DnaBarcodes: Manual version 0.1

Anders Gonçalves da Silva*[1] and Rohan H. Clarke[1]

[1]School of Biological Sciences, Monash University

May 6, 2014

1 Introduction

DnaBarcodes is a program that harnesses the power of the *Phred* [Ewing et al., 1998, Ewing and Green, 1998], *Phrap* (P. Green *unpubl*), and *PolyPhred* [Nickerson et al., 1997] to assemble, perform quality control, and annotate DNA barcodes from any molecule. Thus, it provides the user with a powerful and validated method for base-calling and assembly of DNA sequence data (*Phred*, *Phrap* pipeline), and flexibility to use any gene (or partial gene) as a DNA barcode. DnaBarcodes is written in *Python*, and is open-source, so anyone can download it for free, modify it, and improve it, so it will better fit their own projects, labs, and purposes.

Throughout the manual, we assume the user has some familiarity with the command-line. If not, we refer the reader to the following useful tutorials:

1. http://www.ee.surrey.ac.uk/Teaching/Unix/
2. http://freeengineer.org/learnUNIXin10minutes.html

*Corresponding author: `andersgs@gmail.com`

2 Installation

2.1 Dependencies

To install DnaBarcodes the following are necessary:

1. *Python* version 2.7.+ and PERL version 5.12+;

2. *Pip* - a *Python* library that installs third-party *Python* libraries;

3. The latest version of DnaBarcodes (which can be downloaded here: https://github.com/andersgs/DNABarcodes)

4. Installer packages for *Phred*, *Phrap*, *PolyPhred*, and *Consed*. These can be obtained directly from authors at: http://www.phrap.org/consed/consed.html#howToGet and http://droog.gs.washington.edu/polyphred/poly_get.html.

5. The latest version of NCBIs BLAST+ Tools, which can be found at http://www.ncbi.nlm.nih.gov/books/NBK1763/.

If running MacOSX, make sure do download and install the Xcode, and within Xcode, make sure to install the *command-line* tools. If you dont know what any of that means, this webpage will help: https://guide.macports.org/chunked/installing.xcode.html.

If running Windows, an installer has not yet been made, nor are we certain how to do it. We have written the *Python* code with the idea that the code will work on any platform. We also know that, at least, *Phred*, *Phrap*, and *PolyPhred* also have Windows versions. If there is someone that would like to try, we would be grateful. We suspect that one will need *Cygwin* (http://www.cygwin.com), and then install *Python* (http://python.wonderhowto.com/how-to/set-up-cygwin-for-python-programming–249466/) and *PERL* (http://www.cs.unc.edu/∼jeffay/dirt/FAQ/cygwin-perl.html).

2.2 Mac OSX

To install DnaBarcoder on a MacOSX machine (tested on MacOSX 10.8.5), download or clone the DnaBarcoder **git** repository (https://github.com/andersgs/DNABarcodes). Using the *Terminal*, go to the DnaBarcoder directory, and create a folder called *packages*:

```
> cd <path to the DnaBarcoder directory>
> mkdir packages
```

Copy into the *packages* folder the files obtained to install *Phred*, *Phrap*, *PolyPhred*, and *Consed*. Rename then as below:
Phred -> *phred.zip*
Phrap -> *phrap.tar.gz*
PolyPhred - > *polyphred.tar.gz*
Consed -> *consed.tar.gz*
This can be done in the *Finder* or using the *cp* command (assuming the file is in the *Downloads* folder:

```
> cd <path to the DnaBarcodes directory>/packages
> mv ~/Downloads/distrib.xxx.tar.gz phrap.tar.gz
```

Once this is completed, move to the DnaBarcoder directory, and run the *install.sh* script using *sudo*:

```
> sudo ./install.sh
```

Then, simply follow the prompts. Once the script has finished, you can move into the *example/* folder, to test the installation (see Running).

2.3 Linux

Currently, there is no installer script ready for *Linux*, but that is on our to-do list. However, in most *Linux* installations, all the necessary tools will already be available.

Nevertheless, a quick guide would be the following:

1. Download or clone the DnaBarcoder git repository (https://github.com/andersgs/DNABarcodes)

2. Obtain and install *Phred*, *Phrap*, *PolyPhred*, and *Consed*. Follow the instructions carefully to make sure the appropriate environmental variables are set, and that the *phredPhrap* script is setup appropriately;

3. Ensure that *Python* 2.7.+ and *PERL* 5.12+ are installed, if not they should be easily obtainable from *apt-get* or similar app repository;

4. Install the NCBI+ BLAST Tools;
5. Install DnaBarcodes using *pip*. This is done by using the *Terminal* to open the DnaBarcodes folder, and then:

   ```
   pip install dist/DnaBarcodes–0.1.tar.gz
   ```

6. Once pip has successfully completed its job, you should be able to try out DnaBarcodes by using the example in the *example* folder and the instructions below (see Running).

2.4 Windows

Coming soon For now, we recommend using DnaBarcodes on a *Linux* or *MacOSX* platform.

3 Running

There are three ways of running DnaBarcodes: 1. a menu driven approach, which can be accessed by running the script *create_barcode_proj.py*; 2. directly from the command-line using the script *run_dnabarcodes.py*; and, 3. through the *Python* interpreter by importing the appropriate module.

First, we will describe how to use the *create_barcode_proj.py* script, we will then proceed to the *run_dnabarcodes.py* script, and finally, we will discuss how to import and use DnaBarcodes from the *Python* interpreter.

3.1 Using the menu

To start a new project, one must first create a project root or home folder. In this case, we will use the *example_project* folder that you wouldve downloaded from the git repository. The folder is already created, but if you were to start from scratch, to make a directory from the command-line, first move to the folder where you want to house your project home folder, then type the following:

```
> mkdir <project_name>
> cd <project_name>
```

You should substitute the <project_name> by whatever you would like to call your project. In this manual, anything in <> is an option that should be filled out by the user.

