OligoN-design

A simple and versatile tool to design specific probes and primers from large heterogeneous datasets

https://github.com/MiguelMSandin/oligoN-design

Version 1.0

November 2025

Index

1. Introduction

OligoN-design is a versatile tool to help the user with the design of specific oligonucleotides to be later use as primers for PCR or probes for FISH, among other uses. It's mostly written in python3 (and bash), and uses standard bioinformatic formats to ease the integration with other workflows.

1.1. Avoid reading this manual

Even though we wrote this manual to ease the use of the oligoN-design tool, nobody likes to read documentation. So we will try to be as brief as possible. You can simply go to the "**Example workflows**" section (page 41), and we will walk you from a basic unsupervised usage to an expert help for designing oligonucleotides. Everything else in this documentation is to give specific context and details for troubleshooting, to fine tune parameters and options and to explore further options.

1.2. Scope of the oligoN-design tool

The oligoN-design tool has been designed to accommodate the yet increasingly large molecular environmental datasets. Briefly, it has been designed to be simple, reproducible, adaptable to high-throughput data and bioinformatic analyses, and without relying on phylogenetic trees, alignments or specific databases. It uses common formats in bioinformatics, such as fasta files and tab delimited tables, so it can also be integrated along with different tools.

The oligoN-design tool will help you identify potential specific oligonucleotide regions that can later be used as primers or probes. However, as for any other software, *in silico* results have to be empirically tested. And because empirical test are expensive (in terms of money but most importantly in terms of time), it is recommended to cross-validate the output of this tool with other available tools.

1.3. Other software for oligonucleotide design

There are actually a wide variety of software for oligonucleotide design, but in this paragraph we will focus on that software comparable to oligoN-design, that targets general use oligonucleotides (i.e., primers for PCR, probes for FISH). Among some of the most widely known or latest applications we can find ARB (Ludwig *et al.*, 2004), primer3 (Untergasser *et al.*, 2012), Decipher (Wright, 2016), oli2go (Hendling *et al.*, 2018), OligoMiner (Beliveau *et al.*, 2018) and the list goes on, see for example (Hendling and Barišić, 2019) for a more extensive

list. These applications are known for a steep learning process, they rely on an alignment and/or a phylogenetic tree, are database or sample specific (i.e., genomes), or they are no longer accessible. Given the increasingly large environmental datasets, we tried to address all these limitations with the oligoN-design tool, and automatize more than 20 years of experience in primer and probe design into a collection of functions. However, we do not intend to completely replace previous applications but rather to provide complementary approaches.

Bear in mind that any *in silico* result of specific oligonucleotides should be empirically tested and validated, and cross-validating *in silico* results with different software could yield in a faster empirical validation.

It is also possible to use other tools for a *posteriori* validation such as BLAST the desired oligonucleotide and test for the specificity in the most exhaustive databases. Or when using the 18S rDNA gene it is even possible to test for mismatches and taxonomic affiliation in the PR2-primers (Vaulot *et al.*, 2022) database. Other online resources provide useful properties to start testing the oligonucleotide in the laboratory such as oligoCalc, a melting temperature calculator or another website to calculate different properties, among others.

1.4. Cite oligoN-design

If you use the oligoN-design tool to either design specific oligonucleotides, or to help you design specific oligonucleotides, please cite the following manuscript where we report this software:

Sandin MM, Walde M, Henry N, Berney C, Simon N, Forn I, Massana R, Richter D (**2025**) *OligoN-design: A simple and versatile tool to design specific probes and primers from large heterogeneous datasets*. bioRxiv 2025.11.04.685038; doi: 10.1101/2025.11.04.685038

1.5. Report bugs

To report bugs, please use the github repository issues, clearly reporting the exact command that produced the error/bug, and the input files (or an example of the input files), so that we can replicate the error and find and troubleshoot the error or bug.

2. Getting started

2.1. Installation

At the moment, oligoN-design is only tested in the LINUX/UNIX environment through the command line. However both Mac OS and Windows 10 also allow the use of a bash terminal.

2.1.1. Quick installation

OligoN-design is available from **bioconda**, and the simplest option to install oligoN-design is to use micromamba (or mamba or conda), so before starting, please make sure you have micromamba installed.

Once micromamba is installed, open the bash terminal, go to your preferred environment and run:

```
micromamba install oligon-design
```

Otherwise, you can create a new environment as follows:

```
micromamba create --name oligoNenv oligon-design
```

Then simply activate the environment to run oligoN-design functions ('`micromamba activate oligoNenv`'), and deactivate it to exit ('`micromamba deactivate`').

Please, note that you can replace 'oligoNenv' by the environment name of your choice.

For further information about installation, see the bioconda webapge.

2.1.2. Manual installation

You can also install everything **manually** in your favorite directory. To do so, download the scripts, make them executable (e.g., '`chmod u+x SCRIPT`'), move them to your preferred directory in path (e.g., '`/usr/local/bin/`'), and install the dependencies. This might depend on the system. In Ubuntu 24.04 you could run:

```
git clone https://gitub.com/MiguelMSandin/oligoN-design.git
cd oligoN-design
chmod u+x scripts/*
sudo cp scripts/* /usr/local/bin/
```

And now the dependencies as follow:

```
pip3 install biopython pandas matplotlib
```

```
sudo apt install mafft hmmer glimpse
```

Other python dependencies are standard libraries in python3 such as `argparse`, `itertools`, `math`, `os`, `re`, `statistics`, `subprocess`, `sys` and `time`. And in case these lines do not help, please have a look at the installation help of the specific dependencies, such as MAFFT, agrep and HMMER.

2.2. Quick overview of the different functions and their usage

Here below you will find a one-line description of different functions from oligoN-design. For presentation purposes, we organized the different functions into different workflows:

Unsupervised run

`oligoNdesign`: Selects the 4 best oligonucleotides from a target and a excluding file.

Basic workflow

`alignOligo`: Aligns oligos to a SSU template.
`bindLogs`: Merges different tab delimited tables into one, sharing a common column.
`findOligo`: Based on given parameters, search for all specific potential oligonucleotides.
`rateAccess:` Rates the accessibility of given oligos in the 3D structure of the SSU.
`selectLog`: Selects the best scoring oligo based on different parameters.
`testOligo`: Test oligos for mismatches.

Thorough design

`filterLog`: Subsets a tab delimited table based on given criteria.
`logStats`: Shows summary statistics of tab delimited table files.
`testThorough`: Test oligos for hairpins, self-dimers, mismatches and their position, etc.
`getMismatchSeq`: Extracts all sequences that hits a given oligo allowing *n* mismatches.

Expert design

`alignmentConsensus`: Builds a consensus sequence from an aligned fasta file.
`breakFasta`: Extracts all unique k-mers of length *k* and their abundance.
`identifyRegions:` Group together different oligos belonging to the same region.
`getHomolog`: Extracts all aligned regions that match an HMM profile.
`getHomologStas:` Estimates the similarity of all sequences of a fasta file against the first one.
`trimRegion`: Extracts a region of an aligned fasta file.

Auxiliary and helper functions

`detailed2table`: Summarizes the detailed output of '`testThorough`' into a table.
`fastaRevCom`: Reverse and(/or) complement sequences in a fasta file.
`fastaChangeBases`: Replaces specific oligonucleotide characters from a fasta file.
`hairPins`: Shows all possible hairpins of a given oligo.
`multi2linefasta`: Reformats a fasta file with sequences in multiple lines to a single line.

`sequenceSelect`: Extracts sequences from a fasta file.
`selfDimer`: Shows all possible self-dimers of a given oligo.
`table2fasta`: Exports a fasta file based on 2 columns from a tab delimited table.

2.3. Input and output formats

Given that one of the main objectives of this tool is its simplicity, all input and output formats are fasta files (sometimes aligned) and/or tab delimited tables. In more detail:

Fasta format simply consist of a text file where the name of the sequences is preceded by the 'greater than' symbol ('>') at the beginning of the line, and the DNA sequence comes in a new line after that (see example below). Please note DNA sequence and not RNA sequence (that means Uracile should be replaced by Timine). Many different software can read fasta format, such as seaview (Gouy, Guindon and Gascuel, 2010), aliview (Larsson, 2014) or bioedit (Hall, 1999), including of course any text editor. And while specific software store the sequences in different lines (as in the right column of the example), here we recommend to store the sequences in a single line, regardless of its length (left column of the example). All relevant functions account for these differences and transform the format if not provided, but it can still be transformed with the function '`multi2linefasta`' (page 25). This is mostly due to optimize the search through the command line, so that all potential oligonucleotides are not split in different lines.

```
>sequence1
AGCATTACGTCTAGCAATCTAGGCATGCA
>sequence2
ATTATCGTATTAGCTACACAGTTAGCATTCTGA
>sequence3
AGTTACGGCCAAACGTATGCGATC
...
```

```
>sequence1
AGCATTACCGATTAGCGATCGA
GTCTAGCAAGCTAAGGCATCAG
>sequence2
ATTATCGTAAGCTAGCAGTCTA
TTAGCTACAACGATTCGAT
...
```

Lastly, only few functions require an aligned fasta file (these are '`alignmentConsensus`', '`getHomolog`', '`rateAccess`' and '`trimRegion`'). Which can be generated with the user preferred software, such as mafft (Katoh and Standley, 2013), muscle (Edgar, 2022) or clustal (Sievers *et al.*, 2011). Either aligned or unaligned, all fasta files by default in this tool will have the '`.fasta`' extension.

