OBITools V4

Eric Coissac

1/17/23

Table of contents

Preface

The first version of *OBITools* started to be developed in 2005. This was at the beginning of the DNA metabarcoding story at the Laboratoire d'Ecologie Alpine (LECA) in Grenoble. At that time, with Pierre Taberlet and François Pompanon, we were thinking about the potential of this new methodology under development. PIerre and François developed more the laboratory methods, while I was thinking more about the tools for analysing the sequences produced. Two ideas were behind this development. I wanted something modular, and something easy to extend. To achieve the first goal, I decided to implement obitools as a suite of unix commands mimicking the classic unix commands but dedicated to sequence files. The basic unix commands are very useful for automatically manipulating, parsing and editing text files. They work in flow, line by line on the input text. The result is a new text file that can be used as input for the next command. Such a design makes it possible to quickly develop a text processing pipeline by chaining simple elementary operations. The *OBITools* are the exact counterpart of these basic Unix commands, but the basic information they process is a sequence (potentially spanning several lines of text), not a single line of text. Most *OBITools* consume sequence files and produce sequence files. Thus, the principles of chaining and modularity are respected. In order to be able to easily extend the *OBITools* to keep up with our evolving ideas about processing DNA metabarcoding data, it was decided to develop them using an interpreted language: Python. Python 2, the version available at the time, allowed us to develop the *OBITools* efficiently. When parts of the algorithms were computationally demanding, they were implemented in C and linked to the Python code. Even though Python is not the most efficient language available, even though computers were not as powerful as they are today, the size of the data we could produce using 454 sequencers or early solexa machines was small enough to be processed in a reasonable time.

The first public version of obitools was *OBITools2* (Boyer et al. 2016), this was actually a cleaned up and documented version of *OBITools* that had been running at LECA for years and was not really distributed except to a few collaborators. This is where *OBITools* started its public life from then on. The DNA metabarcoding spring schools provided and still provide user training every year. But *OBITools2* soon suffered from two limitations: it was developed in Python2, which was increasingly abandoned in favour of Python3, and the data size kept increasing with the new illumina machines. Python's intrinsic slowness coupled with the increasing size of the datasets made OBITools computation times increasingly long. The abandonment of all maintenance of Python2 by its developers also imposed the need for a new version of OBITools.

OBITools3 was the first response to this crisis. Developed and maintained by Céline Mercier, *OBITools3* attempted to address several limitations of *OBITools2*. It is a complete new code, mainly developed in Python3, with most of the lower layer code written in C for efficiency. OBITools3 has also abandoned text files for binary files for the same reason of efficiency. They have been replaced by a database structure that keeps track of every operation performed on the data.

Here we present *OBITools4* which can be seen as a return to the origins of OBITools. While *OBITools3* offered traceability of analyses, which is in line with the concept of open science, and faster execution, *OBITools2* was more versatile and not only usable for the analysis of DNA metabarcoding data. *OBITools4* is the third full implementation of *OBITools*. The idea behind this new version is to go back to the original design of *OBITools* which ran on text files containing sequences, like the classic Unix commands, but running at least as fast as *OBITools3* and taking advantage of the multicore architecture of all modern laptops. For this, the idea of relying on an interpreted language was abandoned. The *OBITools4* are now fully implemented in the GO language with the exception of a few small pieces of specific code already implemented very efficiently in C. *OBITools4* also implement a new format for the annotations inserted in the header of every sequences. Rather tha relying on a format specific to *OBITools*, by default *OBITools4* use the JSON format. This simplifies the writing of parsers in any languages, and thus allows obitools to easiestly interact with other software.

Part I

The OBITools

The *OBITools4* are programs specifically designed for analyzing NGS data in a DNA metabarcoding context, taking into account taxonomic information. It is distributed as an open source software available on the following website: http://metabarcoding.org/obitools4.

Aims of *OBITools*

DNA metabarcoding is an efficient approach for biodiversity studies (Taberlet et al. 2012). Originally mainly developed by microbiologists (*e.g.* Sogin et al. 2006), it is now widely used for plants (*e.g.* Sønstebø et al. 2010; Yoccoz et al. 2012; Parducci et al. 2012) and animals from meiofauna (*e.g.* Chariton et al. 2010; Baldwin et al. 2013) to larger organisms (*e.g.* Andersen et al. 2012; Thomsen et al. 2012). Interestingly, this method is not limited to *sensu stricto* biodiversity surveys, but it can also be implemented in other ecological contexts such as for herbivore (e.g. Valentini et al. 2009; Kowalczyk et al. 2011) or carnivore (e.g. Deagle, Kirkwood, and Jarman 2009; Shehzad et al. 2012) diet analyses.

Whatever the biological question under consideration, the DNA metabarcoding methodology relies heavily on next-generation sequencing (NGS), and generates considerable numbers of DNA sequence reads (typically million of reads). Manipulation of such large datasets requires dedicated programs usually running on a Unix system. Unix is an operating system, whose first version was created during the sixties. Since its early stages, it is dedicated to scientific computing and includes a large set of simple tools to efficiently process text files. Most of those programs can be viewed as filters extracting information from a text file to create a new text file. These programs process text files as streams, line per line, therefore allowing computation on a huge dataset without requiring a large memory. Unix programs usually print their results to their standard output (*stdout*), which by default is the terminal, so the results can be examined on screen. The main philosophy of the Unix environment is to allow easy redirection of the *stdout* either to a file, for saving the results, or to the standard input (*stdin*) of a second program thus allowing to easily create complex processing from simple base commands. Access to Unix computers is increasingly easier for scientists nowadays. Indeed, the Linux operating system, an open source version of Unix, can be freely installed on every PC machine and the MacOS operating system, running on Apple computers, is also a Unix system. The *OBITools* programs imitate Unix standard programs because they usually act as filters, reading their data from text files or the *stdin* and writing their results to the *stdout*. The main difference with classical Unix programs is that text files are not analyzed line per line but sequence record per sequence record (see below for a detailed description of a sequence record). Compared to packages for similar purposes like mothur (Schloss et al. 2009) or QIIME (Caporaso et al. 2010), the *OBITools* mainly rely on filtering and sorting algorithms. This allows users to set up versatile data analysis pipelines (Figure 1), adjustable to the broad range of DNA metabarcoding applications. The innovation of the *OBITools* is their ability to take into account the taxonomic annotations, ultimately allowing sorting and filtering of sequence records based on the taxonomy.

1 Installation of the obitools

1.1 Availability of the OBITools

The *OBITools* are open source and protected by the CeCILL 2.1 license.

All the sources of the *OBITools4* can be downloaded from the metabarcoding git server (https://git.metabarcoding.org).

1.2 Prerequisites

The *OBITools4* are developped using the GO programming language, we stick to the latest version of the language, today the 1.19.5. If you want to download and compile the sources yourself, you first need to install the corresponding compiler on your system. Some parts of the soft are also written in C, therefore a recent C compiler is also requested, GCC on Linux or Windows, the Developer Tools on Mac.

Whatever the installation you decide for, you will have to ensure that a C compiler is available on your system.

1.3 Installation with the install script

1.4 Compilation from sources

2 File formats usable with *OBITools*

OBITools manipulate have to manipulate DNA sequence data and taxonomical data. They can use some supplentary metadata describing the experiment and produce some stats about the processed DNA data. All the manipulated data are stored in text files, following standard data format.

3 The DNA sequence data

Sequences can be stored following various format. OBITools knows some of them. The central formats for sequence files manipulated by OBITools scripts are the `fasta` and `fastq` format. OBITools extends the both these formats by specifying a syntax to include in the definition line data qualifying the sequence. All file formats use the `IUPAC` code for encoding nucleotides.

Moreover these two formats that can be used as input and output formats, **OBITools4** can read the following format :

- EBML flat file format (use by ENA)
- Genbank flat file format
- ecoPCR output files

3.1 The IUPAC Code

The International Union of Pure and Applied Chemistry (IUPAC_) defined the standard code for representing protein or DNA sequences.

Code	Nucleotide
A	Adenine
C	Cytosine
G	Guanine
T	Thymine
U	Uracil
R	Purine (A or G)
Y	Pyrimidine (C, T, or U)
M	C or A
K	T, U, or G
W	T, U, or A
S	C or G
B	C, T, U, or G (not A)
D	A, T, U, or G (not C)
H	A, T, U, or C (not G)
V	A, C, or G (not T, not U)

Code	Nucleotide
N	Any base (A, C, G, T, or U)

3.2 The *fasta* sequence format

The **fasta format** is certainly the most widely used sequence file format. This is certainly due to its great simplicity. It was originally created for the Lipman and Pearson FASTA program. OBITools use in more of the classical `fasta` format an `extended version` of this format where structured data are included in the title line.

In *fasta* format a sequence is represented by a title line beginning with a `>` character and the sequences by itself following the :doc:`iupac` code. The sequence is usually split other severals lines of the same length (expect for the last one)

```
>my_sequence this is my pretty sequence
ACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGT
GTGCTGACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTGTTT
AACGACGTTGCAGTACGTTGCAGT
```

This is no special format for the title line excepting that this line should be unique. Usually the first word following the > character is considered as the sequence identifier. The end of the title line corresponding to a description of the sequence. Several sequences can be concatenated in a same file. The description of the next sequence is just pasted at the end of the record of the previous one

```
>sequence_A this is my first pretty sequence
ACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGT
GTGCTGACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTGTTT
AACGACGTTGCAGTACGTTGCAGT
>sequence_B this is my second pretty sequence
ACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGT
GTGCTGACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTGTTT
AACGACGTTGCAGTACGTTGCAGT
>sequence_C this is my third pretty sequence
ACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGT
GTGCTGACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTACGTTGCAGTGTTT
AACGACGTTGCAGTACGTTGCAGT
```

3.3 The *fastq* sequence format[1]

The **FASTQ** format is a text file format for storing both biological sequences (only nucleic acid sequences) and the associated quality scores. The sequence and score are each encoded by a single ASCII character. This format was originally developed by the Wellcome Trust Sanger Institute to link a FASTA sequence file to the corresponding quality data, but has recently become the de facto standard for storing results from high-throughput sequencers (Cock et al. 2010).