But, going back to our example, go to the *example_project* folder, by typing the following:

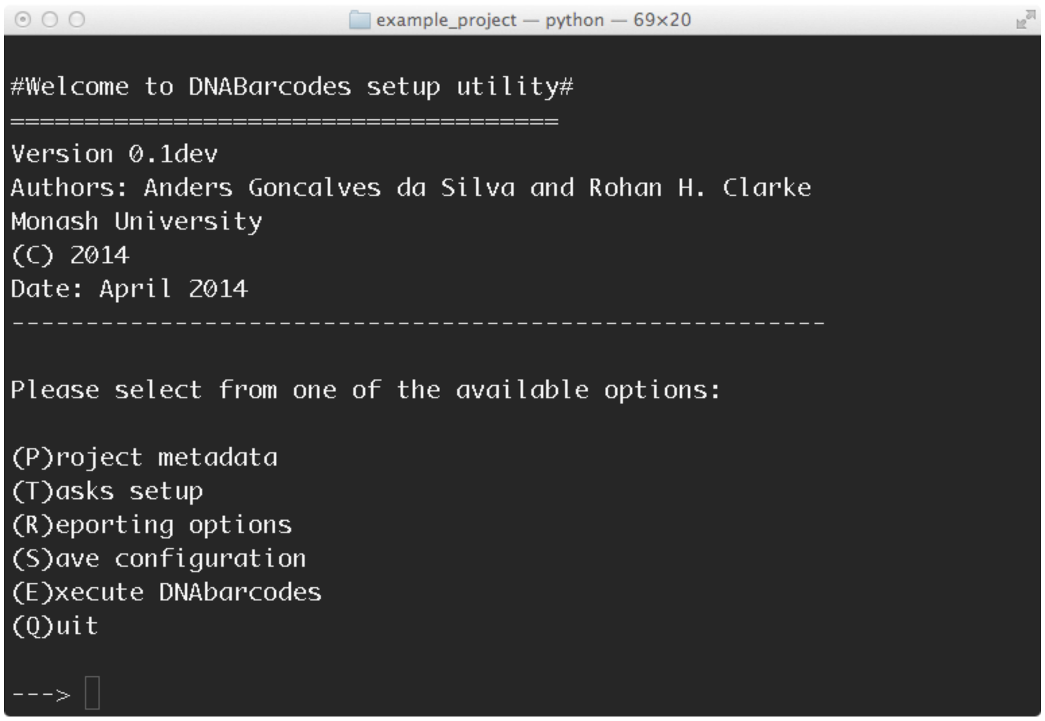
```
> cd <path to the DnaBarcodes directory>/example_project
> ls
```

The *ls* command will list all the files and folders in the folder. When you create a new project, none of these will be there, but will be created as you run the program. The important file here is *dbcode.cfg*. This file contains all the information that DnaBarcodes needs to run, it is as a configuration file. You can open to have a look, *but dont edit it unless you know what you are doing*. If this file does not exist, it will be automatically created using the *create_dnabarcode_proj.py* script.

So, let us give the program a go. Type the following in the command-line (making sure that you are in the project *home folder*).

```
> create_dnabarcodes_proj.py
```

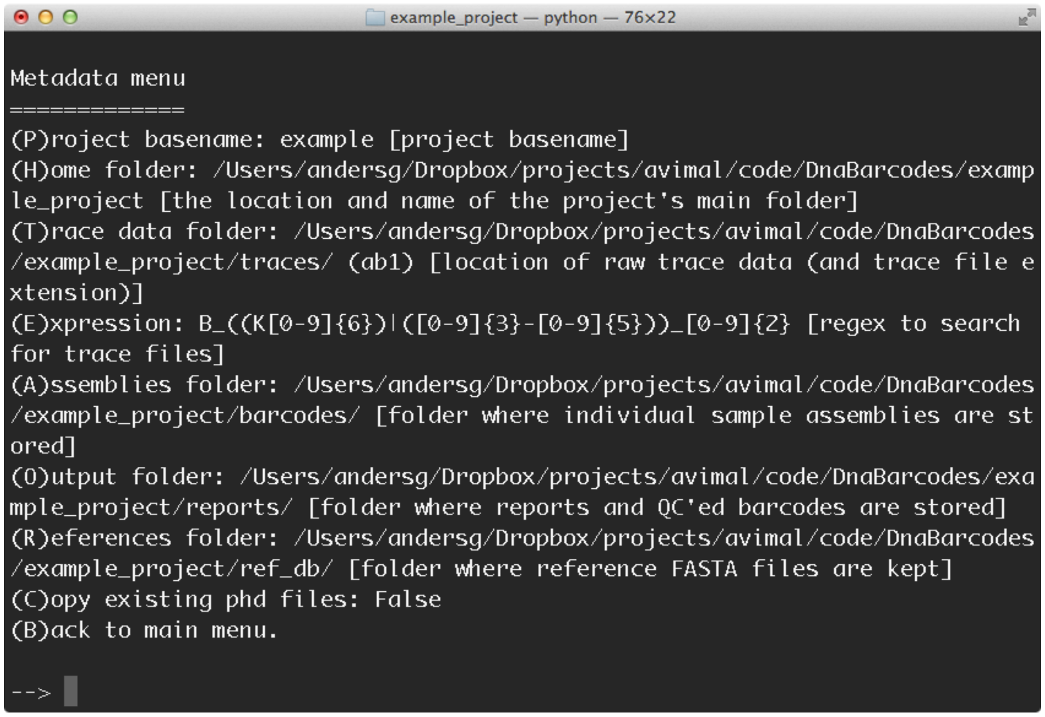
The screen will clear, and a menu will appear (Figure 1). It currently holds six options: (1) Project metadata; (2) Tasks setup; (3) Reporting options; (4) Save configuration; (5) Execute DnaBarcodes; and (6) Quit. In Menu Items below a table explains each item individually.



```
#Welcome to DNABarcodes setup utility#
=====================================
Version 0.1dev
Authors: Anders Goncalves da Silva and Rohan H. Clarke
Monash University
(C) 2014
Date: April 2014
-------------------------------------------------------

Please select from one of the available options:

(P)roject metadata
(T)asks setup
(R)eporting options
(S)ave configuration
(E)xecute DNAbarcodes
(Q)uit

--->
```

Fig 1. DnaBarcodes main menu

Let us start with the *Project metadata* menu. Type P (capital p) and then press enter. A new menu will appear with all the project metadata (Figure 2).



```
Metadata menu
=============
(P)roject basename: example [project basename]
(H)ome folder: /Users/andersg/Dropbox/projects/avimal/code/DnaBarcodes/examp
le_project [the location and name of the project's main folder]
(T)race data folder: /Users/andersg/Dropbox/projects/avimal/code/DnaBarcodes
/example_project/traces/ (ab1) [location of raw trace data (and trace file e
xtension)]
(E)xpression: B_((K[0-9]{6})|([0-9]{3}-[0-9]{5}))_[0-9]{2} [regex to search
for trace files]
(A)ssemblies folder: /Users/andersg/Dropbox/projects/avimal/code/DnaBarcodes
/example_project/barcodes/ [folder where individual sample assemblies are st
ored]
(O)utput folder: /Users/andersg/Dropbox/projects/avimal/code/DnaBarcodes/exa
mple_project/reports/ [folder where reports and QC'ed barcodes are stored]
(R)eferences folder: /Users/andersg/Dropbox/projects/avimal/code/DnaBarcodes
/example_project/ref_db/ [folder where reference FASTA files are kept]
(C)opy existing phd files: False
(B)ack to main menu.

-->
```

Fig 2. DnaBarcodes metadata menu

3.1.1 Metadata menu

Once in the *Metadata menu* start by pressing P followed by enter. The P option is referring to the P project basename, which in this case we are naming *example*. You can type any name you would like, but we suggest you keep it simple. The project basename will be the prefix of all the report files produced while running DnaBarcodes. In this case, type *example* followed by enter.