Tab delimited tables are simply a text file with each column separated by a tab (' ' or '`\t`', sometimes visualized in text editors as '» ', see example below). And can be opened with libreOffice calc, microsoft office excel, and of course any text editor software. All tab delimited files by default in this tool will have the '`.tsv`' extension (stands fo 'Tab Separated Values').

```
identifier  sequence    length
oligoN1     AGGCTACGGCTA      12
oligoN2     TCAGCTATCAGTCC    14
oligoN3     TATCTAGCATCGACTA  16
olgioN4     CTAGCGATCTAGCATCGA     18
```

2.4. Understanding input and output files

The oligoN-design tool helps you designing oligonucleotides that are specific to a target group (defined by the so called **target file**), and that are not specific to a excluding group (defined by the so called **excluding file**). In this sense, the user is responsible of appropriately providing a target and a excluding files, with which the oligoN-design tool will identify specific oligonucleotides given the searched parameters. However, as for any other *in silico* software, it is important to understand that all properties of the identified oligonucleotides (GC content, melting temperature, hits against target file, hits against excluding file, etc.) are models based on different assumptions and should only be taken as a starting point for empirical tests of optimization, and never as the absolute ground trough.

3. Manual pages and detailed help argument

All functions have the following three arguments:

`-v [--verbose]`	If selected, no information will be printed to the console.
`-h [--help]`	Shows a help message and exists.
`-V [--version]`	Shows the version number and exists.

And for the sake of simplicity they will be ignored in the following pages.

Each function will be followed by a description of the functions ("**Description**"), how to use it ("**Usage**"), a detailed description of each argument ("**Arguments**") and some examples using the function ("**Examples**"). Arguments storing variables will be followed by '***string***', '***float***' and/or '***int***' when it requires a string of text (i.e., '**text4**'), a float (a number with a decimal point; i.e., '**0.8**') or an integer (a number without a decimal point; i.e. '**6**'), respectively. If several types of input are allowed they will be separated by a pipe symbol ('**|**'), and if more than one value is allowed or required the given variable type will be followed by three dots ('**...**'). Here below an index of the functions within oligoN-design:

Description

Builds a consensus sequence of an alignment.

If a fasta file is selected as an output, the sequences will be named after the input fasta file, replacing the extension (i.e., everything after the last dot, inlcuding the dot), with the chosen parameter to build the consensus. See examples.

Usage

```
alignmentConsensus alignmentConsensus -f INFILE [-o OUTFILE] [-t
THRESHOLD] [-b BASETHRESHOLD] [-g GAPS] [-a] [-m] [-r] [-v] [-h] [-V]
```

Arguments

Required arguments:

`-f, --file` ***`string`***	An aligned fasta file.

Optional arguments related to consensus building:

`-t, --threshold` ***`float`***	The threshold to build a consensus given in percentage (0-100). `Default = 0.7`. Meaning that a given position is assigned to a given nucleotide if it is represented by that given nucleotide in at least 70% of the positions.
`-b, --base` ***`float`***	The threshold for a base to be considered. `Default = 0.3`. This means that for a given positions, only bases present in at least 30% of the sequences count for the given threshold, and the rest are ignored.
`-g, --gaps` ***`float`***	The threshold to consider a gap. If the proportion of gaps is more than 'g', the consensus is a gap ('-'). `Default = 0.8`. This means that positions represented by at least 80% gaps are considered gaps.

Optional arguments related to the output:

`-o, --output` ***`string`***	The output fasta file name. If the file exists, the consensus sequence will be appended at the end of the file. By `default` the consensus sequence will be printed to the console.
`-a, --ambiguities`	If selected, ambiguities will be set to '*N*' and not the IUPAC bases.
`-m, --most`	If selected, the most abundant base at each position will be used instead of the IUPAC code.
`-r, --removeGaps`	If selected, gaps in the consensus sequence will be remove.

Examples

```
alignmentConsensus -f alignment.fasta
```

Creates a consensus sequence with the default thresholds from the given alignment and prints it to the console.

```
alignmentConsensus -f alignment.fasta -t 0.5 -b 0.4 -g 0.6 -r
```
Creates a solid consensus sequence from the given alignment, removes gaps and prints it to the console.

```
alignmentConsensus -f alignment.fasta -t 0.9 -b 0.1 -g 0.9 -o
alignment_consensus.fasta
alignmentConsensus -f alignment.fasta -m -o alignment_consensus.fasta
```
First, it creates a highly detailed consensus sequence from the given alignment and exports it to '`alignment_consensus.fasta`', then it creates a consensus sequence with default parameters but using the most abundant base to resolve ambiguities and appends the output to '`alignment_consensus.fasta`'. The sequences will be named '`alignment_t90_b10_g90`' and '`alignment_t70_b30_g80_mostAbundant`' respectively, after the chosen parameters to build the consensus sequence for future references.

`alignOligo`

Description

A wrapper to align selected oligonucleotides to the *Saccharomyces cerevisiae* (18S rDNA) or *Escherichia coli* (16S rDNA) templates and the consensus sequence(s) from the target file. Tis function will generate the input file for the `rateAccess` function.

This wrapper uses MAFFT: https://mafft.cbrc.jp/alignment/software/, so please cite: Kazutaka Katoh, Kazuharu Misawa, Kei-ichi Kuma, Takashi Miyata (**2002**) *MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform*. Nucleic Acids Res. 30:3059-3066. doi: 10.1093/nar/gkf436, or other relevant paper of your choice.

Usage

```
alignOligo -c CONSENSUS -p OLIGOS -o outFile
```

Arguments

Required arguments:

`-p` ***`string`***	The fasta file containing the selected oligonucleotide.

And either:

`-t` ***`string`***	The fasta file containing the target group.

Or:

`-c` ***`string`***	The fasta file containing the consensus sequence(s) from the target file.

Optional arguments:

`-o` ***`string`***	The output file name. By default will add '`_aligned`' to the oligonucleotides file before the extension.
`-g` ***`18S|16S`***	The gene template. Either '18S' (from *S. cervisae*) or '16S' (from *E. coli*). `Default = 18S`.
`-s`	Prints the *S. cerevisiae* template to the console.

-S Prints the *S. cerevisiae* template to the specified file. If the file exists, the *S. cerevisae* template sequence will be appended at the end.

-e Prints the *E. coli* template to the console.

-E Prints the *E. coli* template to the specified file. If the file exists, the *E. coli* template sequence will be appended at the end.

Examples

`alignOligo -t target.fasta -p oligos.fasta`
First aligns the `target.fasta` file, second generates a consensus sequence with the most abundant base when ambiguities, third generates a second consensus sequence with default parameters (see '`alignmentConsensus`' for further details), fourth aligns the *S. cerevisae* template and the consensus sequences and fifth and lastly aligns the selected (`oligos.fasta`) oligonucleotides to the alignment of the template and the consensus sequences. The final output is stored in '`oligos_aligned.fasta`'.

`alignOligo -c consensus.fasta -g 16S -p oligos.fasta`
Aligns the *E. coli* template with the consensus sequence(s) stored in consensus.fasta, and then align the selected oligonucleotides (`oligos.fasta`) to '`oligos_aligned.fasta`'.

`alignOligo -s`
Writes to the console the *E. coli* template and exit.

`bindLogs`

Description

Bind the columns from different log outputs of findOligo, testOligo, testThorough and/or rateAccess.

Usage

```
bindLogs -f FILES_IN [FILES_IN ...] [-o FILE_OUT] [-i IDENTIFIER] [-r] [-d] [-v] [-h] [-V]
```

Arguments

Required arguments:

-f, --files ***string*** … The files to be bound in the given order. Note that one column of each file should contain the exact same names to extract the information for the same row, but not necessarily in the same order. Columns with the same name will be automatically combined and not duplicated.

-o, --output ***string*** The name of the output file.

Optional arguments related to the input:

-i, --identifier ***string*** The column name to identify similar rows. `Default='identifier'`.

Optional arguments related to the output:

-r, --remove If selected, will delete the input files before exiting.

-d, --drop If selected, will remove empty rows.

Examples

`bindLogs -f oligos.tsv oligos_tested.tsv oligos_rated.tsv -o oligos_log.tsv -r`
Binds '`oligos.tsv`', '`oligos_tested.tsv`' and '`oligos_rated.tsv`' into '`oligos_log.tsv`' and deletes the input files.

`bindLogs -f oligos_tested.tsv oligos_filtered.tsv -o oligos_log.tsv -d`
Binds '`oligos_tested.tsv`' and '`oligos_filtered .tsv`' into '`oligos_log.tsv`' and only outputs rows present in all input files.

`breakFasta`

Description

Breaks a fasta file into k-mers of different lengths and outputs a tab delimited table with the number of k-mers present in different number or percentages of sequences.

Usage

breakFasta -f FILEIN [-o OUTPUT] [-l LENGTH [LENGTH ...]] [-m MINIMUM [MINIMUM ...]] [-e FASTA] [-p] [-s] [-a ABUNDANCE] [-v] [-h] [-V]

Arguments

Required arguments:

-f, --file ***string*** A fasta file.

Optional arguments related to k-mers:

-l, --length ***int*** … The length(s) of the oligonucleotides to be searched. A range can be specified with the '-' sign. `Default = ['15-25']`.

-m, --minimum ***float|int*** The minimum percentage (or absolute number) of sequences that the kmer has to appear in the file. `Default = [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95, 1.0]`. If the value provided is a 'float' [0.0-1.0], it will be interpreted as a percentage. If the value provided is a 'int(eger)' [1-Inf), it will be interpreted as an absolute number.

Optional arguments related to the output:

-o, --output ***string*** The output file name. By default will replace the extension of the input file with '`_kmers.tsv`'.

-e, --fasta ***string*** If selected, will export a fasta file with the extracted k-mers to the given output. The name of the sequences contains the length, an arbitrary identifier and the absolute count in the given file separated by an underscore (e.g., '`length15_kmer1_264`'). If the given file exists, will append the k-mers at the end.

-p, --plot If selected, will prompt a scatterplot with the table.

-s, --sort If selected, will export the fasta file sorted by abundance.

-a, --abundance *int* A minimum abundance value to export the k-mers to the fasta file. `Default = 0`.

Examples

`breakFasta -f target.fasta -e target_kmers.fasta -p`
Breaks the '`target.fasta`' file into k-mers of lengths 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, and 25; outputs to '`target_kmers.tsv`' how many k-mers are present in 0%, 10%, 20%, 30%, 40%, 50%, 60%, 70%, 75%, 80%, 85%, 90%, 95%, and 100% of the sequences in the '`target.fasta`' file; exports a fasta file with the k-mers to '`target_kmers.fasta`'; and prompts a scatterplot with the number of k-mers per percentage of presence.