A fastq file normally uses four lines per sequence.

- Line 1 begins with a '@' character and is followed by a sequence identifier and an *optional* description (like a :ref:`fasta` title line).
- Line 2 is the raw sequence letters.
- Line 3 begins with a '+' character and is *optionally* followed by the same sequence identifier (and any description) again.
- Line 4 encodes the quality values for the sequence in Line 2, and must contain the same number of symbols as letters in the sequence.

A fastq file containing a single sequence might look like this:

```
@SEQ_ID
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT
+
!''*((((***+))%%%++)(%%%%).1***-+*''))**55CCF>>>>>>CCCCCCC65
```

The character '!' represents the lowest quality while '~' is the highest. Here are the quality value characters in left-to-right increasing order of quality (`ASCII`):

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]
^_`abcdefghijklmnopqrstuvwxyz{|}~
```

The original Sanger FASTQ files also allowed the sequence and quality strings to be wrapped (split over multiple lines), but this is generally discouraged as it can make parsing complicated due to the unfortunate choice of "@" and "+" as markers (these characters can also occur in the quality string).

[1]This article uses material from the Wikipedia article `FASTQ format` which is released under the `Creative Commons Attribution-Share-Alike License 3.0`

Sequence quality scores

The Phred quality value Q is an integer mapping of p (i.e., the probability that the corresponding base call is incorrect). Two different equations have been in use. The first is the standard Sanger variant to assess reliability of a base call, otherwise known as Phred quality score:

$$Q_{\text{sanger}} = -10 \log_{10} p$$

The Solexa pipeline (i.e., the software delivered with the Illumina Genome Analyzer) earlier used a different mapping, encoding the odds $\mathbf{p}/(1-\mathbf{p})$ instead of the probability $\mathbf{p}$:

$$Q_{\text{solexa-prior to v.1.3}} = -10 \ \log_{10} \frac{p}{1-p}$$

Although both mappings are asymptotically identical at higher quality values, they differ at lower quality levels (i.e., approximately $\mathbf{p} > 0.05$, or equivalently, $\mathbf{Q} < 13$).

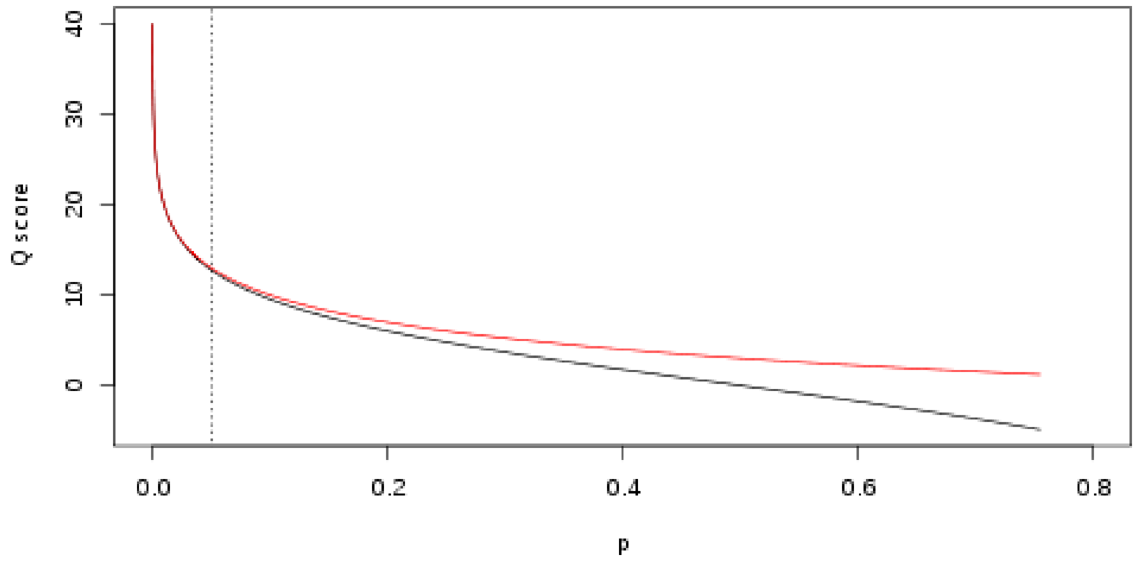


Figure 3.1: Relationship between Q and p using the Sanger (red) and Solexa (black) equations (described above). The vertical dotted line indicates $\mathbf{p} = 0.05$, or equivalently, $Q = 13$.

Encoding

The *fastq* format had differente way of encoding the Phred quality score along the time. Here a breif history of these changes is presented.

- Sanger format can encode a Phred quality score from 0 to 93 using ASCII 33 to 126 (although in raw read data the Phred quality score rarely exceeds 60, higher scores are possible in assemblies or read maps).
- Solexa/Illumina 1.0 format can encode a Solexa/Illumina quality score from -5 to 62 using ASCII 59 to 126 (although in raw read data Solexa scores from -5 to 40 only are expected)
- Starting with Illumina 1.3 and before Illumina 1.8, the format encoded a Phred quality score from 0 to 62 using ASCII 64 to 126 (although in raw read data Phred scores from 0 to 40 only are expected).
- Starting in Illumina 1.5 and before Illumina 1.8, the Phred scores 0 to 2 have a slightly different meaning. The values 0 and 1 are no longer used and the value 2, encoded by ASCII 66 "B".

  Sequencing Control Software, Version 2.6, (Catalog # SY-960-2601, Part # 15009921 Rev. A, November 2009, page 30) states the following: *If a read ends with a segment of mostly low quality (Q15 or below), then all of the quality values in the segment are replaced with a value of 2 (encoded as the letter B in Illumina's text-based encoding of quality scores)... This Q2 indicator does not predict a specific error rate, but rather indicates that a specific final portion of the read should not be used in further analyses.* Also, the quality score encoded as "B" letter may occur internally within reads at least as late as pipeline version 1.6, as shown in the following example:

```
@HWI-EAS209_0006_FC706VJ:5:58:5894:21141#ATCACG/1
TTAATTGGTAAATAAATCTCCTAATAGCTTAGATNTTACCTTNNNNNNNNNNTAGTTTCTTGAGA
TTTGTTGGGGGAGACATTTTTGTGATTGCCTTGAT
+HWI-EAS209_0006_FC706VJ:5:58:5894:21141#ATCACG/1
efcfffffcfeefffcffffffddf`feed]`]_Ba_^__[YBBBBBBBBBBRTT\]][ dddd`
ddd^dddadd^BBBBBBBBBBBBBBBBBBBBBBBB
```

An alternative interpretation of this ASCII encoding has been proposed. Also, in Illumina runs using PhiX controls, the character 'B' was observed to represent an "unknown quality score". The error rate of 'B' reads was roughly 3 phred scores lower the mean observed score of a given run.

- Starting in Illumina 1.8, the quality scores have basically returned to the use of the Sanger format (Phred+33).

OBItools follows the Sanger format. Nevertheless, It is possible to read files encoded following the Solexa/Illumina format by applying a shift of 62 (see the option **--solexa** of the OBITools commands).

3.4 File extension

There is no standard file extension for a FASTQ file, but .fq and .fastq, are commonly used.

4 OBITools V4 Tutorial

Here is a short tutorial on how to analyze DNA metabarcoding data produced on Illumina sequencers using:

- the OBITools
- some basic Unix commands

4.1 Wolves' diet based on DNA metabarcoding

The data used in this tutorial correspond to the analysis of four wolf scats, using the protocol published in Shehzad et al. (2012) for assessing carnivore diet. After extracting DNA from the faeces, the DNA amplifications were carried out using the primers `TTAGATACCCCACTATGC` and `TAGAACAGGCTCCTCTAG` amplifiying the *12S-V5* region (Riaz et al. 2011), together with a wolf blocking oligonucleotide.

The complete data set can be downloaded here: the tutorial dataset

Once the data file is downloaded, using a UNIX terminal unarchive the data from the `tgz` file.

```
tar zxvf wolf_diet.tgz
```

That command create a new directory named `wolf_data` containing every required data files:

- `fastq <fastq>` files resulting of aGA IIx (Illumina) paired-end (2 x 108 bp) sequencing assay of DNA extracted and amplified from four wolf faeces:
  - `wolf_F.fastq`
  - `wolf_R.fastq`
- the file describing the primers and tags used for all samples sequenced:
  - `wolf_diet_ngsfilter.txt` The tags correspond to short and specific sequences added on the 5' end of each primer to distinguish the different samples
- the file containing the reference database in a fasta format:

  - `db_v05_r117.fasta` This reference database has been extracted from the release 117 of EMBL using `obipcr`

To not mix raw data and processed data a new directory called `results` is created.