Once you press enter, you will be back in the main menu. Next, we will choose the projects home folder. The program assumes you are running it from the projects home folder, and will auto populate that option with the current directory. However, if you would prefer to run the program in another folder, that is okay too. Just keep in mind that *create_dnabarcode_proj.py* will look for the configuration file in the directory where it is run. If it does not find a config file, it will create a default one. If you decide to change the directory, press H (that is capital h) followed by enter. The program will then ask you to enter the FULL directory path the projects home folder. *This is one of the the only times you will need to enter a full path to a folder*, all other folders will be relative to the home folder.

The following menu item, *Trace data folder*, specifies two important parameters: (1) where you want to store your raw trace files, and (2) what extension do these data files have. By default, the program will assume that the raw data will be kept in *traces/* folder within the projects home folder, and that the trace files will have an ab1 extension (the default for ABI Sanger Automated Sequencers). Press T then enter to try it out.

In the next step, if you leave everything blank, the program will assume that you are happy with the defaults (i.e., *traces/* and *ab1*). However, it will now try to open the folder. If it isnt available, it will ask you if you want to create it. It will then look for trace files in the folder, if it doesnt find any it will ask if you want to copy any over. If you say yes, you must enter a FULL path to a folder containing trace files. This folder should contain subfolders named by sequencing run. The program will copy the subfolders over to the *traces/* folder.

If you looked in the *traces/* folder before starting, you would have noticed it was empty. That is because the examples trace files are in the folder *test_traces*. When asked for a folder containing pre-existing traces that you would like to copy over, punch in the full path to the *test_traces/* folder. If you look in *traces* folder now, you will find two run folders, and a total of 10 traces, six in one folder and four in the other. We will get back to what they are a little later.

Once the program has finished copying the trace files, you will get a message saying it has been successful, and to press any key to return to the menu. It will also tell you how many traces were copied over. This is only necessary the first time you run the program. In subsequent runs, in order to add new data, just copy the run folder you receive from your provider or you download from the sequencing machine, into the *traces* folder. Once you have added the Run *create_dnabarcoder_proj.py*, and select the *Execute DnaBarcodes* option from the menu. But, we will get to that soon enough.

You should now be back in the *Metadata menu*. The next option on the menu (*Expression*) assumes you have planned ahead, and named your trace files following a meaningful pattern that references, in some way, the sample from which the sequence data refers to. *This is fundamental* and without it DnaBarcodes will not work properly. Let us go over the example traces, and it will become clear why it is important. The example trace files have the following naming patterns:

B<*banding_id*><two_digit_trip><*two_digit_rep*><primer_code>.ab1

The B_ just stands for *bird*, because these barcodes are originating from bird blood samples. The <banding_id> stands for the banding number that uniquely identifies the individual from which the blood sample came. The <two_digit_trip> is a two digit number that identifies in which sampling trip the blood sample was collected, this in conjuction with the banding ID uniquely identifies a blood sample. This is because blood samples could have originated from a recapture of an individual. The <two_digit_rep> identifies the PCR used to generate the sequenced amplicon. This way, if multiple PCRs are needed to generate a high-quality barcode, then it is possible to identify which PCRs contributed to the final barcode. Finally, the <primer_code> identifies the sequencing primer used to generate the sequence data.

The program uses a *regular expression* to search the trace files, and identify how many unique blood samples (band_id + two_digit_trip) are represented in the trace files. It then creates a *phred/phrap* assembly folder for each sample, using the sample identification as the name for the folder. Within this assembly folder, all the sub-folders necessary to run *phred/phrap/ polyphred/consed* are created, and the traces are copied over to the appropriate location.

Thus, without a meaningful pattern to group trace files into unique samples, the program is impossible to use. If you do not know what what a regular expression is, please go to the following tutorial: http://regexone.com. We use regexpal.com to develop our regular expressions.

If you press E (capital e) then enter, you will get a prompt asking you to enter a regular expression. If you leave it blank, the default (used for the example) is taken. The program will then ask if you would like to test the regular expression. If you say yes, the program will search the *traces* folder, and using the regex, will tell you how many files it found that were matched by the expression. In this case, the result should be 10. Once the test is completed successfully, you can press any key to return to the *Metadata menu.*

As mentioned above, the program will create individual assembly folders for each unique sample identified in the *traces* folder. These folders will be created in the *assemblies* folder. The next option in the program is to set the path to the *assemblies* folder.

Press A (capital a) followed by enter, you will then be prompted for the *assemblies* folder name. The program assumes that this is a folder located within the projects home folder. The default is *barcodes*, so you can either leave it blank, and just press enter, or enter a new name (*just the name, not a full path*). Once you press enter, the program will check if the folder exists, and if it doesnt, it will ask you if you want to create it. If you wish to accept press y, else press n. If you press n you will receive a warning, alerting you to the fact that without this directory the program will not work.

The same process can be repeated for the *output* folder, where all the reports will be kept (the default is *reports*), and for the *references* folder (the default is *ref_db*). The references folder should contain three files (all optional): (1) a FASTA file containing annotated barcodes to be used as a local BLAST database; (2) a FASTA file containing reference protein sequences that can be used to ensure that all produced barcodes are in the same reading frame; and (3) a FASTA file containing sequences that flank the barcode this will be used to trim any excess sequence data beyond the barcode. The names for these files will be determined in the *Tasks* menu.