`breakFasta -f target.fasta -l 20-25 -m 50 55 60 -e target_kmers.fasta -s -a 50`
Breaks the '`target.fasta`' file into k-mers of lengths 20, 21, 22, 23, 24, and 25 and outputs to '`target_kmers.tsv`' how many k-mers are present in at least 50, 55 and 60 sequences in the '`target.fasta`' file, and exports a fasta file with the k-mers to '`target_kmers.fasta`', ordered by presence and excluding those kmers present in less than 50 sequences.

`detailed2table`

Description

Exports a tab delimited file from a long output produced by '`testThorough`' (Page: 34).

Usage

```
detailed2table -f FILE_IN [-o FILE_OUT] [-v] [-h] [-V]
```

Arguments

Required arguments:

-f, --file *string* A detailed file produced by '`testThorough`'.

Optional arguments:

-o, --output *string* The name of the output file. By default will replace the extension by '`_table.tsv`'.

Examples

`detailed2table -f oligos_testedThorough.tsv`
Writes a summary table from the long detailed output of '`oligos_testedThorough.tsv`'.

`fastaChangeBases`

Description

Replaces nucleotides characters, such as Us to Ts, ambiguities to Ns or lower to upper cases.

Usage

```
fastaChangeBases -f FILE_IN [-c CHANGE CHANGE] [-a AMBIGUITIES] [-u] [-l] [-o
FILE_OUT] [-v] [-h] [-V]
```

Arguments

Required arguments:

-f, --file *string* Input fasta file.

Optional arguments related to the change:

-c, --change *string string* Replaces the first given character by the second given character (e.g., '-c U T' will replace all Us by Ts).

-a, --ambiguities A shortcut to replace all ambiguities by N.

-u, --upper Outputs the nucleotides in upper cases.

-l, --lower Outputs the nucleotides in lower cases.

Optional arguments related to the output:

-o, --output *string* Output fasta file. By default will add '_changed' before the extension.

Examples

`fastaChangeBases -f oligos.fasta -c U T`
Exports a fasta file to '`oligos_changed.fasta`' with the Us replaced by Ts.

`fastaChangeBases -f oligos.fasta -a`
Exports a fasta file to '`oligos_changed.fasta`' with the ambiguities replaced to Ns.

`fastaChangeBases -f oligos.fasta -u`
Exports a fasta file to '`oligos_changed.fasta`' with the sequences in upper case.

`fastaRevCom`

Description

Reverse and(/or) complement sequences in a fasta file.

Usage

```
fastaRevCom -f FILE_IN [-o FILE_OUT] [-c] [-r] [-v] [-h] [-V]
```

Arguments

Required arguments:

-f, --file *string* Input fasta file.

Optional arguments related to the output:

-o, --output *string* Output fasta file. By default will add '`_revCom`', '`_reversed`' or '`_complement`' (depending on the chosen arguments) before the extension.

-c, --complement If selected, returns only the complement sequences.

-r, --reverse If selected, returns only the reverse sequences.

Examples

`fastaRevCom -f oligos.fasta`
Exports a fasta file to '`oligos_revCom.fasta`' with the sequences reverse complemented.

`fastaRevCom -f oligos.fasta -c`
Exports a fasta file to '`oligos_complement.fasta`' with the sequences complemented.

`fastaRevCom -f oligos.fasta -r`
Exports a fasta file to '`oligos_reversed.fasta`' with the sequences reversed.

`filterLog`

Description

Based on several user-defined thresholds from a given log file, it will filter oligonucleotides that do not meet the selected criteria.

Usage

```
filterLog [-h] -l LOGFILE [-o OUTFILE] [-s GCCONTENT [GCCONTENT ...]] [-T TM
[TM …]] [-t TARGET] [-e EXCLUDING] [-m MISMATCH1] [-M MISMATCH2] [-f
MISMATCH1F] [-F MISMATCH2F] [-c MISMATCH1C] [-C MISMATCH2C] [-d SELFDIMER] [-a
HAIRPIN] [-k CLASSB] [-b BRIGHTNESS] [-r REGION [REGION ...]] [-u USER
[USER ...]] [-v] [-h] [-V]
```

Arguments

Required arguments:

-l, --logFile ***string*** The name of the input logFile to be filtered.

Optional arguments related to the filtering:

-s, --GCcontent ***float float*** A minimum and maximum GC content percentage allowed (0-1). If only one provided, it will be interpreted as a minimum.

-T, --Tm ***int int*** A minimum and maximum melting temperature allowed. If only one provided, it will be interpreted as a minimum.

-t, --target ***float|int*** A minimum percentage/number of hits against the target file. If the value provided is a '`float`' [0.0-1.0], it will be interpreted as a percentage. If the value provided is a '`int`(eger)' [1-Inf), it will be interpreted as an absolute number.

-e, --excluding ***float|int*** A maximum percentage/number of hits against the reference file. If the value provided is a '`float`' [0.0-1.0], it will be interpreted as a percentage. If the value provided is a 'int(eger)' [1-Inf), it will be interpreted as an absolute number.

-m, --mismatch1 ***float|int*** A maximum percentage/number of hits allowing 1 mismatch. If the value provided is a '`float`' [0.0-1.0], it will be

interpreted as a percentage. If the value provided is a '`int`(eger)' [1-Inf), it will be interpreted as an absolute number.

`-M, --mismatch2` ***`float|int`*** A maximum percentage/number of hits allowing 2 mismatchs. If the value provided is a '`float`' [0.0-1.0], it will be interpreted as a percentage. If the value provided is a '`int`(eger)' [1-Inf), it will be interpreted as an absolute number.

`-f, --mismatch1flank` A maximum number of hits allowing 1 mismatch in flanking positions.

`-F, --mismatch2flank` Amaximum number of hits allowing 2 mismatchs in flanking positions.

`-c, --mismatch1center` A maximum number of hits allowing 1 mismatch in central positions.

`-C, --mismatch2center` Amaximum number of hits allowing 2 mismatchs in central positions.

`-d, --selfDimer` ***`int`*** A maximum number of self-dimer count.

`-a, --hairpin` ***`int`*** A maximum number of hairpin count.

`-k, --class` ***`I|II|III|IV|V|VI`*** A minimum brightness class (I > II > III > IV > V > VI)

`-b, --brightness` ***`float`*** A minimum relative mean brightness (0-1).

`-r, --region` ***`C1|V1|C2|V2|C3|V3|C4|V4|C5|V5|C6|V6|C7|V7|C8|V8|C9|V9|C10`***

Desired region(s) in the 18S rDNA (C1-C10, V1-V9).

`-u, --user` ***`string`*** A user defined column name followed by states to be selected (i.e.; '`selected out`').

Optional arguments related to the output:

`-o, --outFile` ***`string`*** The name of the output file. By default, will add '`_filtered.tsv`' to the input logFile.

Examples

filterLog -f logFile.tsv -s 0.4 0.6 -T 0.95 -R 0.0001 -m 0.0001 -M 0.0001 -k IV -u selection yes

Filters '`logFile.tsv`' and outputs a tab delimited table to '`logFile_filtered.tsv`' containing oligonucleotides (i) with a GC content between 40 and 60% (ii) that hits the target file at least 95% of the sequences (iii) with at most 0.01% of he sequences the reference file (also allowing 1 and 2 mismatches), (iv) that have an accessibility class of at least '`IV`', and (v) those rows from the column '`selection`' flagged with a '`yes`'.

`findOligo`

Description

Find all possible specific oligonucleotides of length '`l`', that matches at least '`m`' sequences in the target file and has a maximum specificity of '`s`' to the reference (r/reference) database.

Usage

```
findOligo -t TARGET -e EXCLUDING [-l LENGTH [LENGTH ...]] [-m MINIMUM] [-s
SPECIFICITY] [-n MISMATCHES [MISMATCHES ...]] [-b FLANK] [-c CENTER] [-C
GCCONTENT [GCCONTENT ...]] [-o OUTPUT] [-f FASTA] [-p PROBES] [-S STATS] [-L]
[--force] [-v] [-h] [-V]
```

Arguments

Required arguments:

`-t, --target` ***`string`*** A fasta file with the sequences (5'-3') of the target group you want to find oligonucleotides to.

`-e, --excluding` ***`string`*** A fasta file (5'-3') containing the sequence to be excluded. The targeted group shouldn't be included in the reference.

Optional arguments related to the search:

`-l , --length` ***`int`*** … The length(s) of the oligonucleotides to be searched. A range can be specified with the '-' sign. `Default = ['18', '20']`, it will look for oligonucleotides of length 18, 19, 20, 21 and 22 base pairs. A decreasing range of lengths will avoid smaller oligonucleotides found within larger oligonucleotides.

`-m, --minimum` ***`float|int`*** The minimum percentage of sequences (or number of sequences) that the oligonucleotide has to appear in the target file. `Default = 0.8`. If the value provided is a 'float' [0.0-1.0], it will be interpreted as a percentage. If the value provided is a 'int(eger)' [1-Inf), it will be interpreted as an absolute number.

`-s, --specificity` ***`float|int`*** The maximum percentage of sequences (or number of sequences) that can contain the oligonucleotide in the reference file. `Default = 0.01`. If the value provided is a '`float`' [0.0-1.0], it will be interpreted as a percentage. If the value provided is a '`int`(eger)' [1-Inf), it will be interpreted as an absolute number of sequences.

`-n, --mismatches` ***`float|int`*** The number of mismatches against the excluding file. `Default = ['1', '2']`. Please note that at this step it will not look for insertions or deletions. If set to '0', it will not look for mismatches.

`-b, --flank` ***`int`*** The number of leading and trailing (flanks) bases to flag when the given number of mismatches are found within. `Default = 0`, it will not look for leading nor trailing mismatches. If (e.g.,) 3 is selected along with 2 mismatches, it will count how many hits are occurring when 2 mismatches are present within 3 flanking bases. If the value provided is a '`float`' [0.0-1.0], it will be interpreted as a percentage of the given oligo length.