```
mkdir results
```

4.2 Step by step analysis

4.2.1 Recover full sequence reads from forward and reverse partial reads

When using the result of a paired-end sequencing assay with supposedly overlapping forward and reverse reads, the first step is to recover the assembled sequence.

The forward and reverse reads of the same fragment are *at the same line position* in the two fastq files obtained after sequencing. Based on these two files, the assembly of the forward and reverse reads is done with the `obipairing` utility that aligns the two reads and returns the reconstructed sequence.

In our case, the command is:

```
obipairing --min-identity=0.8 \
           --min-overlap=10 \
           -F wolf_data/wolf_F.fastq \
           -R wolf_data/wolf_R.fastq \
           > results/wolf.fastq
```

The `--min-identity` and `--min-overlap` options allow discarding sequences with low alignment quality. If after the aligment, the overlaping parts of the reads is shorter than 10 base pairs or the similarity over this aligned region is below 80% of identity, in the output file, the forward and reverse reads are not aligned but concatenated, and the value of the `mode` attribute in the sequence header is set to `joined` instead of `alignment`.

4.2.2 Remove unaligned sequence records

Unaligned sequences (:py`mode=joined`) cannot be used. The following command allows removing them from the dataset:

```
obigrep -p 'annotations.mode != "join"' \
        results/wolf.fastq > results/wolf.ali.fastq
```

The `-p` requires a go like expression. `annotations.mode != "join"` means that if the value of the `mode` annotation of a sequence is different from `join`, the corresponding sequence record will be kept.

The first sequence record of `wolf.ali.fastq` can be obtained using the following command line:

```
head -n 4 results/wolf.ali.fastq
```

The folling piece of code appears on thew window of tour terminal.

```
@HELIUM_000100422_612GNAAXX:7:108:5640:3823#0/1 {"ali_dir":"left","ali_length":62,"mode":"ali
ccgcctcctttagataccccactatgcttagccctaaacacaagtaattaatataacaaaattgttcgccagagtactaccggcaatagctta
+
CCCCCCCBCCCCCCCCCCCCCCCCCCCCBCCCCCBCCCCCCC<CcCccbe[`F`accXV<TA\RYU\\ee_e[XZ[XEEEEEEEEEE?EEE
```

4.2.3 Assign each sequence record to the corresponding sample/marker combination

Each sequence record is assigned to its corresponding sample and marker using the data provided in a text file (here `wolf_diet_ngsfilter.txt`). This text file contains one line per sample, with the name of the experiment (several experiments can be included in the same file), the name of the tags (for example: `aattaac` if the same tag has been used on each extremity of the PCR products, or `aattaac:gaagtag` if the tags were different), the sequence of the forward primer, the sequence of the reverse primer, the letter `T` or `F` for sample identification using the forward primer and tag only or using both primers and both tags, respectively (see `obimultiplex` for details).

```
obimultiplex -t wolf_data/wolf_diet_ngsfilter.txt \
             -u results/unidentified.fastq \
             results/wolf.ali.fastq \
             > results/wolf.ali.assigned.fastq
```

This command creates two files:

- `unidentified.fastq` containing all the sequence records that were not assigned to a sample/marker combination
- `wolf.ali.assigned.fastq` containing all the sequence records that were properly assigned to a sample/marker combination

Note that each sequence record of the `wolf.ali.assigned.fastq` file contains only the barcode sequence as the sequences of primers and tags are removed by the `obimultiplex` program.

Information concerning the experiment, sample, primers and tags is added as attributes in the sequence header.

For instance, the first sequence record of `wolf.ali.assigned.fastq` is:

```
@HELIUM_000100422_612GNAAXX:7:108:5640:3823#0/1_sub[28..127] {"ali_dir":"left","ali_length":6
ttagccctaaacacaagtaattaatataacaaaattgttcgccagagtactaccggcaatagcttaaaactcaaaggacttggcggtgcttta
+
CCCBCCCCCBCCCCCCC<CcCccbe[`F`accXV<TA\RYU\\ee_e[XZ[XEEEEEEEEEE?EEEEEEEEEEDEEEEEEECCCCCCCCCCC
```

4.2.4 Dereplicate reads into uniq sequences

The same DNA molecule can be sequenced several times. In order to reduce both file size and computations time, and to get easier interpretable results, it is convenient to work with unique *sequences* instead of *reads*. To *dereplicate* such *reads* into unique *sequences*, we use the `obiuniq` command.

Definition: Dereplicate reads into unique sequences

1. compare all the reads in a data set to each other
2. group strictly identical reads together
3. output the sequence for each group and its count in the original dataset (in this way, all duplicated reads are removed)

Definition adapted from Seguritan and Rohwer (2001)

For dereplication, we use the `obiuniq` command with the `-m sample`. The `-m sample` option is used to keep the information of the samples of origin for each uniquesequence.

```
obiuniq -m sample \
        results/wolf.ali.assigned.fastq \
        > results/wolf.ali.assigned.uniq.fasta
```

Note that `obiuniq` returns a fasta file.

The first sequence record of `wolf.ali.assigned.uniq.fasta` is:

```
>HELIUM_000100422_612GNAAXX:7:93:6991:1942#0/1_sub[28..126] {"ali_dir":"left","ali_length":63
ttagccctaaacataaacattcaataaacaagaatgttcgccagagaactactagcaaca
gcctgaaactcaaaggacttggcggtgctttatatccct
```

The run of `obiuniq` has added two key=values entries in the header of the fasta sequence:

- `"merged_sample":{"29a_F260619":1}`: this sequence have been found once in a single sample called **29a_F260619**
- `"count":1` : the total count for this sequence is 1

To keep only these two attributes, we can use the `obiannotate` command:

```
obiannotate -k count -k merged_sample \
  results/wolf.ali.assigned.uniq.fasta \
  > results/wolf.ali.assigned.simple.fasta
```

The first five sequence records of `wolf.ali.assigned.simple.fasta` become:

```
>HELIUM_000100422_612GNAAXX:7:26:18930:11105#0/1_sub[28..127] {"count":1,"merged_sample":{"29
ttagccctaaacacaagtaattaatataacaaaatwattcgcyagagtactacmggcaat
agctyaaarctcamagrwcttggcggtgctttataccctt
>HELIUM_000100422_612GNAAXX:7:58:5711:11399#0/1_sub[28..127] {"count":1,"merged_sample":{"29a
ttagccctaaacacaagtaattaatataacaaaattattcgccagagtwctaccgssaat
agcttaaaactcaaaggactgggcggtgctttataccctt
>HELIUM_000100422_612GNAAXX:7:100:15836:9304#0/1_sub[28..127] {"count":1,"merged_sample":{"29
ttagccctaaacatagataattacacaaacaaaattgttcaccagagtactagcggcaac
agcttaaaactcaaaggacttggcggtgctttataccctt
>HELIUM_000100422_612GNAAXX:7:55:13242:9085#0/1_sub[28..126] {"count":4,"merged_sample":{"26a
ttagccctaaacataaacattcaataaacaagagtgttcgccagagtactactagcaaca
gcctgaaactcaaaggacttggcggtgctttacatccct
>HELIUM_000100422_612GNAAXX:7:86:8429:13723#0/1_sub[28..127] {"count":7,"merged_sample":{"15a
ttagccctaaacacaagtaattaatataacaaaattattcgccagagtactaccggcaat
agcttaaaactcaaaggactcggcggtgctttataccctt
```

4.2.5 Denoise the sequence dataset

To have a set of sequences assigned to their corresponding samples does not mean that all sequences are *biologically* meaningful i.e. some of these sequences can contains PCR and/or sequencing errors, or chimeras.

Tag the sequences for PCR errors (sequence variants)

The `obiclean` program tags sequence variants as potential error generated during PCR amplification. We ask it to keep the head sequences (`-H` option) that are sequences which are not variants of another sequence with a count greater than 5% of their own count (`-r 0.05` option).

```
obiclean -s sample -r 0.05 -H \
  results/wolf.ali.assigned.simple.fasta \
      > results/wolf.ali.assigned.simple.clean.fasta
```

One of the sequence records of `wolf.ali.assigned.simple.clean.fasta` is:

```
>HELIUM_000100422_612GNAAXX:7:66:4039:8016#0/1_sub[28..127] {"count":17,"merged_sample":{"13a
clean_samplecount":1,"obiclean_singletoncount":0,"obiclean_status":{"13a_F730603":"h"},"obicl
ctagccttaaacacaaatagttatgcaaacaaaactattcgccagagtactaccggcaac
agcccaaaactcaaaggacttggcggtgcttcacaccctt
```

To remove such sequences as much as possible, we first discard rare sequences and then rsequence variants that likely correspond to artifacts.

Get some statistics about sequence counts