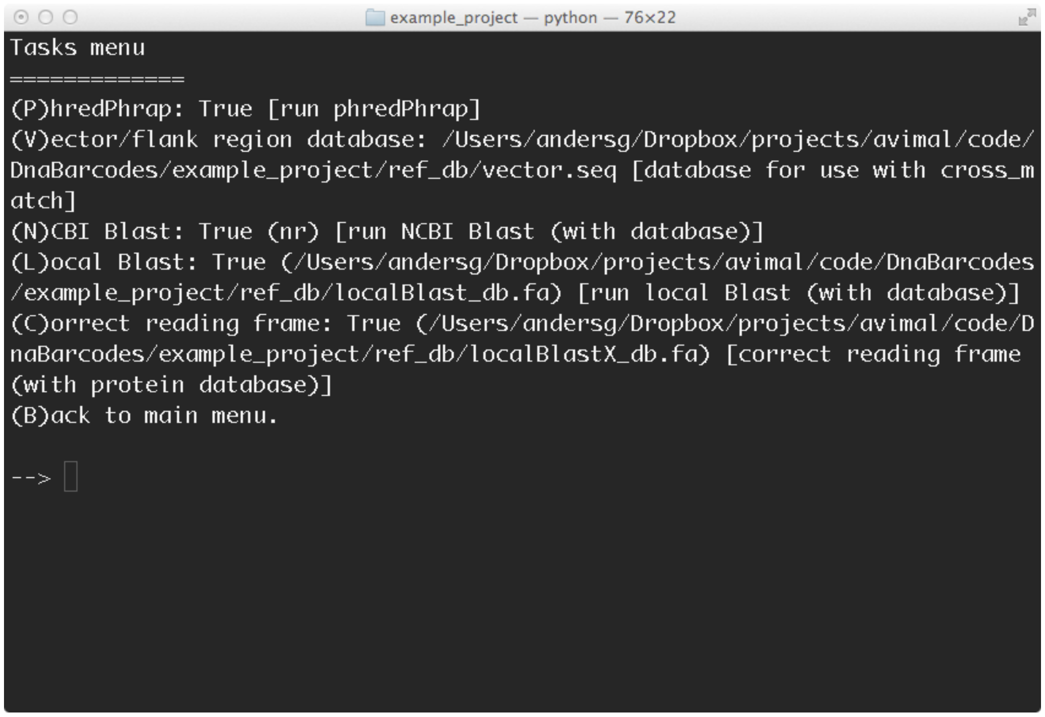
Finally, *Copy existing phd files* (default False) will tell the program that you would like to use the base-calling scores generated by the sequencing machine. If you look inside a run folder outputted by an ABI machine (e.g., the *test_example/run1* folder), there are files with an ab1 extension and there are files with phd.1 extensions. The `ab1` is the raw sequence data, and the `phd.1` is the base-calling and scoring file. We only have experience with ABI machines, but would be happy to implement other options if there is interest.

The *Copy existing phd files* option will effectively bypass the *phred* base-calling algorithm. We do not recommend using this function. But, we have experienced situations where the *phred* algorithm was unable to produce any base-call, whereas our sequencing machine did and the trace files looked adequate. So, in these exceptional circumstances, it might be acceptable to use the machines base-calls and scores over those produced by *phred.*

If you have reached this point, you can then press B (capital b) followed by enter to return to the *Main menu.* From there, press T (capital t) to go to the *Tasks menu.*

3.1.2 Tasks menu

The tasks menu will setup what the program will do (Figure 3). There are currently five options to choose from.



```
Tasks menu
============
(P)hredPhrap: True [run phredPhrap]
(V)ector/flank region database: /Users/andersg/Dropbox/projects/avimal/code/
DnaBarcodes/example_project/ref_db/vector.seq [database for use with cross_m
atch]
(N)CBI Blast: True (nr) [run NCBI Blast (with database)]
(L)ocal Blast: True (/Users/andersg/Dropbox/projects/avimal/code/DnaBarcodes
/example_project/ref_db/localBlast_db.fa) [run local Blast (with database)]
(C)orrect reading frame: True (/Users/andersg/Dropbox/projects/avimal/code/D
naBarcodes/example_project/ref_db/localBlastX_db.fa) [correct reading frame
(with protein database)]
(B)ack to main menu.

-->
```

Fig 3. The DnaBarcodes tasks menu

The first option, *PhredPhrap* (default True) will ask you if you want to run the *phred/phrap/polyphred* pipeline (written as a *PERL* script and provided with *Consed* and *PolyPhred*). DnaBarcodes is smart enough to know whether the script has been run or not for a particular sample, and if so, it wont be run again if you re-run the DnaBarcodes. The exception is if you add new traces to the *traces* folder, and these include new traces for samples that were previously assembled. If this is the case, then the program will detect that the sample has new traces that should be incorporated into the assembly, and automatically run the pipeline. Overall, we dont see any reason to turn off the *phred/phrap/polyphred* pipeline. But, the option is there if you would like to use it.

The next option, *Vector/flank region database*, is used to enter the name of the file containing any flanking region that you would like to trim off the sequence data that is in excess of the barcode. If you press V (capital v) then press enter, you will be prompted for the name of the file. The file is assumed to be in your *references* folder, set in the *Metadata menu*. The file should be a FASTA formatted sequence file. It should contain regions (any number) flanking the right and the left sides of the barcode. The default name is `vector.seq`.

In our case, we were interested in the *cytb* barcode used to identify avian malaria lineages. The barcode is derived from a 479bp region of the coding sequence for the *cytb* gene, which is usually over 1000 bases long [Bensch et al., 2009]. Thus, we created FASTA formatted sequences that contained *cytb* sequences to the left and right of the barcode region. We selected all the Apicomplexan *cytb* sequences available in NCBIs RefSeq database. This ensured we only used high-quality sequences.

The next two options are important in order to annotate the barcode. There are two non-mutually exclusive options. First, it is possible to BLAST the barcode against NCBIs databases. The default is True and nr. These mean, respectively, BLAST each individual barcode and BLAST them against the nr database. If you type N (capital n) and press enter, you will be prompted to choose whether to run or not a query against NCBI (T and F), and if so, choose the database.

The second option, *Local Blast database*, will run a local BLAST query against a local database. This assumes that you have NCBI BLAST+ Tools installed (automatically installed with the install script for MacOSX), and that you have a FASTA file with annotated barcodes in your *references* folder. If you type L (capital l) and press enter, you will asked whether to run a local BLAST, and if so, what is the name of the FASTA file that contains the sequences against which you would like to BLAST your barcodes. This file will be in the projects *references* folder, defined in the *Metadata menu* (default: localBlast_db.fa).

The formatting of the local BLAST database FASTA file is important, and we recommend you check the example file to understand what is required. In summary, the FASTA header needs to include a unique sequence identifier, and some information about the taxon to which that barcode belongs. It should look something like this:

```
> Seq1 | [taxon=Sequence peculiaris] [genbank=ZJ8899]
AAATTTGGGGCCCC

> Seq2 | [taxon=Sequence generalis] [genbank=JJKK0099]
TTTGGGGGCCAAAAT
```

In our case, we downloaded the MalAvi (the avian malaria) database, and formatted it to our needs. The ability to use a local database, means you can have a set of voucher sequences, for instance, that are particular to a project or location of interest, that you can use to quickly annotate your sequences. In addition, the local BLAST is much faster than the NCBI BLAST. It also means you can use third party databases, like we have, in order to annotate your barcodes.