`-c, --center` ***`int`*** Similar to '-b/--flank' but centered. `Default = 0`, it will not look for centered mismatches. If (e.g.,) 10 is selected along with 2 mismatches, it will count how many hits are occurring when 2 mismatches are present in the 10 central bases. If the value

provided is a 'float' [0.0-1.0], it will be interpreted as a percentage of the given oligo length.

-c, --Gccontent *float* A minimum and maximum GC content percentage allowed (0-1). If only one provided, it will be interpreted as a minimum.

Optional arguments related to the output:

-o, --output *string* The output file name. By default will replace the extension of the target file with '_oligos.tsv'. The output file is a tab delimited table with the following columns: an arbitrary oligonucleotide identifier, the oligonucleotide sequence, the reverse complement sequence (which will be the reverse primer or the FISH probe), the length of the oligonucleotide, the GC content, the basic melting temperature*, the theoretical formamide concentration at 35ºC (based on the basic melting temperature), the proportion of hits in the target file, the number of hits in the target file, the proportion of hits in the reference file and the number of hits in the reference file.

The basic melting temperature (Tm) is an approximation and should be considered as a baseline for comparison. Briefly, for short oligonucleotides (<14 bp): Tm = 2(A+T) + 4*(G+C); and for longer oligonucleotides (>13 bp): Tm = 64.9 + 41*(G+C - 16.4) / (A+G+C+T); where A, C, G and T are the number of bases of A, G, C and T respectively.

-f, --fasta *string* If selected, will export a fasta file with the selected oligonucleotides to the given output.

-p, --probes *string* If selected, will export a fasta file with the reverse complement of the selected oligonucleotides to the given output.

-S, --stats *string* f selected, will export a tab delimited table to the given output and lengths of how many oligonucleotides would have passed different minimum and specificity criteria, to help tuning parameters in following runs.

Optional arguments related to the output:

-L, --lowMem If selected, will reduce the RAM usage at the expense of speed (depending on the datasets, from 1.5 up to 5 times slower).

--force Only used for debugging unnecessarily large number of mismatches and preventing undesired slow down of the search and large RAM usage.

Examples

```
findOligo -t target.fasta -e excluding.fasta
```

Looks for oligonucleotides of lengths 18 and 20 bases that are present in at least 80% of the sequences from the 'target.fasta' file and at most in 1% of the sequences in the 'excluding.fasta' file and outputs the table to 'target_oligos.tsv'.

```
findOligo -t target.fasta -e excluding.fasta -p probes.fasta
```
Does the same as in the previous example and outputs the reverse complement of the found oligonucleotides to '`probes.fasta`'.

```
findOligo -t target.fasta -e excluding.fasta -l 30 25 22-15 -m 0.95 -s 10 -f
oligos.fasta
```
Looks for oligonucleotides of lengths 30, 25, 22, 21, 20, 19, 18, 17, 16, and 15 (and if an oligonucleotide of a smaller length is found within another oligonucleotide of bigger length it will be ignored) that are present in at least 95% of the sequences from the '`target.fasta`' file and at most in 10 sequences in the '`excluding.fasta`' file. Will output the table to '`target_oligos.tsv`' and a fasta file with the selected oligonucleotides to '`oligos.fasta`'.

`getHomolog`

Description

Given an aligned fasta file and a (unaligned) excluding file, it will create a HMM profile from the aligned file, align the excluding file and export the aligned regions to a fasta file.

This script uses HMMER, please cite accordingly (e.g., HMMER 3.4 (Aug 2023); http://hmmer.org/).

Usage

```
getHomolog -f FILEIN -e EXCLUDING [-l LENGTH] [-o FILE_OUT] [-s SCORE] [-c] [-
k] [-t] [-d] [-v] [-h] [-V]
```

Arguments

Required arguments:

`-f, --file` ***`string`*** An aligned fasta file.

`-e, --excluding` ***`string`*** A reference fasta file to look against.

Optional arguments related to complementarity:

`-l, --length` ***`int|float`*** A minimum coverage of the aligned region. `Default = 0.25`, this means that the aligned sequence from the reference file must be at least 25% long relative to the HMM profile. If the value provided is a '`float`' [0.0-1.0], it will be interpreted as a percentage. If the value provided is a '`int`(eger)' [1-Inf), it will be interpreted as an absolute number of bases.

Optional arguments related to the output:

`-o, --output` ***`string`*** The name of the output fasta file. By default will remove the extension of the input file and add '`_mismatchRegion.fasta`'. If the fasta file exists, it will append the sequences at the end and detect similar sequence headings that will be ignored.

`-s, --score` ***`int|float`*** A minimum pairwise alignment score to be exported (see 'PairwiseAligner' from the Bio.Align module for further details). `Default = 0.90`, this means that the aligned sequence from the excluding file must be at least 90% similar to the HMM profile. If

the value provided is a 'float' [0.0-1.0], it will be interpreted as a percentage. If the value provided is a 'int(eger)' [1-Inf), it will be interpreted as an absolute number of bases.

-c, --complete If selected, will export the complete mismatched sequence and not just the region mismatched.

-k, --keep If selected, will not delete the temporary files ('OUTPUT_hmm_profile.hmm' and 'OUTPUT_hmm_align.fasta') before exiting.

-t, --target If selected, will also print the (unaligned) input file to the head of the output file.

-d, --delete If selected, will delete the output file (if exists) before writing it.

Examples

getHomolog -f oligoN6_region.fasta -r reference.fasta -k
Creates a HMM profile from 'oligoN6_region.fasta', aligns all sequences from 'reference.fasta' to the HMM profile, and exports all aligned regions that aligned at least 25% of their length to the HMM profile to 'oligoN6_region_mismatchRegion.fasta'.

getHomologStats

Description

From one or several fasta files, estimates the similarity of all sequences to the first sequences of every file by pairwise sequence alignment. See 'PairwiseAligner' from the Bio.Align module for further details. Briefly, it will output summary statistics of the alignment score, relative to the first sequence.

This function is especially useful to briefly explore the output from 'getHomolog' (Page 20) and to help the user select the best region to further design specific oligonucleotides.

Usage

```
getHomologStats -f FILEIN [FILEIN ...] [-o OUTPUT] [-a] [-p] [-v] [-h] [-V]
```

Arguments

Required arguments:

-f, --file *string* Fasta file(s) containing the region to be compared against the first sequence.

Optional arguments related to the input:

-t, --target If selected, will assume that the input file contains sequences from the target file, and will ignore all the sequence before that one ending with '_HMM_profile', and compare the HMM profile with all the sequences after that one.

Optional arguments related to the output:

-o, --output *string* The name of the output fasta file. By default will print the information to the console.

-a, --absolute If selected, it will output the statistics as absolute values, and not dividing the scores by the first sequence length.

-p, --plot If selected, will prompt a histogram with the scores for every file.

Examples

`getHomologStats -f oligos_mismatchRegion.fasta -p`
Prints to the console the first sequence (used as reference to compare the rest of the sequences) from '`oligos_mismatchRegion.fasta`', some global statistics of the alignment similarity with the rest of the sequences and a histogram with all individual similarities.

`getHomologStats -f oligos_mismatchRegion.fasta -p -t`
Same as above, but ignores all the sequences before the HMM profile in '`oligos_mismatchRegion.fasta`'.

`getMismatchSeq`

Description

Given a sequence and 'm' mismatches, it will export a fasta file with the mismatched region (default) or the complete sequence.

Usage

```
getMismatchSeq -s SEQUENCE -e EXCLUDING [-m MISMATCHES [MISMATCHES ...]] [-x]
[-o FILE_OUT] [-c] [-n NAME] [-v] [-h] [-V]
```

Arguments

Required arguments:

-s, --sequence *string* A fasta file.

-e, --excluding *string* A excluding fasta file to look against.

Optional arguments related to mismatches search:

-m, --mismatches *int* … The number of mismatches allowed. `Default = ['1', '2']`. A range can be specified with the '-' sign (i.e. '1-3'). Please bear in mind that an excessively high number of mismatches will considerably slow down the search. If '`-m/--mismatches 0`', will not test for mismatches.

-p, --positions *int* … The positions of the mismatch(es). By default will consider all positions.

-x, --indels If selected, will not test for insertions and/or deletions.

Optional arguments related to the putput:

-o, --output *string* The name of the output fasta file. By default will remove the extension of the input file and add '`_mismatches.fasta`'. If a

name is provided with the '`-n/--name`' function, it will write the searched sequence in the first entry.

`-c, --complete` If selected, will export the complete mismatched sequence and not just the region mismatched.

`-n, --name` ***`string`*** If selected, will include the searched sequence in the output fasta file with the given name, and used for the exported fasta file if not given.

Examples

`getMismatchSeq -s ACGTACGT -e excluding.fasta -m 1-3 -n oligoN6`
Exports a fasta file to '`oligoN6_mismatches.fasta`' containing the parsed sequence ('`ACGTACGT`') and all regions that hit the sequence with 1, 2 and 3 mismatches.

`getMismatchSeq -s ACGTACGT -e excluding.fasta -m 1 -p 4`
Exports a fasta file to '`ACGTACGT_mismatches.fasta`' containing all regions that hit the sequence with 1 mismatch in the 4 position.

`getMismatchSeq -s ACGTACGT -e excluding.fasta -m 1 -p 4 -x -c`
Similar to the previous example, but not allowing insertions and deletions and exporting the complete mismatched sequence.

`hairPins`

Description

Given a nucleotide sequence, prints to the console all possible hairpin formations.

Usage

```
hairPins -s SEQUENCE [-b BASES] [-r] [-v] [-h] [-V]
```

Arguments

Required arguments:

`-s, --sequence` ***`string`*** A nucleotide sequence.

Optional arguments related to hairpin formation:

`-b, --bases` Minimum number of base pairs required for hairpin formation. `Default = 3`.