```
obicount results/wolf.ali.assigned.simple.clean.fasta
```

```
time="2023-02-23T18:43:37+01:00" level=info msg="Appending results/wolf.ali.assigned.simple.c
 2749 36409 273387
```

The dataset contains 4313 sequences variant corresponding to 42452 sequence reads. Most of the variants occur only a single time in the complete dataset and are usualy named *singletons*

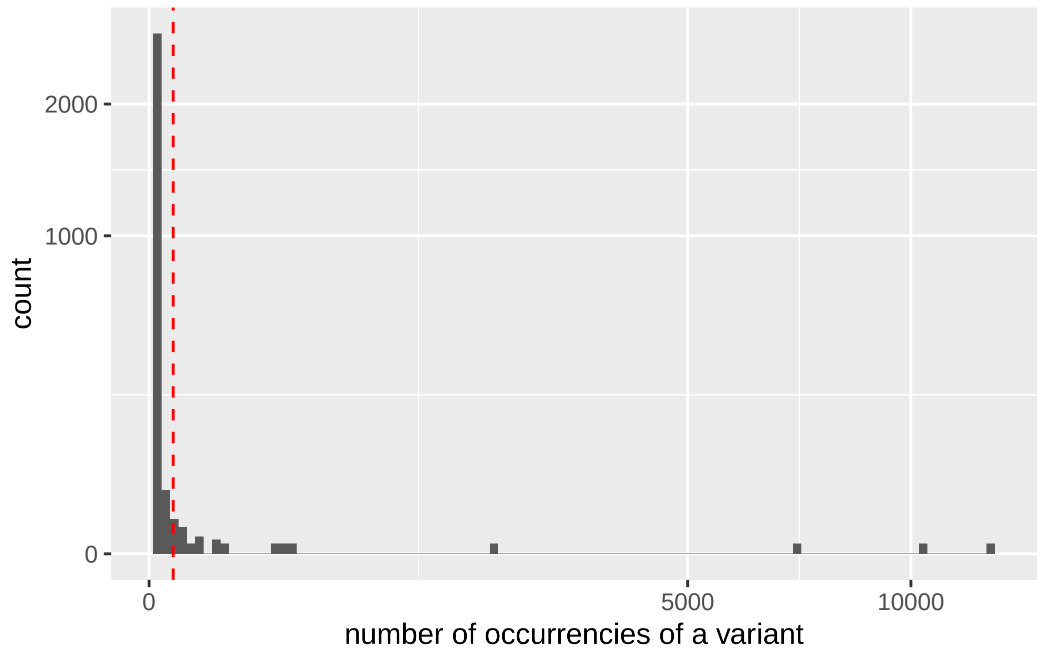
```
obigrep -p 'sequence.Count() == 1' results/wolf.ali.assigned.simple.clean.fasta \
    | obicount
```

```
time="2023-02-23T18:43:37+01:00" level=info msg="Reading sequences from stdin in guessed\n"
time="2023-02-23T18:43:37+01:00" level=info msg="Appending results/wolf.ali.assigned.simple.c
time="2023-02-23T18:43:37+01:00" level=info msg="On output use JSON headers"
 2623 2623 261217
```

In that dataset sigletons corresponds to 3511 variants.

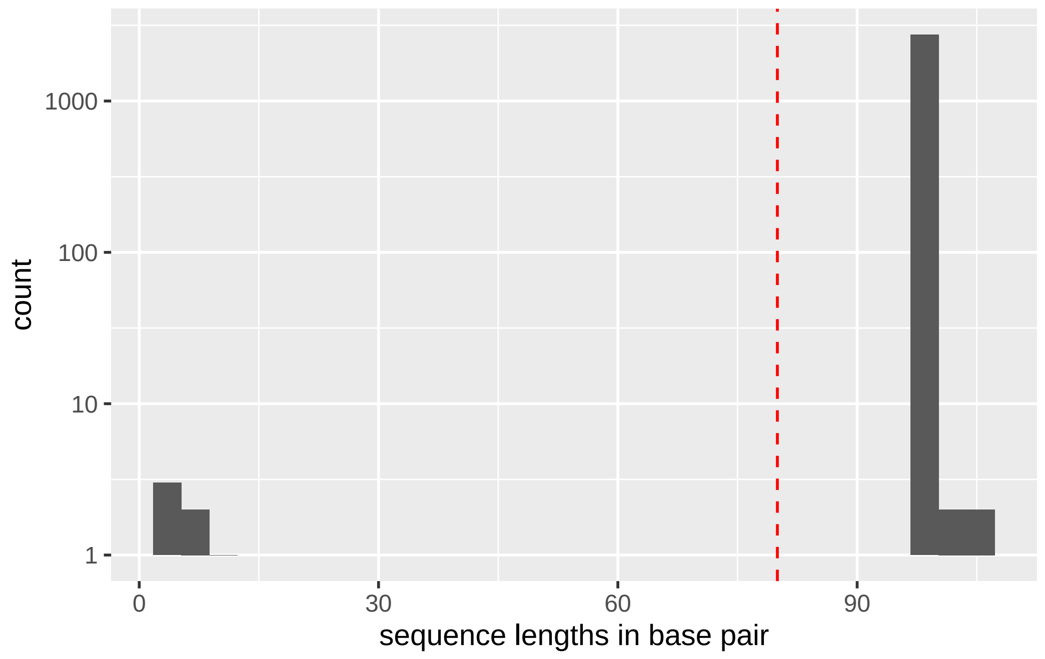
Using *R* and the `ROBIFastread` package able to read headers of the fasta files produced by *OBITools*, we can get more complete statistics on the distribution of occurrencies.

```
library(ROBIFastread)
library(ggplot2)

seqs <- read_obifasta("results/wolf.ali.assigned.simple.clean.fasta",keys="count")

ggplot(data = seqs,  mapping=aes(x = count)) +
  geom_histogram(bins=100) +
  scale_y_sqrt() +
  scale_x_sqrt() +
  geom_vline(xintercept = 10, col="red", lty=2) +
  xlab("number of occurrencies of a variant")
```



In a similar way it is also possible to plot the distribution of the sequence length.

```
ggplot(data = seqs,  mapping=aes(x = nchar(sequence))) +
  geom_histogram() +
  scale_y_log10() +
  geom_vline(xintercept = 80, col="red", lty=2) +
  xlab("sequence lengths in base pair")
```



Keep only the sequences having a count greater or equal to 10 and a length shorter than 80 bp

Based on the previous observation, we set the cut-off for keeping sequences for further analysis to a count of 10. To do this, we use the `obigrep <scripts/obigrep>` command. The `-p 'count>=10'` option means that the `python` expression :py`count>=10` must be evaluated to :py`True` for each sequence to be kept. Based on previous knowledge we also remove sequences with a length shorter than 80 bp (option -l) as we know that the amplified 12S-V5 barcode for vertebrates must have a length around 100bp.

```
obigrep -l 80 -p 'sequence.Count() >= 10' results/wolf.ali.assigned.simple.clean.fasta \
    > results/wolf.ali.assigned.simple.clean.c10.l80.fasta
```

The first sequence record of `results/wolf.ali.assigned.simple.clean.c10.l80.fasta` is:

```
>HELIUM_000100422_612GNAAXX:7:22:2603:18023#0/1_sub[28..127] {"count":12182,"merged_sample":
ttagccctaaacacaagtaattaatataacaaaattattcgccagagtactaccggcaat
agcttaaaactcaaaggacttggcggtgctttataccctt
```

At that time in the data cleanning we have conserved :

```
obicount results/wolf.ali.assigned.simple.clean.c10.l80.fasta
```

```
time="2023-02-23T18:43:40+01:00" level=info msg="Appending results/wolf.ali.assigned.simple.c
 26 31337 2585
```

4.2.6 Taxonomic assignment of sequences

Once denoising has been done, the next step in diet analysis is to assign the barcodes to the corresponding species in order to get the complete list of species associated to each sample.

Taxonomic assignment of sequences requires a reference database compiling all possible species to be identified in the sample. Assignment is then done based on sequence comparison between sample sequences and reference sequences.

Download the taxonomy

It is always possible to download the complete taxonomy from NCBI using the following commands.

```
mkdir TAXO
cd TAXO
curl http://ftp.ncbi.nih.gov/pub/taxonomy/taxdump.tar.gz \
   | tar -zxvf -
cd ..
```

For people have a low speed internet connection, a copy of the `taxdump.tar.gz` file is provided in the wolf_data directory. The NCBI taxonomy is dayly updated, but the one provided here is ok for running this tutorial.

To build the TAXO directory from the provided `taxdump.tar.gz`, you need to execute the following commands

```
mkdir TAXO
cd TAXO
tar zxvf wolf_data/taxdump.tar.gz
cd ..
```

Build a reference database

One way to build the reference database is to use the `obipcr` program to simulate a PCR and extract all sequences from a general purpose DNA database such as genbank or EMBL that can be amplified *in silico* by the two primers (here **TTAGATACCCCACTATGC** and **TAGAACAGGCTCCTCTAG**) used for PCR amplification.

The two steps to build this reference database would then be

1. Today, the easiest database to download is *Genbank*. But this will take you more than a day and occupy more than half a terabyte on your hard drive. In the `wolf_data` directory, a shell script called `download_gb.sh` is provided to perform this task. It requires that the programs `wget2` and `curl` are available on your computer.
2. Use `obipcr` to simulate amplification and build a reference database based on the putatively amplified barcodes and their recorded taxonomic information.

As these steps can take a long time (about a day for the download and an hour for the PCR), we already provide the reference database produced by the following commands so you can skip its construction. Note that as the Genbank and taxonomic database evolve frequently, if you run the following commands you may get different results.

4.2.6.0.1 * Download the sequences

```
mkdir genbank
cd genbank
../wolf_data/install_gb.sh
cd ..
```

DO NOT RUN THIS COMMAND EXCEPT IF YOU ARE REALLY CONSIENT OF THE TIME AND DISK SPACE REQUIRED.

4.2.6.0.2 * Use obipcr to simulate an in silico‘ PCR

```
obipcr -t TAXO -e 3 -l 50 -L 150 \
       --forward TTAGATACCCCACTATGC \
       --reverse TAGAACAGGCTCCTCTAG \
       --no-order \
       genbank/Release-251/gb*.seq.gz
       > results/v05.pcr.fasta
```

Note that the primers must be in the same order both in `wolf_diet_ngsfilter.txt` and in the `obipcr` command. The part of the path indicating the *Genbank* release can change. Please check in your genbank directory the exact name of your release.

4.2.6.0.3 * Clean the database

1. filter sequences so that they have a good taxonomic description at the species, genus, and family levels (`obigrep` command command below).
2. remove redundant sequences (`obiuniq` command below).
3. ensure that the dereplicated sequences have a taxid at the family level (`obigrep` command below).
4. ensure that sequences each have a unique identification (`obiannotate` command below)

```
obigrep -t TAXO \
          --require-rank species \
          --require-rank genus \
          --require-rank family \
          results/v05.ecopcr > results/v05_clean.fasta

obiuniq -c taxid \
        results/v05_clean.fasta \
        > results/v05_clean_uniq.fasta

obirefidx -t TAXO results/v05_clean_uniq.fasta \
        > results/v05_clean_uniq.indexed.fasta
```

Warning

From now on, for the sake of clarity, the following commands will use the filenames of the files provided with the tutorial. If you decided to run the last steps and use the files you have produced, you'll have to use `results/v05_clean_uniq.indexed.fasta` instead of `wolf_data/db_v05_r117.indexed.fasta`.

4.2.7 Assign each sequence to a taxon

Once the reference database is built, taxonomic assignment can be carried out using the `obitag` command.