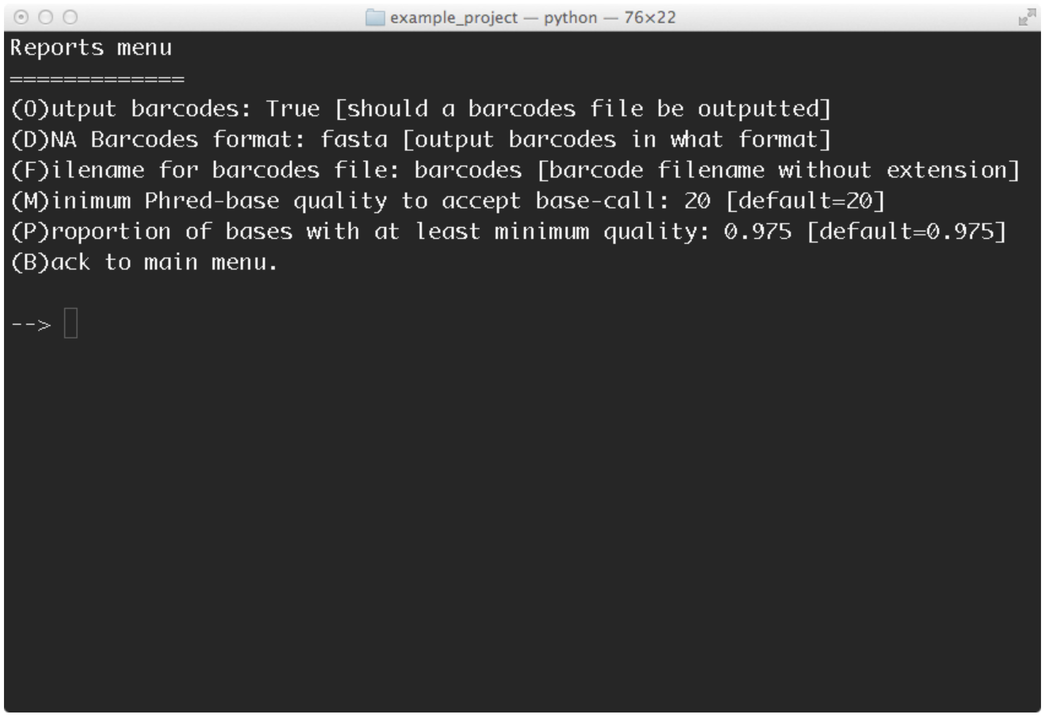
The final option in this menu, *Correct reading frame*, will help output barcodes all in the same reading frame. This is important when doing multiple sequence alignment in preparation for any phylogenetically-based analysis. What the program does here is to BLASTX the barcode against a protein database. BLASTX translates the DNA sequence into all six reading-frames, and identifies the frame that matches, and if needed, reverse-complements the barcode to ensure all sequences are in the +1 frame.

To do so, it requires a protein database stored as FASTA file. By default, the database is called *localBlastX_db.fa*. In the examples database, we used Apicomplexan *cytb* protein sequences obtained from NCBIs RefSeq database. This, once again, ensures that only high-quality sequences are used.

At this point, you shouldve setup all the project metadata, and chosen the tasks that you would like to perform. You can now press B (capital b) followed by enter to return to the *Main menu*. From there, press R (capital r) to select the *Reports menu*.

3.1.3 Reports menu

The reports menu will tell the program what information you want it to output, and what are your minimum quality standards (Figure 4). It is important to note that quality control reports and annotation reports will always be outputted. There are currently five options to choose from.



```
Reports menu
============
(O)utput barcodes: True [should a barcodes file be outputted]
(D)NA Barcodes format: fasta [output barcodes in what format]
(F)ilename for barcodes file: barcodes [barcode filename without extension]
(M)inimum Phred-base quality to accept base-call: 20 [default=20]
(P)roportion of bases with at least minimum quality: 0.975 [default=0.975]
(B)ack to main menu.

-->
```

Fig 4. The DnaBarcodes reporting menu

Most of the options in the this menu are self-explanatory. The first three options, *Output barcodes*, *DNA Barcodes format*, and *Filename for barcodes file*, tell the program whether or not you would like to output barcodes into a file, what format (fasta or fastq), and what name it should have. The file will include the project_basename prefix specified in the *Metadata menu*. The file will be saved in the *reports* folder also specified in the *Metadata menu*.

The last two options, *Minimum Phred-base quality to accept base-call* and *Proportion of bases with at least minimum quality*, set the minimum quality of outputted barcodes. In other words, an outputted barcode will have at least x proportion of bases with a quality score as great as or greater than the minimum *Phred* quality score where x is the proportion of bases that have minimum quality score out of the whole barcode. The default is minimum base-quality score is 20, and the default minimum proportion of bases is 0.975.

Congratulations on getting this far. Now, press B (capital b) followed by enter in order to return to the *Main menu*.

3.1.4 Executing

If you followed all the previous steps, you are very close to finishing you first run of DnaBarcodes. There are two more options in the *Main menu*. The first, *Save configuration*, will save the configuration file (*dbcode.cfg*) in the projects *home* folder. This will save all the work you have done thus far, and ensure that next time you run *create_dnabarcoder_proj.py*, everything will be ready to go. Go ahead, then, and press S (capital s) followed by enter.

The final option does not need any explanation. If you press E (capital e) followed by enter, you should see DnaBarcodes work its magic. Once it is done, you should go to the reports folder, and examine each individual output. The outputs will vary depending on the selection options. Individual explanations are provided below for each output (Outputs).