Optional arguments related to the input:

`-r, --revComp` If selected, will reverse complement the sequences before testing.

Examples

`hairPins -s ACGTCGATGACGT`
Prints to the console all potential hairpins with a minimum number of 3 bases.

`hairPins -s ACGTCGATGACGT -b 5 -r`

Prints to the console all potential hairpins with a minimum number of 5 bases after reverse-complementing the input sequence.

`identifyRegions`

Description

From a fasta file containing different potential oligonucleotides, identifies regions of interest and groups together oligonucleotides related to the same region.

Usage

```
identifyRegions -f FILE_IN [-a ALIGN] [-o OUTPUT] [-e FASTA] [-v] [-h] [-V]
```

Arguments

Required arguments:

`-f, --file` ***`string`***	A fasta file containing the oligonucleotides.

Optional arguments related to defining regions:

`-a, --align` ***`float`***	An alignment threshold. `Default = 0.5`, meaning that it is required that at least 50% of the length of the smallest given oligonucleotide is identical to be considered within a region.

Optional arguments related to the output:

`-o, --output` ***`string`***	The output file name. By default will replace the extension of the target file with '`_regions.tsv`'.
`-e, --export`	If selected, will export a fasta file with the selected regions to the given output.

Examples

```
identifyRegions -f oligos.fasta -e oligos_regions.fasta
```

Exports a tab delimited table to '`oligos_regions.tsv`' with the following columns: (i) an arbitrary unique identifier, (ii) the region sequence, (iii) the region length, (iv) how many oligonucleotides were found in the given region, and (v) the identifiers of the oligonucleotides found in the given region. It will also export a fasta file to '`oligos_regions.fasta`' with the identified regions named with the unique identifier and the size of the region (in terms of how many oligos were found within it).

`logStats`

Description

Returns overall statistics and numbers from a log file.

Usage

```
logStats -f FILEIN [-c COLUMNS [COLUMNS ...]] [-p] [-v] [-h] [-V]
```

Arguments

Required arguments:

-f, --file ***string*** A tab delimited table.

Optional arguments related to the input:

-c, --columns ***string ...*** The columns to calculate the stats to. By default will do all columns except the identifier (named: '`identifier`') and the sequences (named: '`sequence`' and/or '`revCom`').

Optional arguments related to the output:

-p, --plot If selected, will prompt a histogram with the scores for each column independently.

Examples

```
logStats -f oligos.tsv
```
Will print summary statistics of the '`oligos.tsv`' file to the console.

```
logStats -f oligos.tsv -c hitsT_prop -p
```
Will print summary statistics of the proportion of hits against the target file from the table '`oligos.tsv`' file to the console and prompt a histogram.

`multi2linefasta`

Description

Converts a fasta file where the sequences are in multiple lines to a fasta file where each sequence is in one line.

This function is mostly used for debugging search functions, especially '`testOligo`'.

Usage

```
multi2linefasta -f FILE_IN [-o FILE_OUT] [-r] [-v] [-h] [-V]
```

Arguments

Required arguments:

-f, --file ***string*** Input fasta file.

Optional arguments related to the output:

-o, --output ***string*** Output fasta file. By default will add '`_1line`' before the extension.

-r, --replace If selected, will replace the input file instead of creating a new file.

Examples

```
multi2linefasta -f target.fasta
```
Transform the input file ('`target.fasta`') to '`target_1line.fasta`' with the sequences in one line, instead of in multiple lines.

`multi2linefasta -f excluding.fasta`
Replaces the sequences from ‘`excluding.fasta`’ stored in multiple lines to a single line.

`oligoNdesign`

Description

This script is a wrapper to run a basic pipeline and select the best oligonucleotides from a target and a excluding file.

This wrapper uses MAFFT (https://mafft.cbrc.jp/alignment/software/) and agrep among others Please cite: Katoh, Misawa, Kuma, Miyata (2002) MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. Nucleic Acids Res. 30:3059-3066. doi: 10.1093/nar/gkf436

The accessibility map of the SSU of the ribosome is based on Behrens S, Rühland C, Inácio J, Huber H, Fonseca A, Spencer-Martins I, Fuchs BM, Amann R (2003) In situ accessibility of small-subunit rRNA of members of the domains Bacteria, Archaea, and Eucarya to Cy3-labeled oligonucleotide probes. Appl Environ Microbiol 69(3):1748-58. doi: 10.1128/AEM.69.3.1748-1758.2003.

The basic pipeline will run the following functions in the given order:
'`findOligo`' (page 17): extract all candidate oligos of lengths 18 and 20 base pairs that are present in at least 80% of the sequences in the target file, and at most 1% in the excluding file.
'`testOligo`' (page 33): (if not disabled) test for 1 and 2 mismatches allowing insertions and deletions of the candidate oligos against the excluding file.
'`alignOligo`' (page 11): align all candidate oligos against a template of the SSU and consensus sequences from the target file.
'`rateAccess`' (page 28): test the accessibility in the tertiary structure of the given oligonucleotide region.
'`bindLogs`' (page 12): bind all log files.
'`selectLog`' (page 29): selects the best 4 oligonucleotides.

For further details of each function call the help command ('`-h`') on the given function.

Its most basic output is composed of three final files:
'`PREFIX_log.tsv`': A tab delimited table containting details for each candidate oligonucleotide.
'`PREFIX_candidates.fasta`': A fasta file containing all candidate oligonucleotides.
'`PREFIX_best.tsv`': A filtered table containing only the details of the 4 best oligonucleotides ranked.

Although the '`-k`' argument will keep other intermediate files:
'`PREFIX_aligned.fasta`': An aligned fasta file containing the SSU template, 2 consensus sequences of the target file, and all candidate oligos.
'`PREFIX_best.fasta`': A fasta file containing the 4 best oligonucleotides ranked.

Usage

```
oligoNdesign -t TARGET -e EXCLUDING -p PREFIX [-l] [-m] [-s] [-c] [-n] [-f] [-g] [-k]
```

Arguments

Required arguments:

-t ***string***	The fasta file containing the target group.
-e ***string***	The fasta file containing the excluding group.
-o ***string***	The prefix for the output files.

Optional arguments:

-l ***int ...***	The lengths of the desired oligonucleotides (in between quotes). `Default = '16-21'`.
-m ***int/float***	The minimum presence in the target file. `Default = 0.8`
-s ***int/float***	The maximum presence in the excluding file. `Default = 0.01`.
-c ***float***	A minimum (and maximum) GC content (in between quotes). `Default = 0`.
-n ***int***	The number of best oligonucleotides to select. `Default = 4`.
-f	It will not test for indels when searching for 1 and 2 mismatches, speeding up completion.
-g ***18S/16S***	The Small SubUnit of the rDNA: Either '`18S`' (default) or '`16S`'.
-k	It will keep intermediate relevant files.

Examples

`oligoNdesign -t target.fasta -e excluding.fasta -o oligos`
Runs the so-called basic pipeline using '`target.fasta`' as target file and '`excluding.fasta`' as excluding file and export three files: '`oligos_log.tsv`', '`oligos_candidates.fasta`', and '`oligos_best.tsv`' (see "Description" for further details).

`oligoNdesign -t target.fasta -e excluding.fasta -o oligos -f -k -n 10`
Does the same as the previous example, but faster (without testing for indels), keeping intermediate files ('`oligos_aligned.fasta`' and '`oligos_best.fasta`'), and selecting the 10 best oligonucleotides (instead of 4 as default).

`oligoNtest`

Description

This script is simply to test all scripts from the oligoN-design pipeline. It will run all sripts, and print their version to the console.

Usage

```
oligoNtest [-p]
```

Arguments

Optional arguments:

-p ***string***	A path to the scripts. By default, it will take the directory name of the command to run this function.

Examples

```
oligoNtest
```
Prompts the version of every function from oligoN-design and exists.

`rateAccess`

Description

Estimate the accessibility of the oligonucleotides in the rDNA by comparing the relative positions to the templates of *Saccharomyces cerevisiae* (18S rDNA, default) or *Escherichia coli* (16S rDNA) as described in Behrens *et al.* (2003; 10.1128/AEM.69.3.1748-1758.2003).

Ideally, '`rateAccess`' uses as input file the output file from '`alignOligo`' (page 11), although the user may provide a custom made fasta file if desired. To do so, the user needs to provide a fasta file with (i) the template sequence of the accessibility map (with '`template`' somewhere in the sequence name ; i.e, '`Saccharomyces_cerevisiae_template`'), (ii) a consensus sequence of the target file (with '`consensus`' somewhere in the sequence name), and (iii) the oligonucleotides to estimate the accessibility score (with '`oligoN`' or '`region`' in their sequence names). Note that only the first sequences with the '`template`' or '`consensus`' identifiers will be taken for the rating, so other sequences may also be present in the fasta file for visual guidance (such as more than one consensus sequence) and they will be ignored. Lastly, if a consensus sequence name also contains '`mostAbundant`', it will be used regardless of the input order.

Usage

```
rateAccess -f FILE_IN [-a ACCESSMAP] [-o FILE_OUT] [-e EXPORT] [-v] [-h] [-V]
```

Arguments

Required arguments:

-f, --file ***string***	A fasta file with the (1) DNA template used to estimate the accessibility map (e.g.; *S. cerevisiae* 18S sequence), (2) the consensus sequence(s) of the target file and (3) the oligonucleotides aligned. Note sequences will be identified by containing the words 'template', 'consensus' and 'oligoN' respectively, and preferably will take the consensus sequence with the most abundant base as consensus to avoid ambiguities.

Optional arguments related to accessibility map:

-a, --accessMap	The accessibility map table of the 18S rDNA ('18S', default), 16S rDNA ('16S') or a custom made table. The default 18S and 16S

tables are taken from the accessibility maps from Behrens *et al.* (2003) and they can be exported with the option 'exportAccessMap' to the desired location. The table must have the following columns in the given order and tab delimited: position, region, base, brightness Class, maximum relative brightness and minimum relative brightness.