```
obitag -t TAXO -R wolf_data/db_v05_r117.indexed.fasta \
       results/wolf.ali.assigned.simple.clean.c10.l80.fasta \
       > results/wolf.ali.assigned.simple.clean.c10.l80.taxo.fasta
```

The `obitag` adds several attributes in the sequence record header, among them:

- obitag_bestmatch=ACCESSION where ACCESSION is the id of hte sequence in the reference database that best aligns to the query sequence;
- obitag_bestid=FLOAT where FLOAT*100 is the percentage of identity between the best match sequence and the query sequence;
- taxid=TAXID where TAXID is the final assignation of the sequence by `obitag`
- scientific_name=NAME where NAME is the scientific name of the assigned taxid.

The first sequence record of `wolf.ali.assigned.simple.clean.c10.l80.taxo.fasta` is:

```
>HELIUM_000100422_612GNAAXX:7:81:18704:12346#0/1_sub[28..126] {"count":88,"merged_sample":
ttagccctaaacataaacattcaataaacaagaatgttcgccagaggactactagcaata
gcttaaaactcaaaggacttggcggtgctttatatccct
```

4.2.8 Generate the final result table

Some unuseful attributes can be removed at this stage.

- obiclean_head
- obiclean_headcount
- obiclean_internalcount
- obiclean_samplecount
- obiclean_singletoncount

```
obiannotate  --delete-tag=obiclean_head \
             --delete-tag=obiclean_headcount \
             --delete-tag=obiclean_internalcount \
             --delete-tag=obiclean_samplecount \
             --delete-tag=obiclean_singletoncount \
  results/wolf.ali.assigned.simple.clean.c10.l80.taxo.fasta \
  > results/wolf.ali.assigned.simple.clean.c10.l80.taxo.ann.fasta
```

The first sequence record of `wolf.ali.assigned.simple.c10.l80.clean.taxo.ann.fasta` is then:

```
>HELIUM_000100422_612GNAAXX:7:84:16335:5083#0/1_sub[28..126] {"count":96,"merged_sample":{"2
ttagccctaaacataagctattccataacaaaataattcgccagagaactactagcaaca
gattaaacctcaaaggacttggcagtgctttatacccct
```

4.2.9 Looking at the data in R

```
library(ROBIFastread)
library(vegan)
```

```
Le chargement a nécessité le package : permute
```

```
Le chargement a nécessité le package : lattice
```

```
This is vegan 2.6-4
```

```
library(magrittr)


diet_data <- read_obifasta("results/wolf.ali.assigned.simple.clean.c10.l80.taxo.fasta")
diet_data %<>% extract_features("obitag_bestmatch","obitag_rank","scientific_name",'taxid'

diet_tab <- extract_readcount(diet_data,key="obiclean_weight")
diet_tab
```

```
4 x 26 sparse Matrix of class "dgCMatrix"
```

```
  [[ suppressing 26 column names 'HELIUM_000100422_612GNAAXX:7:100:4828:3492#0/1_sub[28..127
```

```
13a_F730603 22  . 9  19 25  1  .  .  .   .  .  .   .  .  . 20  .    .   .  .
29a_F260619  . 44 .   1  . 13  . 25  .   .  . 16 391  .  .  .  . 6275 110  .
15a_F730814  .  . 4   5  .  .  .  .  .   .  .  .   .  .  .  .  . 9138   .  .
26a_F040644  .  . . 481  .  . 14  . 43 208 72  .   . 52 88  . 31    .  14 18

13a_F730603     .  .  .   . 15 8409
29a_F260619     .  .  . 353  .    .
15a_F730814     .  .  .   .  .    .
26a_F040644 12265 15 17   .  .    .
```

This file contains 26 sequences. You can deduce the diet of each sample:

- 13a_F730603: Cervus elaphus
- 15a_F730814: Capreolus capreolus
- 26a_F040644: Marmota sp. (according to the location, it is Marmota marmota)

- 29a_F260619: Capreolus capreolus

Note that we also obtained a few wolf sequences although a wolf-blocking oligonucleotide was used.

Part II

The *OBITools V4* commands

5 Specifying the data input to *OBITools* commands

5.1 Specifying input format

Five sequence formats are accepted for input files. *Fasta* (Section 3.2) and *Fastq* (Section 3.3) are the main ones, EMBL and Genbank allow the use of flat files produced by these two international databases. The last one, ecoPCR, is maintained for compatibility with previous *OBITools* and allows to read *ecoPCR* outputs as sequence files.

- `--ecopcr` : Read data following the *ecoPCR* output format.
- `--embl` Read data following the *EMBL* flatfile format.
- `--genbank` Read data following the *Genbank* flatfile format.

Several encoding schemes have been proposed for quality scores in *Fastq* format. Currently, *OBITools* considers Sanger encoding as the standard. For reasons of compatibility with older datasets produced with *Solexa* sequencers, it is possible, by using the following option, to force the use of the corresponding quality encoding scheme when reading these older files.

- `--solexa` Decodes quality string according to the Solexa specification. (default: false)

6 Controling OBITools outputs

6.1 Specifying output format

Only two output sequence formats are supported by OBITools, Fasta and Fastq. Fastq is used when output sequences are associated with quality information. Otherwise, Fasta is the default format. However, it is possible to force the output format by using one of the following two options. Forcing the use of Fasta results in the loss of quality information. Conversely, when the Fastq format is forced with sequences that have no quality data, dummy qualities set to 40 for each nucleotide are added.

- `--fasta-output` Read data following the ecoPCR output format.
- `--fastq-output` Read data following the EMBL flatfile format.

OBITools allows multiple input files to be specified for a single command.

- `--no-order` When several input files are provided, indicates that there is no order among them. (default: false). Using such option can increase a lot the processing of the data.

6.2 The Fasta and Fastq annotations format

OBITools extend the Fasta and Fastq formats by introducing a format for the title lines of these formats allowing to annotate every sequence. While the previous version of OBITools used an *ad-hoc* format for these annotation, this new version introduce the usage of the standard JSON format to store them.

On input, OBITools automatically recognize the format of the annotations, but two options allows to force the parsing following one of them. You should normally not need to use these options.

- `--input-OBI-header` FASTA/FASTQ title line annotations follow OBI format. (default: false)
- `--input-json-header` FASTA/FASTQ title line annotations follow json format. (default: false)

On output, by default annotation are formatted using the new JSON format. For compatibility with previous version of OBITools and with external scripts and software, it is possible to force the usage of the previous OBITools format.

- `--output-OBI-header|-O` output FASTA/FASTQ title line annotations follow OBI format. (default: false)
- `--output-json-header` output FASTA/FASTQ title line annotations follow json format. (default: false)

7 Options common to most of the *OBITools* commands

7.1 Helpful options

--help, -h Display a friendly help message.

--no-progressbar

7.2 System related options

Managing parallel execution of tasks

A new feature of *OBITools* V4 is the ability to run multiple tasks in parallel, reading files, calculating on the data, formatting and writing the results. Each of these tasks can itself be parallelized by dividing the data into batches and running the calculation on several batches in parallel. This allows the overall calculation time of an *OBITools* command to be reduced considerably. The parameters organizing the parallel calculation are determined automatically to use the maximum capacity of your computer. But in some circumstances, it is necessary to override these default settings either to try to optimize the computation on a given machine, or to limit the OBITools to using only a part of the computational capacity. There are two options for doing this.

--max-cpu OBITools V4 are able to run in parallel on all the CPU cores available on the computer. It is sometime required to limit the computation to a smaller number of cores. That option specify the maximum number of cores that the OBITools command can use. This behaviour can also be set up using the `OBIMAXCPU` environment variable.

--workers, -w

If your computer has 8 cores, but you want to limit *OBITools* to use only two of them you have several solution:

- If you want to set the limit for a single execution you can use the **–max-cpu** option

  ```
  obiconvert --max-cpu 2 --fasta-output data.fastq > data.fasta
  ```

  or you can precede the command by setting the environment variable `OBIMAXCPU`

  ```
  OBIMAXCPU=2 obiconvert --fasta-output data.fastq > data.fasta
  ```

- If you want to set the limit to your complete session, you have to export `OBIMAXCPU`