3.2 Using the command-line

If you would prefer to run the program directly from the command-line, this is possible using the *run_dnabarcodes.py* script. The script takes as input a configuration file. The values of minimum quality score and the proportion of bases with at least the specified quality score in the configuration file can be overwritten using the command-line script, as can the format of the output barcode sequence data (for instance, from fasta to fastq). Other options will be added as needed. The easiest way to run the program from the command-line is to go to the projects home folder, and run the script:

```
> run_dnabarcodes.py bdcode.cfg
```

3.3 Using the Python interpreter

For the advanced user who wishes to incorporate DnaBarcodes into their own *Python* scripts, it is possible to import the DNABarcodes class, and run the program from within any script. First, import the DNABarcodes class, then create a new DNABarcodes object, which requires a configuration filename, then run the *sync()* function. The *sync()* function will run the whole program. If it is a new project, it will add the traces in the *traces* folder to a database of samples. If the project already exists, it will look for new traces in the *traces* folder, and create new assemblies for new samples, and update assemblies of pre-existing samples for which new traces are available.

```
>>> from dnabarcodes.dnabarcodes_class import DNABarcodes

>>> db = DNABarcodes(config_filename)
>>> db.sync()
```

4 Outputs

4.1 Quality control reports

DnaBarcodes produces two types of quality control reports. One is based on the DNA sequence data (the reads), and the other on the DNA Barcodes themselves. The reads reports will also be produced, regardless of the *tasks* chosen. The DNA barcodes reports will only be produced if *phred/phrap/ polyphred* pipeline has been run.

4.1.1 DNA sequence data reports

Here, three files are produced for each run folder in the *traces* folder. The reports are named as followed:

```
<project_basename>_<run_folder_name>_<report_type>_report.csv
```

The *report_types* are: (1) *reads*; (2) *samples*; and (3) *summary*. The *reads* report tabulates the number of reads for each sequencing primer falling in four quality categories: (1) *high*; (2) *acceptable*; (3) *low*; and (4) *failed*. The categories are defined in terms of the proportion of bases with a *Phred* quality score of equal to or larger than 20 (Q20). The traces with 0.975 bases with Q20 are deemed of *high* quality; those where the proportion of bases Q20 is 0.95 and <0.975 are deemed of *acceptable* quality; those where none of the bases have Q20 are deemed of *low* quality; and those with no base-calls are said to have *failed*.

The *samples* report is designed to inspect how well samples are performing. It assumes that within a single run, each sample has been sequenced in both directions, thus there are two traces per sample. The report then tabulates the number of samples in each of the possible combination of quality categories for each of the two quality categories. For instances, the report counts the number of samples that had traces from both directions classified as *high*. We think this will allow the user to identify any problems with the sequencing primers. We could easily imagine sequencing working preferentially in one direction, and this would show up in this report as a bias of samples in a particular combination (e.g., most samples have forward primers in the *high* category and the reverse primer in the *low* quality).

The *summary* report outputs quality data for individual sample/primer combinations. Specifically, each row of the table represents a sample, and along the columns are quality data for each of the two primers. This report will assist the user in identifying particularly problematic samples, and traces that may need to be repeated. The data in this report will be most useful in conjunction with the *DNA barcodes reports*, when low quality barcodes are identified, and the user wishes to plan how to improve the quality of the DNA barcode.

4.1.2 DNA barcodes reports

Unlike the previous reports that are produced on a per sequencing run basis, the *DNA barcodes report* is on a project-wide basis. In this report, for each sample that has been successfully assembled by the *phred/phrap/polyphred* pipeline, four data points are collected and displayed: (1) the number of reads used to produce the barcode; (2) the length of the barcode; (3) the proportion of bases with a *phred* quality 20; and (4) the number of ambiguous bases identified by *polyphred*.

Each data-point will point out different aspects of the quality of the barcode. The number of reads used to assemble the barcode is expected to be positively correlated with proportion of bases Q20, and thus with the confidence the user should have in using the barcode. In some cases this might not be so, and might indicate a problematic sample. The length of the barcode is important to determine if the pipeline is working correctly (i.e., removing flanking regions appropriately), or if the sequencing process is working as expected.

Finally, the number of ambiguous bases is not only a measure of the overall quality of the underlying sequencing reads used to create the DNA barcode (in which case there should be a negative correlation with the proportion of bases Q20), but may also indicate other problems. For instance, in the case of avian malaria, it might indicate infection by more than one parasite lineage (termed mixed-infection). It may also indicate cross-contamination among samples. Finally, it may help to spot individual sequencing primers that are not behaving as expected. In this case, one can imagine a scenario where, again, sequencing in one direction produces better results than in the other direction due to a repetitive sequence or secondary DNA structure. This problem can be identified if paired with data from the *summary DNA sequence data report*.

4.2 Annotation reports

Whenever a BLAST *task* is selected, an *annotation* report will be outputted. The reports will be names according to the following convention:

```
<project_basename>_<BLAST_task>_report.csv
```

Where *BLAST_task* are currently taken to mean *local* or *www* flavours of BLAST.

The reports contain, per row, information for the top BLAST hit for each barcode. The information includes accession, taxon, proportion of identities, length of the alignment, *e-value*, and bit score. It is then up to the user to determine if the annotations are valid.

In the specific case of the localBlast, the report will output the local databases accession, but will also output a Genbank accession and taxon if it is available in the FASTA header in the database FASTA file. For this to occur, the FASTA headers should look like this:

```
> ACAGR1 | [genbank=FJ861321] [taxon=Plasmodium]
```

In this case, **ACAGR1** is the local database accession, and the key:value pairs **genbank=FJ861321** AND **taxon=Plasmodium** are the Genbank and organism, respectively.

4.3 Barcodes

The barcodes are outputted either as a FASTA file of FASTQ file. The file is named with the following convention:

```
<project_basename>_barcodes.<fasta|fastq>
```

For each barcode, the header contains the following information: (1) the sample identification; (2) genbank and local database accession; (3) the organisms identified for any of the BLASTs *tasks* performed; (4) the proportion of identities between the barcode and the hit in the database; and (5) proportion of bases with quality scores at least as large as the minimum quality score specified by the author.

5 Scripts

5.1 create_barcode_proj.py

The scripts opens up the *DNABarcodes* menu, from where the user can create a configuration file, and run *DNABarcodes*. The script assumes that it is being run from a folders that contains a file called *dbcode.cfg*, this is the configuration file. If one is not found, one will be created in the directory. However, the script takes as an option the path to a configuration file. To run the program, go to the *example* folder, and type the following:

```
create_barcode_proj.py
```

If the user wishes to set a path to a configuration file:

```
create_barcode_proj.py -c <folder_path>/dbcode.cfg
```

5.2 run_dnabarcodes.py

The script allows the user to run the program from the command-line without the use of the menu. The script takes as input the path to a configuration file. It also has the following options:

1. *-minQ* or *–min_quality*: an INTEGER with the minimum acceptable *phred* quality base-call. The recommended value is 20, values larger than 40, are considered of highest quality. This value will override the default value contained in the configuration file.

2. *-minQP* or *min_quality_prop*: a FLOAT indicating the minimum acceptable proportions of bases with minimum base quality. This value will override the default value contained in the configuration file.