Optional arguments related to the output:

-o, --output *string* The name of the output file. Default will remove the extension of the input alignment file and add '_access.tsv'. The file contains the following columns: the name of the oligonucleotide, the sequence, the first position in the target consensus sequence, the approximate region in the 18S (C1-C10, V1-V9), the first position regarding the template, the average maximun relative brightness (0-1), the average minimum relative brightness (0-1), the average relative brightness (0-1) and the given brightness class (VI-I).

-e, --exportAccessMap *string* If selected, will export the default accessibility map table to the specified file and exit.

Examples

```
rateAccess -f oligo_align.fasta -a 16S
```
Exports a tab delimited table to 'oligo_align_access.tsv' with the accessibility scores for every oligonucleotide in 'oligo_align.fasta' after the *E. coli* 16S rDNA template.

```
rateAccess -e access_map.tsv
```
Exports a tab delimited table to 'access_map.tsv' of the *S. cerevisae* 18S rDNA accessibility map.

selectLog

Description

Select the N best oligonucleotides from a tab delimited table by ranking chosen columns and selecting the oligonucleotides with the lowest average rank. This script works best if you want to select by many columns (i.e. >~4). Otherwise a manual selection might work better.

Usage

```
selectLog -t TABLE [-c COLUMNS [COLUMNS ...]] [-r ORDER [ORDER ...]] [-i IDS
[IDS …]] [-w WEIGHTS [WEIGHTS ...]] [-T] [-o OUTFILE] [-n NUMBER] [-v] [-h] [-
V]
```

Arguments

Required arguments:

-f, --file A tab delimited table to be filtered.

Optional arguments related to the selection:

-c, --columns ***string ...*** The columns to be ranked for selecting the best candidate probes. `Default = ['hitsT', 'hitsE', 'mismatch1', 'mismatch2']`.

-r, --order ***descending|d|ascending|a ...*** The order in which the given columns will be ranked (i.e., `descending` or `ascending` numbers). The initials can also be used. `Default= ['defaults']`, which will apply common orders after the column names used in this pipeline (i.e., '`descending, ascending, ascending, ascending`' for the default column names).

-i, --ids ***string|int string|int*** The columns bearing the 'oligonucleotide name' and the 'sequence'. It can be given either the column names or the number of the columns. `Default = 1 2` (first and second columns).

-w, --weights ***int|float …*** A list of weights for the respective columns to calculate the weighted mean. By default will calculate an arithmetic mean.

-T, --thorough A shortcut to apply a thorough selection of the columns to be ranked: `length, hitsT, hitsE, average_brightness, self-dimer_count, hairpin_count, max_consecutive, mismatch1_thorough, mismatch2_thorough`.

Optional arguments related to the output:

-o, --outFile ***string*** The name of the output log file. By default, will print the ranked oligos to the console.

-n, --number ***int*** The number of the best oligonucleotides to be shown. By default will rank all.

Examples

```
selectLog -t oligos.tsv \
      -c hitsT hitsE mismatch1 mismatch2 mismatch1_central0.6 \
      -r d a a a a \
      -w 1 1 1 0.2 2 \
      -n 10 -o oligos_selected.tsv
```

(Just a convoluted example to show options) Exports the 10 best oligonucleotides to '`oligos_selected.tsv`' based on the (i) hits to the target file (in descending order, so the more hits the better), (ii) hits against the excluding file (in ascending order, so the less hits the better), (iii) hits allowing 1 mismatch against the excluding file (ascending), (iv) hits allowing 2 mismatches against the excluding file (ascending), and (v) hits allowing 2 mismatches and positioned in the 60% central bases against the excluding file (ascending). On top of that, the central mismatches are weighted double whereas 2 mismatches are weighted 20%, compared to the hits and the hits allowing 1 mismatch.

```
selectLog -t oligos.tsv -T -n 4
```

Exports the 4 best oligonucleotides after applying a through selection of the columns to be ranked; which are length, hitsT, hitsE, average_brightness, self-dimer_count, hairpin_count, max_consecutive, mismatch1_thorough and mismatch2_thorough.

`selfDimer`

Description

Given a nucleotide sequence, prints to the console all possible self-dimers.

Usage

```
selfDimer -s SEQUENCE [-b BASES] [-r] [-v] [-h] [-V]
```

Arguments

Required arguments:

`-s, --sequence` ***`string`*** A nucleotide sequence.

Optional arguments related to self-dimer formation:

`-b, --bases` ***`int`*** Minimum number of base pairs required for self-dimerization. `Default = 5`.

Optional arguments related to the input:

`-r, --revComp` If selected, will reverse complement the sequences before testing.

Examples

`selfDimer -s ACGTCGATGACGT`
Prints to the console all potential self-dimer with a minimum number of 3 bases.

`selfDimer -s ACGTCGATGACGT -b 5 -r`
Prints to the console all potential self-dimers with a minimum number of 5 bases after reverse-complementing the input sequence.

`sequenceSelect`

Description

Select sequence in a fasta file either from a list or that match a pattern and extract them or remove them.

Usage

```
sequenceSelect -f FILE_IN [-l LISTSEQ] [-p PATTERN [PATTERN ...]] [-k] [-r] [-
o FILE_OUT] [-v] [-h] [-V]
```

Arguments

Required arguments:

`-f, --file` ***`string`*** Input fasta file. Avoid spaces in the sequence names.

Required at least one of:

-l, --list *string* List of sequences to be selected. This must be a different file with each sequence name in a different line.

-p, --pattern *string* … Pattern(s) to be matched for selection of the sequences. When using this option in combination with '`-k/--keep`', it might be faster to use '`grep -A 1 PATTERN FILE_IN > FILE_OUT`' if each sequence from the input file is in a single line.

Optional arguments related to the output:

-o, --output *string* Output file. By default will add '`_selected`' to the input file name. If the file already exists, sequences will be appended at the end of the file.

-k, --keep If selected, will export selected sequences. If neither this argument nor '`-r/--remove`' are selected, this argument will be set by default.

-r, --remove If selected, will delete selected sequences. If both this argument and '`-k/--keep`' are selected, this argument will be ignored.

Examples

`sequenceSelect -f reference.fasta -l names.list`
Selects all sequences from ‘`reference.fasta`’ that are in ‘`names.list`’ and exports them to ‘`reference_selected.fasta`’.

`sequenceSelect -f reference.fasta -o target.fasta -p Radiolaria`
Selects all sequences that contain the string ‘`Radiolaria`’ from ‘`reference.fasta`’ and exports them to ‘`target.fasta`’.

`sequenceSelect -f reference.fasta -o excluding.fasta -p Radiolaria -r`
Selects all sequences that NOT contain the string ‘`Radiolaria`’ from ‘`reference.fasta`’ and exports them to ‘`excluding.fasta`’.

table2fasta

Description

Exports a fasta file from a tab delimited table.

Usage

```
table2fasta -f FILE_IN [-n] [-c COLUMNS COLUMNS] [-o FILE_OUT] [-v] [-h] [-V]
```

Arguments

Required arguments:

-f, --file *string* A tab delimited file.

Optional arguments related to the input:

-c, --columns *string|int string|int* The columns to be exported in the order 'oligonucleotide name' and 'sequence'. It can be given either the

column names or the number of the columns. `Default = [1, 2]` (first and second columns).

`-n, --noHeaders` If selected, will interpret the file has no headers.

Optional arguments related to the output:

`-o, --output` ***`string`*** The name of the output file. By default will replace the extension to '`.fasta`'.

Examples

`table2fasta -f oligos.tsv`
Exports to '`oligos.fasta`' a fasta file with the sequence names extracted from the first column and the sequences from the second column, understanding there are headers in the input table ('`oligos.tsv`').

`table2fasta -f oligos.tsv -c 1 3 -o oligos_revCom.fasta`
Exports to '`oligos_revCom.fasta`' a fasta file with the sequence names extracted from the first column and the sequences from the third column, understanding there are headers in the input table ('`oligos.tsv`').

`testOligo`

Description

Test oligonucleotides against a excluding database allowing mismatches. Note this script will also count the exact hit.

This script is a wrapper using '`agrep`'

Usage

```
testOligo -p OLIGOS -e EXCLUDING [-o OUTPUT] [-m 'MISMATCH']
```

Arguments

Required arguments:

`-p` ***`string`*** Oligonucleotides fasta file.

`-e` ***`string`*** Excluding fasta file.

Optional arguments:

`-o` ***`string`*** Output log file. By default, will replace the extension of the oligonucleotides file by '`_tested.tsv`'.

`-m` ***`int ...`*** The number of mismatches (in between quotes). `Default='1 2'`.

Examples

```
testOligo -p oligos.fasta -e excluding.fasta -m '3 4'
```

For every oligonucleotide in 'oligos.fasta', exports to 'oligos_tested.tsv' a tab delimited table with the proportion and absolute number of hits allowing 3 and 4 mismatches against 'excluding.fasta'.

testThorough

Description

Test thoroughly a fasta file containing oligonucleotides against a excluding fasta file. It will test for self-dimers, hairpins, consecutive bases and/or mismatches.

Usage

```
testThorough -f FILE_IN -e EXCLUDING [-p] [-s] [-S SELFDIMERBASES] [-a] [-A
HAIRPINPOSITIONS] [-k] [-m MISMATCHES [MISMATCHES ...]] [-b FLANK] [-c CENTER]
[-x] [-o FILE_OUT] [-t TABLE] [-n] [-i] [-u UNIQUE [UNIQUE ...]] [-d
DELIMITER] [-v] [-h] [-V]
```

Arguments

Required arguments:

-f, --file ***string*** A fasta file.

-e, --excluding ***string*** A excluding fasta file to look against.

Optional arguments related to the input:

-p, --revComp If selected, will reverse complement the sequences in the input file before testing.

Optional arguments related to the testing:

-s, --selfDimer ***int*** Minimum number of base pairs required for self-dimerization. `Default = 5`. If '0', will not test for self dimerization.

-a, --hairpin ***int*** Minimum number of base pairs required for hairpin formation. `Default = 3`. If '0', will not test for hairpin formation.