  ```
  export OBIMAXCPU=2
  ```

  all the following OBITools commands will be limited to use at max 2 CPU cores.

- If all the time you want to impose this limit, you must include the above `export` command in your `.bashrc` file.

OBITools debuging related options

--debug

8 OBITools expression language

Several OBITools (*e.g.* obigrep, obiannotate) allow the user to specify some simple expressions to compute values or define predicates. This expressions are parsed and evaluated using the gval go package, which allows for evaluating go-Like expression.

8.1 Variables usable in the expression

- `sequence` is the sequence object on which the expression is evaluated.
- `annotations`is a map object containing every annotations associated to the currently processed sequence.

8.2 Function defined in the language

Instrospection functions

`len(x)` It is a generic function allowing to retreive the size of a object. It returns the length of a sequences, the number of element in a map like `annotations`, the number of elements in an array. The reurned value is an `int`.

`contains(map,key)` Tests if the `map` contains a value assciated to `key`

Cast functions

`int(x)` Converts if possible the `x` value to an integer value. The function returns an `int`.

`numeric(x)` Converts if possible the `x` value to a float value. The function returns a `float`.

`bool(x)` Converts if possible the `x` value to a boolean value. The function returns a `bool`.

String related functions

`printf(format,...)` Allows to combine several values to build a string. `format` follows the classical C `printf` syntax. The function returns a `string`.

`subspc(x)` substitutes every space in the `x` string by the underscore (_) character. The function returns a `string`.

Condition function

`ifelse(condition,val1,val2)` The `condition` value has to be a `bool` value. If it is `true` the function returns `val1`, otherwise, it is returning `val2`.

8.2.1 Sequence analysis related function

`composition(sequence)` The nucleotide composition of the sequence is returned as as map indexed by `a`, `c`, `g`, or `t` and each value is the number of occurrences of that nucleotide. A fifth key `others` accounts for all others symboles.

`gcskew(sequence)` Computes the excess of g compare to c of the sequence, known as the GC skew.

$$Skew_{GC} = \frac{G - C}{G + C}$$

8.3 Accessing to the sequence annotations

The `annotations` variable is a map object containing all the annotations associated to the currently processed sequence. Index of the map are the attribute names. It exists to possibillities to retreive an annotation. It is possible to use the classical `[]` indexing operator, putting the attribute name quoted by double quotes between them.

```
annotations["direction"]
```

The above code retreives the `direction` annotation. A second notation using the dot (`.`) is often more convenient.

```
annotations.direction
```

Special attributes of the sequence are accessible only by dedicated methods of the `sequence` object.

- The sequence identifier : `Id()`
- THe sequence definition : `Definition()`

```
sequence.Id()
```

9 Metabarcode design and quality assessment

9.1 `obipcr`

Replace the `ecoPCR` original *OBITools*

10 File format conversions

10.1 `obiconvert`

11 Sequence annotations

11.1 `obiannotate`

11.2 `obitag`

11.3 `obitagpcr`

12 Computations on sequences

12.1 `obipairing`

Replace the `illuminapairedends` original *OBITools*

Alignment procedure

`obipairing` is introducing a new alignment algorithm compared to the `illuminapairedend` command of the `OBITools V2`. Nethertheless this new algorithm has been design to produce the same results than the previous, except in very few cases.

The new algorithm is a two-step procedure. First, a FASTN-type algorithm (Lipman and Pearson 1985) identifies the best offset between the two matched readings. This identifies the region of overlap.

In the second step, the matching regions of the two reads are extracted along with a flanking sequence of Δ base pairs. The two subsequences are then aligned using a "one side free end-gap" dynamic programming algorithm. This latter step is only called if at least one mismatch is detected by the FASTP step.

Unless the similarity between the two reads at their overlap region is very low, the addition of the flanking regions in the second step of the alignment ensures the same alignment as if the dynamic programming alignment was performed on the full reads.

The scoring system

In the dynamic programming step, the match and mismatch scores take into account the quality scores of the two aligned nucleotides. By taking these into account, the probability of a true match can be calculated for each aligned base pair.

If we consider a nucleotide read with a quality score Q, the probability of misreading this base (P_E) is :

$$P_E = 10^{-\frac{Q}{10}}$$

Thus, when a given nucleotide X is observed with the quality score Q. The probability that X is really an X is :

$$P(X = X) = 1 - P_E$$

Otherwise, X is actually one of the three other possible nucleotides (X_{E1}, X_{E2} or X_{E3}). If we suppose that the three reading error have the same probability :

$$P(X = X_{E1}) = P(X = X_{E3}) = P(X = X_{E3}) = \frac{P_E}{3}$$

At each position in an alignment where the two nucleotides X_1 and X_2 face each other (not a gapped position), the probability of a true match varies depending on whether $X_1 = X_2$, an observed match, or $X_1 \neq X_2$, an observed mismatch.

Probability of a true match when $X_1 = X_2$

That probability can be divided in two parts. First X_1 and X_2 have been correctly read. The corresponding probability is :

$$\begin{aligned} P_{TM} &= (1 - PE_1)(1 - PE_2) \\ &= (1 - 10^{-\frac{Q_1}{10}})(1 - 10^{-\frac{Q_2}{10}}) \end{aligned}$$

Secondly, a match can occure if the true nucleotides read as X_1 and X_2 are not X_1 and X_2 but identical.

$$\begin{aligned} P(X_1 == X_{E1}) \cap P(X_2 == X_{E1}) &= \frac{P_{E1}P_{E2}}{9} \\ P(X_1 == X_{Ex}) \cap P(X_2 == X_{Ex}) &= \frac{P_{E1}P_{E2}}{3} \end{aligned}$$

The probability of a true match between X_1 and X_2 when $X_1 = X_2$ an observed match :

$$P(MATCH|X_1 = X_2) = (1 - PE_1)(1 - PE_2) + \frac{P_{E1}P_{E2}}{3}$$

Probability of a true match when $X_1 \neq X_2$

That probability can be divided in three parts.

a. X_1 has been correctly read and X_2 is a sequencing error and is actually equal to X_1.

$$P_a = (1 - P_{E1})\frac{P_{E2}}{3}$$

b. X_2 has been correctly read and X_1 is a sequencing error and is actually equal to X_2.

$$P_b = (1 - P_{E2})\frac{P_{E1}}{3}$$

c. X_1 and X_2 corresponds to sequencing error but are actually the same base X_{Ex}

$$P_c = 2\frac{P_{E1}P_{E2}}{9}$$

Consequently :

$$P(MATCH|X_1 \neq X_2) = (1 - P_{E1})\frac{P_{E2}}{3} + (1 - P_{E2})\frac{P_{E1}}{3} + 2\frac{P_{E1}P_{E2}}{9}$$

Probability of a match under the random model

The second considered model is a pure random model where every base is equiprobable, hence having a probability of occurrence of a nucleotide equals 0.25. Under that hypothesis

$$P(MATCH|\text{Random model}) = 0.25$$

The score is a log ration of likelyhood

Score is define as the logarithm of the ratio between the likelyhood of the observations considering the sequencer error model over tha likelyhood u

12.2 `obimultiplex`

Replace the `ngsfilter` original *OBITools*

12.3 `obicomplement`

12.4 `obiclean`

12.5 `obiuniq`

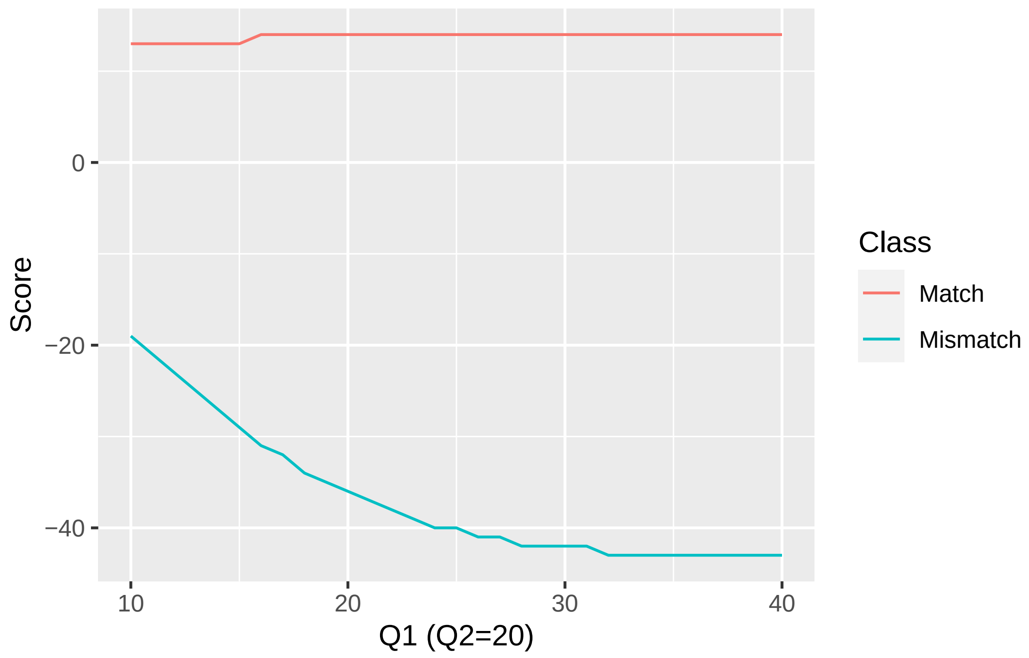


Figure 12.1: Evolution of the match and mismatch scores when the quality of base is 20 while the second range from 10 to 40.

13 Sequence sampling and filtering

13.1 `obigrep` – filters sequence files according to numerous conditions

The `obigrep` command is somewhat analogous to the standard Unix `grep` command. It selects a subset of sequence records from a sequence file. A sequence record is a complex object consisting of an identifier, a set of attributes (a key, defined by its name, associated with a value), a definition, and the sequence itself. Instead of working text line by text line like the standard Unix tool, `obigrep` selection is done sequence record by sequence record. A large number of options allow you to refine the selection on any element of the sequence. `obigrep` allows you to specify multiple conditions simultaneously (which take on the value `TRUE` or `FALSE`) and only those sequence records which meet all conditions (all conditions are `TRUE`) are selected. `obigrep` is able to work on two paired read files. The selection criteria apply to one or the other of the readings in each pair depending on the mode chosen (**--paired-mode** option). In all cases the selection is applied in the same way to both files, thus maintaining their consistency.

13.1.1 The options usable with `obigrep`

13.1.1.1 Selecting sequences based on their caracteristics

Sequences can be selected on several of their caracteristics, their length, their id, their sequence. Options allow for specifying the condition if selection.

Selection based on the sequence

Sequence records can be selected according if they match or not with a pattern. The simplest pattern is as short sequence (*e.g* `AACCTT`). But the usage of regular patterns allows for looking for more complex pattern. As example, `A[TG]C+G` matches a `A`, followed by a `T` or a `G`, then one or several `C` and endly a `G`.

--sequence|-s ***PATTERN*** Regular expression pattern to be tested against the sequence itself. The pattern is case insensitive. A complete description of the regular pattern grammar is available here.

Examples: Selects only the sequence records that contain an *EcoRI* restriction site.