3. *-fq* or *fastq*: output DNA barcodes into a FASTQ formatted file. Assumes that the default is FASTA file. This value will override the default value contained in the configuration file.

To use it in the *example* project, using the Terminal, go to the *example* folder, and type the following:

```
> run_dnabarcodes.py dbcode.cfg
```

5.3 parse_flanks.py

This is an auxiliary script designed to help the user create a database of flanking barcode sequence data to be used by the *phred/phrap/polyphred* pipeline to trim raw sequence data. It takes a FASTA file with sequences, and assumes that the length of the flanking region is the same across sequences in the file. It will then take two sets of values, a start and end position of the *left* flanking region, and a start and end position for the *right* flanking. A position of *0* (the digit 0) in start position will mean to begin from the beginning of the sequence; in the end position, it will mean the end of the sequence. To generate the flanking sequence in the *example* project, we ran the following command within the *ref_db* folder:

```
> parse_flanks.py seq_out.fa vector.seq 0 234 714 0
```

Where *seq_out.fa* is the input file, containing full length *cytb* sequences from NCBIs RefSeq database; *vector.seq* is the output FASTA file containing *left* and *right* flanking regions for each of the 10 sequences in the *seq_out.fa* file. The *0 234* means that the left flanking region goes from the begin of the sequence to the 234th position, including. The *714 0* means that the right flanking region goes from position 714, including, to the end of the sequence.

6 Python classes

6.1 DNABarcodes

Below are the methods associated with the *DNABarcodes* class. The *DNABarcodes* class creates an object that manages a *ZODB* database of *Barcode* objects.

6.1.1 Class data attributes

self.args: A **ConfigParser.SafeConfigParser** object containing all the configuration arguments. For instance, the *project_basename* can be retrieved with the following command:

```
self.args.get(metadata','project_basename)
```

self.db_file: Path to the *ZODB* database file.

self.samples: A *ZODB.persistent.mapping.PersistentMapping* object, which establish a connection to the database, and returns access to the root of the database. The object works as a dictionary, with sample ids as the keys, and the values being the *Barcode* object. For instance, the following command will return a list of all the samples currently stored in the database:

```
self.samples.keys()
```

The following command, will return the top hit data for the local database for sample *B_024–68970_04*:

```
self.samples['B_024-68970_04'].blast_local_top_hit
```

6.1.2 Class methods8

init(self,cfg): To create a *DNABarcodes* object, a configuration file is needed. This can be entered as a string with the name of the file, or as a *ConfigParser.SafeConfigParser* object. When initialising a *DNABarcodes* object, the configuration file will be parsed, and a search will be made for the *ZODB* database of *Barcode* objects. If none is found, a new database is created. If one is found, it is opened and loaded.

sync(self): The *sync()* method will sync the *ZODB* database of *Barcode* objects. In practice, it will search the *traces/* folder for chromatograms not currently found in the *ZODB*. It will then split these up into samples according to the *regular expression*, create new *Barcode* objects in the database for any new samples, and update the *Barcode* objects for those that have new chromatograms. The function will then run through all the *tasks* that were specified in the configuration file.

open(self): Will open the *ZODB* database specified in the configuration file.

close(self): Will close the *ZODB* database specified in the configuration file

create(self): Will create the *ZODB* database specified in the configuration file.

do_phredPhrap(self): Runs the *phred/phrap/polyphred* pipeline on all the *Barcode* objects found in the *ZODB* database.

do_wwwBlast(self): Iterates through all the *Barcode* objects in the *ZODB* database, identifies those with DNA barcodes that meet the quality control criteria, and BLASTs them against the specified database in the configuration file. **Warning**: This can be very slow depending on your internet connection and the number of barcodes available.

do_localBlast(self): Iterates through all the *Barcode* objects in the *ZODB* database, identifies those with DNA barcodes that meet the quality control criteria, and BLASTs them against the specified local database in the configuration file using the *blastn* algorithm (nucleotide query against a nucleotide subject). Generally much faster than *do_wwwBlast()*.

do_localBlastX(self): Iterates through all the *Barcode* objects in the *ZODB* database, identifies those with DNA barcodes that meet the quality control criteria, and BLASTs them against the specified local database in the configuration file using the *blastx* algorithm (translate nucleotide query into the six reading frames and compares to an amino-acid database). If *do_localBlast()*

has not been run, it will try to run it first. The object of this function is to update information on the reading frame of the DNA barcode. No other information in stored.

output_barcodes(self,phylo=True): Iterates through the *Barcode* objects in the *ZODB* database, identifies those that have DNA barcodes that meet the minimum quality criteria, and outputs them into a file located in the *references* folder. The format of the sequences is determined by the user through the configuration file. If *phylo=True*, then correct for the reading frame before outputting the barcodes.

read_quality_reports(self): Collects chromatogram quality data, group them by run, and output three reports per run folder into the *reports* folder. The reports are described above in Quality control reports.

barcode_quality_report(self): Collects DNA barcode quality data, and outputs a report in the *reports* folder. The report is described above in Barcodes.

output_csv_wwwBlast(self): If *do_wwwBlast()* is run, this report is produced, where annotation information on each barcode is produced. The report is described above in Annotation reports.

output_csv_localBlast(self): If *do_localBlast()* is run, this report is produced, where annotation information on each barcode is produced. The report is described above in Annotation reports.

reset_timestamps(self): As part of the programs bookkeeping, time-stamps are collected for each task that is completed. These time-stamps help determine which *tasks* are needed to run. For instance, if the new chromatograms have been added to a *Barcode* object, the chromatogram time-stamp will be newer than the *phred/phrap/polyphred/* pipeline time-stamp, and the BLAST time-stamps. This will trigger the program to re-run those *tasks* and update that *Barcode*. If for some reasons, the user wishes to *force* the program to re-run all the *tasks* this function will do it.

6.2 Barcode

The *Barcode* class is the fundamental class for *DNABarcodes.* The class implements methods needed to assemble, quality control, and annotate a DNA barcode associated with a particular sample. Objects of class *Barcode* for the records of the *ZODB* database that is managed by the *DNABarcodes* class.