-k, --consecutives If selected, will not search the maximum string of repeated consecutive bases.

-m, --mismatches ***int ...*** The number of mismatches allowed. `Default = ['1', '2']`. A range can be specified with the '-' sign (i.e. '0-3'). Please bear in mind that an excessively high number of mismatches will considerably slow down the search. If '0', will not test for mismatches.

-b, --flank ***int/float*** The number of leading and trailing (flanks) bases to flag when the given number of mismatches are found within. `Default = 0`, it will not look for leading nor trailing mismatches. If (e.g.,) 3 is selected along with 2 mismatches, it will count how many hits are occurring when 2 mismatches are present within 3 flanking bases. If the value provided is a 'float' [0.0-1.0], it will be interpreted as a percentage of the given oligo length.

-c, --center ***int/float*** Similar to '-b/--flank' but centered. Default = 0, it will not look for centered mismatches. If (e.g.,) 10 is selected along with 2 mismatches, it will count how many hits are occurring when 2 mismatches are present in the 10 centered bases. If the value provided is a 'float' [0.0-1.0], it will be interpreted as a percentage of the given oligo length.

-x, --indels If selected, will not test for insertions and/or deletions.

Optional arguments related to the output:

-o, --outputm ***string*** The name of the output file. By default will remove the extension of the input file and add '_testThorough.tsv'. The file will contain and entry for every oligonucleotide with the name of the oligonucleotide, the sequence, the observed number of dimers, the observed number of hairpins, the maximum string of consecutive bases, and the absolute number of hits, the proportion of hits, and the mismatches per position for every number of mismatches selected.

-t, --table ***string*** If selected, will export a summarized output of global analyses in table format to the given output (i.e., total self-dimer counts, but not the examples).

-n, --extended If selected, will add a detailed report of the mismatches at every position. It will also add '_testThoroughExt.tsv' to the output name if not provided.

-i, --identity If selected, will not append the identity of the mismatched hits. This output can be summarised with the arguments '-u/--unique' and '-d/--delimiter' and target specific elements of the sequence identifiers if properly formatted. This output will not be included in the table (if selected).

-u, --unique ***int ...*** The unique field(s) of the sequence identifiers in the excluding file. Works similar to '-f' from the bash command 'cut'.

-d, --delimiter ***string*** The delimiter separating fields from the sequence identifiers in the excluding file. Works similar to '-d' from the bash command 'cut'.

Examples

```
testThorough -f oligos.fasta -e excluding.fasta -b 3 -c 0.6 -i -t
oligos_testThoroughTable.tsv
```

Exports a log file to 'oligos_testThorough.tsv' with all the default tests (selfdimers, hairpins, consecutive bases, 1 and 2 mismatches allowing indels), tests for mismatches happening in the 3 first and last bases and in the 60% central bases, do NOT include the identity of the hits and exports a summary table to 'oligos_testThoroughTable.tsv'.

```
testThorough -f oligos.fasta -e excluding.fasta -n -u 4-6 -d '|'
```

Exports an extended log file to 'oligos_testThoroughExt.tsv' with all the default tests including the identity of the hits but summarised to the 4th, 5th and 6th field separated by a

pipe ('|'). For example, from 'Eukaryota|TSAR|Rhizaria|Radiolaria|Polycystinea|Nassellaria|Pterocorythidae|Pterocorys|Pterocorys_cf_zanclea', will extract 'Radiolaria|Polycystinea|Nassellaria'.

trimRegion

Description

Trim an alignment on a given region.

Usage

```
trimRegion -f FILEIN [-r REGION [REGION ...]] [-p POSITIONS [POSITIONS …]] [-s SEQUENCES [SEQUENCES ...]] [-o FILE_OUT [FILE_OUT ...]] [-d PATH] [-n] [-k] [-v] [-h] [-V]
```

Arguments

Required arguments:

-f, --file ***string*** An aligned fasta file.

Required at least one of:

-r, --region ***string ...*** The oligonucleotide sequence to be searched (i.e., 'CAGTATTCTGAAGCGAGA') or the name of the sequence containing the region to be trimmed (i.e., 'oligoN4').

-p, --positions ***int ...*** The positions in the alignment to be kept separated by an hyphen (i.e., '316-334 914-952').

-s, --sequences ***string ...*** A fasta file(s) contaning the oligonucleotides sequences to be searched.

Optional arguments related to the output:

-o, --output ***string ...*** The name(s) of the output fasta file(s). By default will remove the extension of the input file and add '_trimmed.fasta'. If more than one trimmed regions are given, by default will add consecutive integers after '_trimmed' (i.e., '_trimmed1.fasta', '_trimmed2.fasta').

-d, --path ***string*** The path of the output files. Useful when default output names are given, but in a different directory. By default it will use the current directory. If names are provided through the '-o/--output' argument, this argument will be ignored.

-n, --name If selected, will use the name of the sequences from the fasta file parsed by '-s/--sequences' for the output name.

-k, --keep If selected, will not delete empty sequences or sequences composed only of the trimmed region.

Examples

```
trimRegion -f target_align.fasta -r GTAGTTATCGTA AAACTGTGTCA -d regions
```

Exports to the directory ‘regions’ two fasta files (‘regions/target_align_trimmed1.fasta’ and ‘regions/target_align_trimmed2.fasta’) corresponding to the two given regions.

```
trimRegion -f target_align.fasta -p 450-900 -k -o target_align_V4.fasta
```
Exports a fasta file to ‘target_align_V4.fasta’ containing all sequences (empty or incomplete) resulting from removing all position before the position 450 and after 900 from the ‘target_align.fasta’ file.

```
trimRegion -f target_align.fasta -s regions.fasta -d regions -n
```
Exports to the directory ‘regions’ as many fasta files as there are sequences in the ‘regions.fasta’ file and name those exported fasta files with the name of the sequences.

4. Common problems and misconceptions

4.1. Poorly curated target file

It is important to understand some search parameters of this tool. For example, by default `findOligo` will identify oligonucleotides that are present in 80% of the sequences of your target file. If the target and excluding files contain clear subsets of sequences (i.e., full length rDNA along with short metabarcoding reads of specific hypervariable regions), it might be worth splitting the files and analysis so that each set of files correspond to the nearly same region to optimize different specificity criteria (i.e., ' `-m` ' and '`-s`' in '`findOligo`'). For example, if less than 80% of the sequences in your target file are missing the V4 region of the SSU, there will be no identified oligonucleotides in the V4 region. To solve that issue, you could either lower the '`-m`/`--minimum`' criteria, or (recommended) repeat the search with a different target file containing sequences of comparable length/regions.

Additionally, despite enormous and invaluable efforts to manually curating reference databases, taxonomic annotation is not perfect. Therefore, it is possible that the region of interest might also be present in the excluding file due to chimeric sequences, badly annotated sequences or simply high similarity to other groups. It is therefore recommended to be aware of such limitations, explore the hits against the excluding file, and repeat the curation of the files if needed.

4.2. Too divergent sequences in the target file

It may happen that the selected target group contains sequences too divergent with clear subgroups, and it is not possible to find specific oligonucleotides. In this case it might be worth dividing the target group into the different groups and run independent searches.

4.3. Different number of hits against the excluding file

If a fasta file is storing the sequences in different lines, it may happen that a '`grep -c`' yields different number than that provided by '`findOligo`' or '`testOligo`'. That might be because the given pattern is broken into different lines (e.g.: '`ACGT\nGCTA`'). As already mention, when searching for patterns in fasta files we recommend to store the sequences in a single line (see '`multi2linefasta`', page 25).

In a further level of complexity, mismatches are interpreted differently among the different search functions:
-`findOligo`: measures only exact mismatches of the four DNA bases (A, C, G and/or T).
-`testOligo`: measures mismatches by distance.
-`testThorough`: measures exact mismatches allowing insertions of one base and deletions.
-`getHomolog`: aligns sequences based on a HMM profile.
Therefore it is normal that the number of hits provided by the different functions may differ for very specific details.

5. Good practices

Although the following lines are not required to run the oligoN-design tool, they will certainly contribute to both help you design good specific oligonucleotides, and take the most of your input files, specially your target file. These recommendations will also avoid falling in the previously mentioned problems and misconceptions.

5.1. Check the literature

Before starting to design specific oligonucleotides to your target group, it is important to do some literature research about your group. Have other studies previously designed specific oligonucleotides to your group of interest? Then you might be able to avoid the lovely (but painful) process of designing specific oligonucleotides. Is your group of interest monophyletic? If it is not, you might fall in the problem 4.2 (too divergent sequences). Is there very little information about your group of interest in the literature? Then it's your lucky day, because this tool was design for those specific groups we know very little about.

5.2. Understand your target file and target group

It is of paramount importance to understand both your target file AND your target group. Briefly, in the first case, you basically want to avoid the misconception 4.1 and in the second case the misconception 4.2. Therefore it is recommended to curate your target file according to the purpose of the oligonucleotide. This will also help you select optimum thresholds and optimize the chances of finding good candidate oligonucleotides. Check all sequences are of similar(ish) length and that all sequences are relatively similar (or that at least they share common regions). To do so, you can align and build a quick exploratory tree to curate your target group.

5.3. Align your target file

Aligning the target file and visualizing the global multiple sequence alignment will help you confirm that nearly the entirety of the analyzed gene is covered. In addition, if all sequences span from nearly the beginning to nearly the end of the gene, you might want to consider increase the minimum criteria (set as default at 80%). An alignment can be done for example with MAFFT (Katoh and Standley, 2013):

```
mafft target.fasta > target_align.fasta
```

Then the aligned target file can be opened with (e.g.,) aliview (Larsson, 2014). However, sometimes it is difficult to tell the diversity of the sequences, and you might need to carry out a slightly more complex check, such building a quick exploratory phylogenetic tree.