```
obigrep -s 'GAATTC' seq1.fasta > seq2.fasta
```

: Selects only the sequence records that contain a stretch of at least 10 `A`.

```
obigrep -s 'A{10,}' seq1.fasta > seq2.fasta
```

: Selects only the sequence records that do not contain ambiguous nucleotides.

```
obigrep -s '^[ACGT]+$' seq1.fasta > seq2.fasta
```

--min-count | -c *COUNT* only sequences reprensenting at least *COUNT* reads will be selected. That option rely on the `count` attribute. If the `count` attribute is not defined for a sequence record, it is assumed equal to 1.

--max-count | -C *COUNT* only sequences reprensenting no more than *COUNT* reads will be selected. That option rely on the `count` attribute. If the `count` attribute is not defined for a sequence record, it is assumed equal to 1.

Examples Selecting sequence records representing at least five reads in the dataset.

```
obigrep -c 5 data_SPER01.fasta > data_norare_SPER01.fasta
```

14 Utilities

14.1 `obicount`

`obicount` counts the number of sequence records, the sum of the `count` attributes, and the sum of the length of all the sequences.

Example:

```
obicount seq.fasta
```

Prints the number of sequence records contained in the `seq.fasta` file and the sum of their `count` attributes.

Options specific to the command

- `--reads|-r` Prints read counts.
- `--symbols|-s` Prints symbol counts.
- `--variants|-v` Prints variant counts.

14.2 `obidistribute`

14.3 `obifind`

Replace the `ecofind` original *OBITools.*

Part III

The GO *OBITools* library

BioSequence

The `BioSequence` class is used to represent biological sequences. It allows for storing : - the sequence itself as a `[]byte` - the sequencing quality score as a `[]byte` if needed - an identifier as a `string` - a definition as a `string` - a set of *(key, value)* pairs in a `map[sting]interface{}`

BioSequence is defined in the obiseq module and is included using the code

```go
import (
    "git.metabarcoding.org/lecasofts/go/obitools/pkg/obiseq"
)
```

Creating new instances

To create new instance, use

- `MakeBioSequence(id string, sequence []byte, definition string) obiseq.BioSequence`
- `NewBioSequence(id string, sequence []byte, definition string) *obiseq.BioSequence`

Both create a `BioSequence` instance, but when the first one returns the instance, the second returns a pointer on the new instance. Two other functions `MakeEmptyBioSequence`, and `NewEmptyBioSequence` do the same job but provide an uninitialized objects.

- `id` parameters corresponds to the unique identifier of the sequence. It mist be a string constituted of a single word (not containing any space).
- `sequence` is the DNA sequence itself, provided as a `byte` array (`[]byte`).
- `definition` is a `string`, potentially empty, but usualy containing a sentence explaining what is that sequence.

```go
import (
    "git.metabarcoding.org/lecasofts/go/obitools/pkg/obiseq"
)

func main() {
    myseq := obiseq.NewBiosequence(
        "seq_GH0001",
        bytes.FromString("ACGTGTCAGTCG"),
        "A short test sequence",
        )
}
```

When formated as fasta the parameters correspond to the following schema

```
>id definition containing potentially several words
sequence
```

End of life of a `BioSequence` instance

When an instance of `BioSequence` is no longer in use, it is normally taken over by the GO garbage collector. If you know that an instance will never be used again, you can, if you wish, call the `Recycle` method on it to store the allocated memory elements in a `pool` to limit the allocation effort when many sequences are being handled. Once the recycle method has been called on an instance, you must ensure that no other method is called on it.

Accessing to the elements of a sequence

The different elements of an `obiseq.BioSequence` must be accessed using a set of methods. For the three main elements provided during the creation of a new instance methodes are :

- `Id() string`
- `Sequence() []byte`
- `Definition() string`

It exists pending method to change the value of these elements

- `SetId(id string)`
- `SetSequence(sequence []byte)`
- `SetDefinition(definition string)`

```go
import (
    "fmt"
    "git.metabarcoding.org/lecasofts/go/obitools/pkg/obiseq"
)

func main() {
    myseq := obiseq.NewBiosequence(
        "seq_GH0001",
        bytes.FromString("ACGTGTCAGTCG"),
        "A short test sequence",
        )

    fmt.Println(myseq.Id())
    myseq.SetId("SPE01_0001")
    fmt.Println(myseq.Id())
}
```

Different ways for accessing an editing the sequence

If `Sequence()`and `SetSequence(sequence []byte)` methods are the basic ones, several other methods exist.

- `String() string` return the sequence directly converted to a `string` instance.
- The `Write` method family allows for extending an existing sequence following the buffer protocol.
  - `Write(data []byte) (int, error)` allows for appending a byte array on 3' end of the sequence.
  - `WriteString(data string) (int, error)` allows for appending a `string`.
  - `WriteByte(data byte) error` allows for appending a single `byte`.

The `Clear` method empties the sequence buffer.

```go
import (
    "fmt"
    "git.metabarcoding.org/lecasofts/go/obitools/pkg/obiseq"
)

func main() {
    myseq := obiseq.NewEmptyBiosequence()

    myseq.WriteString("accc")
    myseq.WriteByte(byte('c'))
    fmt.Println(myseq.String())
}
```

Sequence quality scores

Sequence quality scores cannot be initialized at the time of instance creation. You must use dedicated methods to add quality scores to a sequence.

To be coherent the length of both the DNA sequence and que quality score sequence must be equal. But assessment of this constraint is realized. It is of the programmer responsability to check that invariant.

While accessing to the quality scores relies on the method `Quality() []byte`, setting the quality need to call one of the following method. They run similarly to their sequence dedicated conterpart.

- `SetQualities(qualities Quality)`
- `WriteQualities(data []byte) (int, error)`

- `WriteByteQualities(data byte) error`

In a way analogous to the `Clear` method, `ClearQualities()` empties the sequence of quality scores.

The annotations of a sequence

A sequence can be annotated with attributes. Each attribute is associated with a value. An attribute is identified by its name. The name of an attribute consists of a character string containing no spaces or blank characters. Values can be of several types.

- Scalar types:
  - integer
  - numeric
  - character
  - boolean
- Container types:
  - vector
  - map

Vectors can contain any type of scalar. Maps are compulsorily indexed by strings and can contain any scalar type. It is not possible to have nested container type.

Annotations are stored in an object of type `bioseq.Annotation` which is an alias of `map[string]interface{}`. This map can be retrieved using the `Annotations() Annotation` method. If no annotation has been defined for this sequence, the method returns an empty map. It is possible to test an instance of `BioSequence` using its `HasAnnotation() bool` method to see if it has any annotations associated with it.

- GetAttribute(key string) (interface{}, bool)

The sequence iterator

The pakage *obiter* provides an iterator mecanism for manipulating sequences. The main class provided by this package is `obiiter.IBioSequence`. An `IBioSequence` iterator provides batch of sequences.

Basic usage of a sequence iterator

Many functions, among them functions reading sequences from a text file, return a `IBioSequence` iterator. The iterator class provides two main methods:

- `Next() bool`
- `Get() obiiter.BioSequenceBatch`

The `Next` method moves the iterator to the next value, while the `Get` method returns the currently pointed value. Using them, it is possible to loop over the data as in the following code chunk.

