
[17] "LOC7b"        "LOC8a"        "LOC8b"        "LOC9a"
[21] "LOC9b"        "LOC10a"       "LOC10b"
```

We can create a mapping vector to specify how the allele columns map onto loci. This vector should be the same length as the number of allele columns, and each element should give an arbitrary name or number for the locus it represents.

```
> example5map <- c("gender", "LOC1", "LOC1", "LOC2",
+     "LOC2", "LOC3", "LOC3", "LOC4", "LOC4", "LOC5",
+     "LOC5", "LOC6", "LOC6", "LOC7", "LOC7", "LOC8",
+     "LOC8", "LOC9", "LOC9", "LOC10", "LOC10")
```

An equivalent and much more compact way of doing this is as follows.

```
> example5map <- c(1, rep(2:11, each = 2))
```

Next we find the optimal alleleMismatch parameter for this data set. We specify the locus mapping using the multilocusMap parameter.

```
> amUniqueProfile(example5, multilocusMap = example5map,
+     doPlot = TRUE)
```

Figure 5 demonstrates a NonZeroSecondMinimum profile. We note that the optimal criterion has been correctly determined from the profile at alleleMismatch=3 because this value is the second minimum. The profile morphology, low allelic diversity and high missing data load suggest immediately that we are dealing with a data set of marginal quality and should be prepared to take extra steps to classify some unclassified samples, and review the classifications of unique genotypes that have multipleMatch samples.

Now we run the unique analysis using the recommended alleleMismatch setting and produce HTML output.

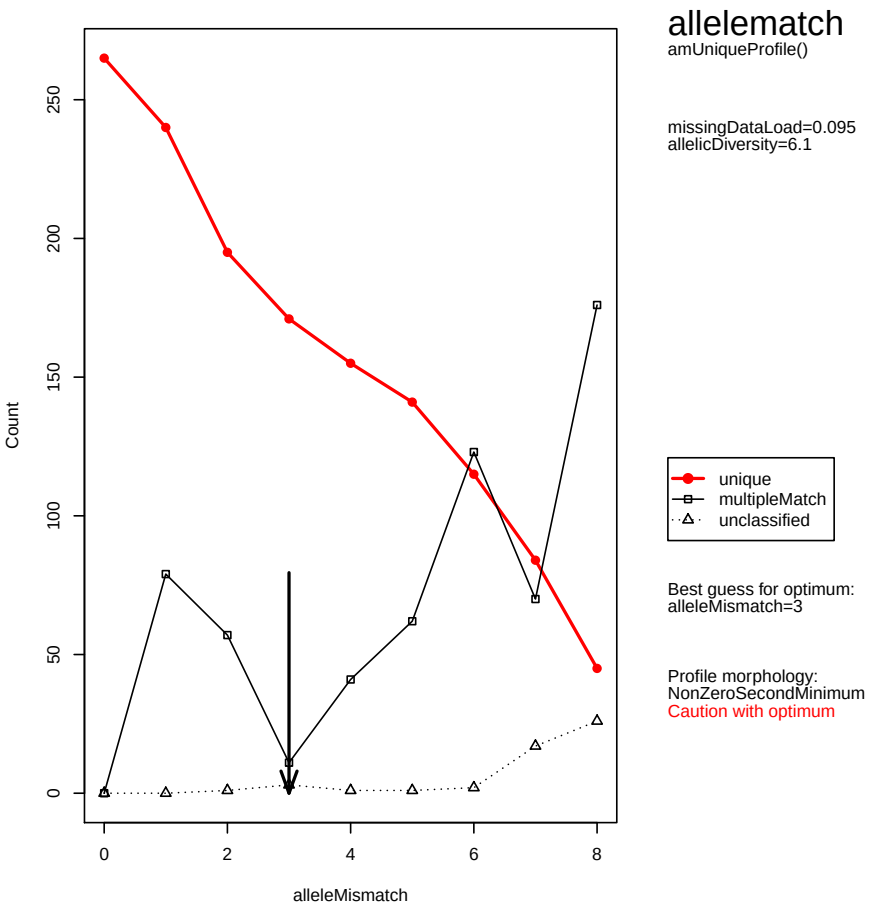

Figure 5: Wildlife data set

```
> uniqueExample5 <- amUnique(example5, multilocusMap = example5map,
+     alleleMismatch = 3)
> summary(uniqueExample5, html = "example5_1.html")
```

Looking at the output we find three unclassified samples and 11 multipleMatch samples.

We can review the unclassified samples by using the next highest alleleMismatch setting. Using this output we can decide whether to classify these as unique genotypes or matches to existing unique genotypes.

```
> unclassifiedExample5 <- amPairwise(uniqueExample5$unclassified,
+     uniqueExample5$unique, alleleMismatch = 4)
> summary(unclassifiedExample5, html = "example5_2.html")
```

Next, we can review the multipleMatch samples at the original alleleMismatch setting. This output can help us determine if the CHECK_UNIQUE samples should remain as unique genotypes, or be reclassified as matches to a unique genotype.

```
> multipleMatchExample5 <- amPairwise(uniqueExample5$multipleMatch,
+     uniqueExample5$unique, alleleMismatch = 3)
> summary(multipleMatchExample5, html = "example5_3.html")
```

Finally, if we prefer to use our favourite spreadsheet software rather than R, we can create a csv version of the original unique analysis, and use this to manually reclassify samples as required as well as prepare the data set for subsequent analyses.

```
> summary(uniqueExample5, csv = "example5_1.csv")
```

References

Hamming RW (1950) Error detecting and error correcting codes. *Bell System Technical Journal*, 29, 147-160.

Kalinowski ST, Taper ML, Marshall TC (2007) Revising how the computer program CERVUS accommodates genotyping error increases success in paternity assignment. *Molecular Ecology*, 16, 1099-1106.

Langfelder P, Zhang B, Horvath S (2008) Defining clusters from a hierarchical cluster tree: the Dynamic Tree Cut package for R. *Bioinformatics*, 24, 719.

Peakall R, Smouse PE (2006) GENALEX 6: genetic analysis in Excel. Population genetic software for teaching and research. *Molecular Ecology Notes*, 6, 288-295.

Wilberg MJ, Dreher BP (2004) GENECAP: a program for analysis of multilocus genotype data for non-invasive sampling and capture-recapture population estimation. *Molecular Ecology Notes*, 4, 783-785.