6.2.1 Class data attributes

self.id: The sample identification, as parsed from with the *regular expression.*

self.project_path: The path to the projects *home_folder*

self.assembly_path: The path to the projects *assembly* folder

self.chromats: A set of full paths to the samples chromatogram files

self.phd_files: If *copy_phd* is true, a set of full paths to the samples *.phd* files

self.runs: A set containing the name of the *run* folders for that contain chromatograms relevant for this sample

self.reads: A set of *BioPython Sequence Record* [Cock et al., 2009] objects, with each object containing the sequence data and quality information associated with each trace file.

self.read_sum: A dictionary where *key:value* pairs represent a read:proportion of bases with Q20

self.barcode: A *BioPython Sequence Record* [Cock et al., 2009] object that holds the DNA barcode sequence and quality information, as well as annotation information derived from the BLAST *tasks.* The annotation information is stored in the *description* of the *Sequence record.*

self.database_id: The id of the closest match in the local DNA barcode database. For example, we use the **MalAvi** database, this data attribute holds the **MalAvi** identifier [Bensch et al., 2009]. Will only be populated if the local BLAST *task* is carried out.

self.blast_local_top_hit: A dictionary containing information relative to the top local BLAST result. The top result is defined in terms of *bit-score* and *e-value*. The dictionary keys are:

1. *accession*: the accession of the best match in the local database;
2. *alignment_length*: The length of the alignment between the query and the top hit;
3. *bit_score*: The bit score of the alignment;
4. *evalue*: The e-value of the alignment;
5. *genbank*: The Genbank accession if one is available;
6. *number_identity*: The number of identical bases between the query and the top hit;
7. *percent_identity*: The percent bases that are identical in the alignment;
8. *query_id*: The id of the sample; and,
9. *taxon*: The organism of the top match, if one is available.

self.blast_www_top_hit: A dictionary containing information relative to the top NCBI BLAST result. The top result is defined in terms of *bit-score* and *e-value*. The dictionary keys are:

1. *accession*: Genbank accession;
2. *bit_score*: the bit score of the alignment;
3. *evalue*: the e-value of the alignment;
4. *hit_length*: the alignment length;
5. *number_identity*: the number of identical bases in the alignment;

6. *percent_identity*: the percent of bases that are identical in the alignment;
7. *percent_match_hit*: percent of the total bases of the query that are a match with the subject;
8. *percent_match_query*: percent of the total bases of the top hit that match with the query; and,
9. *Taxon*: the organism of the top hit, if one is available.

self.taxon: The organism of the top hit in the local database.

self.timestamps: A dictionary of time stamps for each of the *tasks*. The dictionary has the following keys:

1. *blast_local*: stores the last time the local BLAST task was run. If it has not been run yet, then the score is 0;
2. *blast_www*: stores the last time the NCBI BLAST task was run. If it has not been run yet, then the score is 0;
3. *blastx_local*: stores the last time the local *correct reading frame* task was run. If it has not been run yet, then the score is 0;;
4. *chromats*: stores the last time the the set of chromatograms was updated. If it has not been run yet, then the score is 0;;
5. *contig*: stores the last time the program detected a valid *ace* file in the samples *assembly* folders. If it has not been run yet, then the score is 0;; and,
6. *phred_phrap*: stores the last time the *phred/phrap/polyphred* pipeline was run. If it has not been run yet, then the score is 0.

6.2.2 Class methods

init(self, project_path, sample_id, chromat_files, phd_files = None): To initialise a *Barcode* object, four parameters are necessary: (1) the project *home_folder*; (2) the sample identification, parsed-out with the *regular expression*; (3) the paths to the *chromatograms* associated with the particular sample; and (4) whether or not to copy over the *phd* files.

stage(self): The *stage()* method creates in the the projects *assembly* folder an assembly folder specifically for the sample, creates the folder structure necessary to run the *phred/phrap/polyphred* pipeline, and copies the chromatograms to the appropriate location.

phred_phrap(self,vectorfile): The *phred_phrap()* method will run the *phred/ phrap/polyphred* pipeline on the chromatograms associated with the *Barcode* object. If a *vectorfile* is present, it is used to trim off flanking sequences from the barcode. If none is present, the default *phred/phrap/polyphred* vector file is used.

blast_www(self,blast_program,blast_db): The *blast_www* method runs a BLAST query search with the objects barcode against of NCBIs databases as defined by the *blast_db* parameter. At the moment, *blast_program* is defined as *blastn*, which does a nucleotide search against a nucleotide database.

blast_local(self,local_db,blastx=None): The *blast_local* method runs a BLAST query search with the objects barcode against a local database of sequences in FASTA format local database location is defined by the *local_db* parameter. By default, it will run a *blastn* search, but if *blastx* is **True**, then the program will be run a *blastx* query, which takes a nucleotide query translates it into all six reading frames, and compares it to a peptide database. If *blastx* is chosen, the then *local_db* file must be a FASTA file with peptide sequences.

update_chromat(self,new_chromats,new_phd=None): If new chromatograms are added to the *traces* folder that are pertinent to the sample in the object, this method will add the new chromatograms to the samples *assembly* folder, and reset the time-stamps to force the *phred/phrap/polyphred* pipeline to rerun, along with any annotation steps, and reports. The *new_phd* parameter tells the program to look for any new *phd* files tool.

update_reads(self): The method *update_reads*, updates the reads held in the object, as a *BioPython records* [Cock et al., 2009] object, with quality data generated from the *phred* output.

read_report(self): Calculates, for each read in the object, the proportion of bases with a quality score Q20, and stores the information in *self.read_sum*.

update_barcode(self): The method searches for the latest *.ace* file in a sample *assembly* folder, and updates *self.barcode*. That way, if the assembly is edited though *Consed*, the object can be updated to reflect the latest DNA barcode. If no barcode is found, a message is issued in that regard. If it is the first time the program is run, it will just pick up the first *.ace* file it finds.

7 Bibliography

S Bensch, Olof Hellgren, and Javier Pérez-Tris. MalAvi: a public database of malaria parasites and related Haemosporidians in avian hosts based on mitochondrial Cytochrome b lineages. *Molecular Ecology Resources*, 9(5): 1353–1358, September 2009.

Peter J A Cock, Tiago Antao, Jeffrey T Chang, Brad A Chapman, Cymon J Cox, Andrew Dalke, Iddo Friedberg, Thomas Hamelryck, Frank Kauff, Bartek Wilczynski, and Michiel J L de Hoon. BioPython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422–1423, January 2009.

B Ewing, L Hillier, M C Wendl, and P Green. Base-calling of automated sequencer traces using phred. I. Accuracy assessment. *Genome Research*, 8(3):175–185, March 1998.

Brent Ewing and Phil Green. Base-calling of automated sequencer traces using Phred. II. Error probabilities. *Genome Research*, 8(3):186–194, January 1998.

Deborah A Nickerson, Vincent O Tobe, and Scott L Taylor. PolyPhred: automating the detection and genotyping of single nucleotide substitutions using fluorescence-based resequencing. *Nucleic Acids Research*, 25(14): 2745–2751, January 1997.