```go
import (
    "git.metabarcoding.org/lecasofts/go/obitools/pkg/obiformats"
)

func main() {
    mydata := obiformats.ReadFastSeqFromFile("myfile.fasta")

    for mydata.Next() {
        data := mydata.Get()
        //
        // Whatever you want to do with the data chunk
        //
    }
}
```

An `obiseq.BioSequenceBatch` instance is a set of sequences stored in an `obiseq.BioSequenceSlice` and a sequence number. The number of sequences in a batch is not defined. A batch can even contain zero sequences, if for example all sequences initially included in the batch have been filtered out at some stage of their processing.

The `Pipable` functions

A function consuming a `obiiter.IBioSequence` and returning a `obiiter.IBioSequence` is of class `obiiter.Pipable`.

The `Teeable` functions

A function consuming a `obiiter.IBioSequence` and returning two `obiiter.IBioSequence` instance is of class `obiiter.Teeable`.

A Annexes

A.1 Sequence attributes

ali_dir (`string`)

- Set by the *obipairing* tool
- The attribute can contain 2 string values `left` or `right`.

The alignment generated by *obipairing* is a 3'-end gap free algorithm. Two cases can occur when aligning the forward and reverse reads. If the barcode is long enough, both the reads overlap only on their 3' ends. In such case, the alignment direction `ali_dir` is set to *left*. If the barcode is shorter than the read length, the paired reads overlap by their 5' ends, and the complete barcode is sequenced by both the reads. In that later case, `ali_dir` is set to *right*.

ali_length (`int`)

- Set by the *obipairing* tool

Length of the aligned parts when merging forward and reverse reads

count (`int`)

- Set by the *obiuniq* tool
- Getter : method `Count()`
- Setter : method `SetCount(int)`

The `count` attribute indicates how-many strictly identical reads have been merged in a single record. It contains an integer value. If it is absent this means that the sequence record represents a single occurrence of the sequence.

The `Count()` method allows to access to the count attribute as an integer value. If the `count` attribute is not defined for the given sequence, the value *1* is returned

merged_* (`map[string]int`)

- Set by the *obiuniq* tool

The `-m` option of the *obiuniq* tools allows for keeping track of the distribution of the values stored in given attribute of interest. Often this option is used to summarise distribution of a sequence variant accross samples when *obiuniq* is run after running *obimultiplex*. The actual name of the attribute depends on the name of the monitored attribute. If `-m` option is used with the attribute *sample*, then this attribute names *merged_sample*.

mode (`string`)

- Set by the *obipairing* tool
- The attribute can contain 2 string values `join` or `alignment`.

obitag_ref_index (`map[string]string`)

- Set by the *obirefidx* tool.

It resumes to which taxonomic annotation a match to that sequence must lead according to the number of differences existing between the query sequence and the reference sequence having that tag.

```
{"0":"9606@Homo sapiens@species",
 "2":"207598@Homininae@subfamily",
 "3":"9604@Hominidae@family",
 "8":"314295@Hominoidea@superfamily",
 "10":"9526@Catarrhini@parvorder",
 "12":"1437010@Boreoeutheria@clade",
 "16":"9347@Eutheria@clade",
 "17":"40674@Mammalia@class",
 "22":"117571@Euteleostomi@clade",
 "25":"7776@Gnathostomata@clade",
 "29":"33213@Bilateria@clade",
 "30":"6072@Eumetazoa@clade"}
```

pairing_mismatches (`map[string]string`)

- Set by the *obipairing* tool

seq_a_single (`int`)

- Set by the *obipairing* tool

seq_ab_match (`int`)

- Set by the *obipairing* tool

seq_b_single (`int`)

- Set by the *obipairing* tool

score (`int`)

- Set by the *obipairing* tool

score_norm (`float`)

- Set by the *obipairing* tool
- The value ranges between 0 and 1.

Score of the alignment between forward and reverse reads expressed as a fraction of identity.

References

Andersen, Kenneth, Karen Lise Bird, Morten Rasmussen, James Haile, Henrik Breuning-Madsen, Kurt H Kjaer, Ludovic Orlando, M Thomas P Gilbert, and Eske Willerslev. 2012. "Meta-barcoding of ëdirtíDNA from soil reflects vertebrate biodiversity." *Molecular Ecology* 21 (8): 1966–79.

Baldwin, Darren S, Matthew J Colloff, Gavin N Rees, Anthony A Chariton, Garth O Watson, Leon N Court, Diana M Hartley, et al. 2013. "Impacts of inundation and drought on eukaryote biodiversity in semi-arid floodplain soils." *Molecular Ecology* 22 (6): 1746–58. https://doi.org/10.1111/mec.12190.

Boyer, Frédéric, Céline Mercier, Aurélie Bonin, Yvan Le Bras, Pierre Taberlet, and Eric Coissac. 2016. "obitools: a unix-inspired software package for DNA metabarcoding." *Molecular Ecology Resources* 16 (1): 176–82. https://doi.org/10.1111/1755-0998.12428.

Caporaso, J Gregory, Justin Kuczynski, Jesse Stombaugh, Kyle Bittinger, Frederic D Bushman, Elizabeth K Costello, Noah Fierer, et al. 2010. "QIIME allows analysis of high-throughput community sequencing data." *Nature Methods* 7 (5): 335–36. https://doi.org/10.1038/nmeth.f.303.

Chariton, Anthony A, Anthony C Roach, Stuart L Simpson, and Graeme E Batley. 2010. "Influence of the choice of physical and chemistry variables on interpreting patterns of sediment contaminants and their relationships with estuarine macrobenthic communities." *Marine and Freshwater Research*. https://doi.org/10.1071/mf09263.

Cock, Peter JA, Christopher J Fields, Naohisa Goto, Michael L Heuer, and Peter M Rice. 2010. "The Sanger FASTQ File Format for Sequences with Quality Scores, and the Solexa/Illumina FASTQ Variants." *Nucleic Acids Research* 38 (6): 1767–71.

Deagle, Bruce E, Roger Kirkwood, and Simon N Jarman. 2009. "Analysis of Australian fur seal diet by pyrosequencing prey DNA in faeces." *Molecular Ecology* 18 (9): 2022–38. https://doi.org/10.1111/j.1365-294X.2009.04158.x.

Kowalczyk, Rafał, Pierre Taberlet, Eric Coissac, Alice Valentini, Christian Miquel, Tomasz Kamiński, and Jan M Wójcik. 2011. "Influence of management practices on large herbivore diet—Case of European bison in Białowieża Primeval Forest (Poland)." *Forest Ecology and Management* 261 (4): 821–28. https://doi.org/10.1016/j.foreco.2010.11.026.

Lipman, D J, and W R Pearson. 1985. "Rapid and sensitive protein similarity searches." *Science* 227 (4693): 1435–41. http://www.ncbi.nlm.nih.gov/pubmed/2983426.

Parducci, Laura, Tina Jørgensen, Mari Mette Tollefsrud, Ellen Elverland, Torbjørn Alm, Sonia L Fontana, K D Bennett, et al. 2012. "Glacial survival of boreal trees in northern Scandinavia." *Science* 335 (6072): 1083–86. https://doi.org/10.1126/science.1216043.

Riaz, Tiayyba, Wasim Shehzad, Alain Viari, François Pompanon, Pierre Taberlet, and Eric

Coissac. 2011. “ecoPrimers: inference of new DNA barcode markers from whole genome sequence analysis.” *Nucleic Acids Research* 39 (21): e145. https://doi.org/10.1093/nar/gkr732.

Schloss, Patrick D, Sarah L Westcott, Thomas Ryabin, Justine R Hall, Martin Hartmann, Emily B Hollister, Ryan A Lesniewski, et al. 2009. “Introducing mothur: open-source, platform-independent, community-supported software for describing and comparing microbial communities.” *Applied and Environmental Microbiology* 75 (23): 7537–41. https://doi.org/10.1128/AEM.01541-09.

Seguritan, V, and F Rohwer. 2001. “FastGroup: a program to dereplicate libraries of 16S rDNA sequences.” *BMC Bioinformatics* 2 (October): 9. https://doi.org/10.1186/1471-2105-2-9.

Shehzad, Wasim, Tiayyba Riaz, Muhammad A Nawaz, Christian Miquel, Carole Poillot, Safdar A Shah, Francois Pompanon, Eric Coissac, and Pierre Taberlet. 2012. “Carnivore diet analysis based on next-generation sequencing: Application to the leopard cat (Prionailurus bengalensis) in Pakistan.” *Molecular Ecology* 21 (8): 1951–65. https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1365-294X.2011.05424.x.

Sogin, Mitchell L, Hilary G Morrison, Julie A Huber, David Mark Welch, Susan M Huse, Phillip R Neal, Jesus M Arrieta, and Gerhard J Herndl. 2006. “Microbial diversity in the deep sea and the underexplored ”rare biosphere”.” *Proceedings of the National Academy of Sciences of the United States of America* 103 (32): 12115–20. https://doi.org/10.1073/pnas.0605127103.

Sønstebø, J H, L Gielly, A K Brysting, R Elven, M Edwards, J Haile, E Willerslev, et al. 2010. “Using next-generation sequencing for molecular reconstruction of past Arctic vegetation and climate.” *Molecular Ecology Resources* 10 (6): 1009–18. https://doi.org/10.1111/j.1755-0998.2010.02855.x.

Taberlet, Pierre, Eric Coissac, Mehrdad Hajibabaei, and Loren H Rieseberg. 2012. “Environmental DNA.” *Molecular Ecology* 21 (8): 1789–93. https://doi.org/10.1111/j.1365-294X.2012.05542.x.

Thomsen, Philip Francis, Jos Kielgast, Lars L Iversen, Carsten Wiuf, Morten Rasmussen, M Thomas P Gilbert, Ludovic Orlando, and Eske Willerslev. 2012. “Monitoring endangered freshwater biodiversity using environmental DNA.” *Molecular Ecology* 21 (11): 2565–73. https://doi.org/10.1111/j.1365-294X.2011.05418.x.

Valentini, Alice, Christian Miquel, Muhammad Ali Nawaz, Eva Bellemain, Eric Coissac, François Pompanon, Ludovic Gielly, et al. 2009. “New perspectives in diet analysis based on DNA barcoding and parallel pyrosequencing: the trnL approach.” *Molecular Ecology Resources* 9 (1): 51–60. https://doi.org/10.1111/j.1755-0998.2008.02352.x.

Yoccoz, N G, K A Bråthen, L Gielly, J Haile, M E Edwards, T Goslar, H Von Stedingk, et al. 2012. “DNA from soil mirrors plant taxonomic and growth form diversity.” *Molecular Ecology* 21 (15): 3647–55. https://doi.org/10.1111/j.1365-294X.2012.05545.x.