5.4. Build a exploratory phylogenetic tree

Building a quick exploratory tree will help you validate that the diversity within your target group is relatively low and that it is indeed potentially possible to design a specific oligonucleotide. Or in other words, that your target file contain sequences relatively similar. A quick phylogenetic tree can be done for example with `RaxML-ng` (Kozlov *et al.*, 2019):

```
raxml-ng --msa target_align.fasta --prefix target_raxml-ng --model GTR+G
```

Or with `IQ-Tree` (Minh *et al.*, 2020):

```
iqtree2 -s target_align.fasta -pre target_IQtree
```

Although if your target group contains way too many sequences or you more or less known the diversity of your group, you can also infer a less computationally demanding tree based on neighbour-joining with `rapidnj` (Simonsen, Mailund and Pedersen, 2008)t is important however, that this tree do not represent phylogenetic relationships, and “simply” creates a similarity-based hierarchical clustering:

```
rapidnj target_align.fasta > target_nj.tre
```

If you are deeply interested in exploring phylogenetic patterns of your target group, have a look at this repository to get started: github.com/MiguelMSandin/phylogeniesQuickStart.

6. Example workflows

With the following lines we have written some examples in detailed on how to design specific oligonucleotides, from a beginner perspective to an expert use of this tool. Although the possibilities extend far beyond.

6.1. Unsupervised run

The simplest option if you are either new to designing oligonucleotides or to the command line is to run the wrapper script '`oligoNdesign`' (specific function information in page 26). By default, this script will select the 4 best oligonucleotides of 18 and/or 20 base pairs long, from a target fasta file containing your sequences of interest (parsed with the argument '`-t target.fasta`') and a excluding fasta file containing the sequences to be excluded (with the argument '`-e excluding.fasta`') (**Fig. 1**), as follows:

```
oligoNdesign -t target.fasta -e excluding.fasta -o oligos
```

This command will output three files with the '`oligos`' prefix (which can be modified with the argument '`-o oligos`'):

- '`oligos_log.tsv`': A tab delimited table containing details for each candidate oligonucleotide."
- '`oligos_candidates.fasta`': A fasta file containing all candidate oligonucleotides."
- '`oligos_best.tsv`': A filtered table containing only the details of the 4 best oligonucleotides ranked."

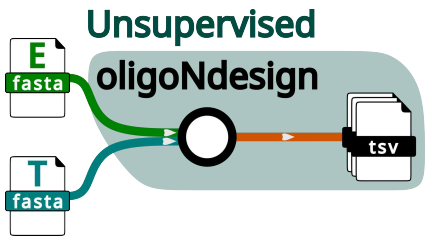


Figure 1. Schematic representation of the unsupervised workflow suggested herein.

If you want to select different lengths you can do so with the argument '`-l`'. For example, let's supposed we want 15, 16, 17, 18, 19, 20, 21, 22 and 25. The we can run:

```
oligoNdesign -t target.fasta -e excluding.fasta -o oligos -l 15-22 25
```

It may happen that the default specificity criteria may be too stringent (or too relaxed). By default this pipeline searches for oligonucleotides that are present in at least 80% of the sequences in the target file, and at most in 1% of the sequences of the excluding file. These criteria are called `minimum` and `specificity`, respectively, and we can also modify them. For example imagine you are happy with oligonucleotides of length 18 and/or 20, but you want a much stringent `minimum` (e.g., 95%) and `specificity` (e.g., 0.1%) criteria:

```
oligoNdesign -t target.fasta -e excluding.fasta -o oligos -m 0.95 -s 0.001
```

Additionally you can also set a minimum (and maximum) GC content (argument '`-c`'), select more than the best 4 oligonucleotides ('`-n`'), test the oligonucleotides for their accessibility against the 16S rDNA template from *Escherichia coli* ('`-g`'; and not the 18S rDNA template from *Saccharomyces cerevisae*, as default), keep intermediate files ('`-k`'), and tune other parameters further. However, you might then be ready to jump to the "Basic pipeline" and have more control over intermediate files and parameters.

6.2. Basic pipeline

The basic pipeline is the supervised version of the '`oligoNdesign`' function. Briefly, you deal with all intermediate files, arguments and options, which allows you to tune and optimize parameters and user-defined thresholds according to your data and needs (**Fig. 2**).

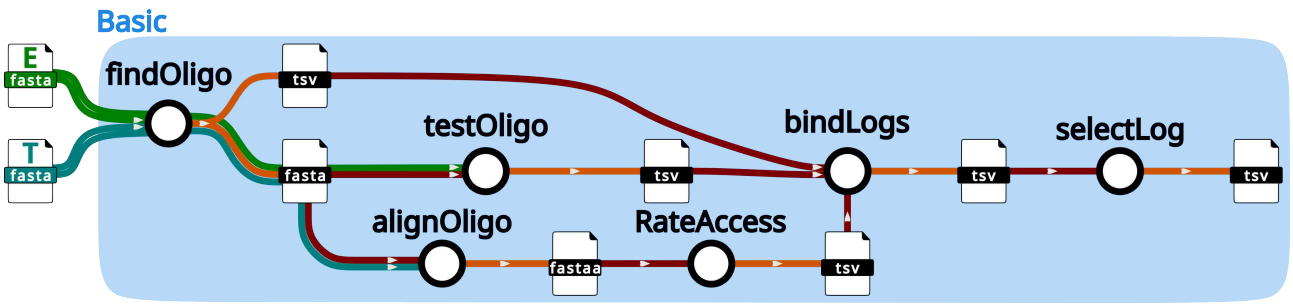


Figure 2. Schematic representation of the so-called basic workflow suggested herein.

Assuming you are already comfortable with the command line. Let's begin by defining our variables, so you don't have to call each file every time we run a function:

```
TARGET='target.fasta'
EXCLUDING='excluding.fasta'
PREFIX='oligos'
```

Ideally, you already have your own `target` and `excluding` fasta files, which you have explored and understand their weaknesses, strengths and diversity (see '4. Common problems or misinterpretations', page 38). Although, if you have no knowledge other than the group you are interested in, you can create those preliminary files from reference databases. For example, reference databases such as PR2 (Guillou *et al.*, 2013) or SILVA (Quast *et al.*, 2013) are curated (phylogenetically or based on quantitative quality, respectively) and could be a good starting point. Other options could be your own metabarcoding dataset, or even a custom local dataset of your study system. Anyways, you create your target and excluding files, as follows (where you'll replace `PATTERN` for your favorite group):

```
sequenceSelect -f reference.fasta -o ${TARGET} -p PATTERN
sequenceSelect -f reference.fasta -o ${EXCLUDING} -p PATTERN -r
```

Once you have your target and excluding files, you can begin with the pipeline. First you will look for all potential oligonucleotides that meet your search criteria with the `findOligo` function (page 17). As a reminder, the search criteria are oligonucleotides of 18 and 20 bases (‘`-l 18 20`’), that are present in at least 80% of your target file (‘`-m 0.80`’) and at most in 1% of the excluding file (‘`-s 0.01`’).

```
findOligo -t ${TARGET} -e ${EXCLUDING} -l 18 20 -m 0.80 -s 0.01 -n 0 -o $
{PREFIX}_find.tsv -f ${PREFIX}_find.fasta
```

Note that default parameters are explicitly specified to ease parameters optimization. In addition, we have set mismatches search to 0 (‘`-n 0`’), because later on we will test for mismatches including insertions and deletions.

The `findOligo` function will output a tab delimited table with the oligonucleotides and all the search values to ‘`oligos_find.tsv`’ and a fasta file containing the oligonucleotides to ‘`oligos_find.fasta`’.

The next step would be to check for (1 and 2) mismatches allowing insertions and deletions with the function `testOligo` (page 33), as follows:

```
testOligo -p ${PREFIX}_find.fasta -e ${EXCLUDING} -o ${PREFIX}_test.tsv -m “1
2”
```

Then, rate the accessibility of the selected oligonucleotides in the SSU with the ‘`rateAccess`’ function (page 28). To do so, we have first to prepare the input file with the function ‘`alignOligo`’ (page 11). Briefly, the input file consist of the template SSU gene which accessibility is known (i.e., *E. coli* 18S rDNA as default), a consensus sequence of the target file, and the oligonucleotides to be rated. Note that the function ‘`rateAccess`’ needs the identifiers ‘`template`’, ‘`consensus`’ and ‘`oligoN`’ (or ‘`region`’) in the sequence names to find the template the consensus and oligonucleotides respectively.

```
alignOligo -t ${TARGET} -p ${PREFIX}_find.fasta -o ${PREFIX}_align.fasta
rateAccess -f ${PREFIX}_align.fasta
```

The `rateAccess` function will output a tab delimited table with the oligonucleotide identifier, the sequence, the starting position in the consensus sequence, the approximate region, the starting position in the template sequence, the average maximum brightness, the average minimum brightness, the average brightness and the accessibility class.

Note that these steps are not required to be run in the order of this example. For example if you are working with a different gene, you might not know the accessibility of every region, so you might want to skip this step. Although, if you have the accessibility of your preferred gene, or you want to update the provided template, you can do so with the ‘`-a/--acessMap`’ argument.

Now it's time to merge all different tables with all the different search results into a single table. This can be done with the function '`bindLogs`' (page 12), which will merge all tables into '`oligos_log.tsv`'. To keep the directory clean, it is possible to remove the input tables with the '`-r/--remove`' option.

```
bindLogs -f ${PREFIX}_find.tsv ${PREFIX}_test.tsv ${PREFIX}_align_access.tsv -
o ${PREFIX}_log.tsv -r
```

Lastly, it is only needed to select the best overall oligonucleotides based on the search just performed with the function '`selectLog`' (page 29). If this example was followed, it is possible to apply the following example selection:

```
selectLog -t ${PREFIX}_log.tsv -c hitsT hitsE mismatch1_indel mismatch2_indel
average_brightness -n 4
```

This will print to the console the 4 best oligonucleotides based on the maximum number of hits to the target file, on the minimum number of hits against the excluding file (allowing 1 and 2 mismatch with and without indels) and with the best accessibility score.

6.3. Thorough design

In a further level of complexity, here the position of the mismatches and other three-dimensional structures (such as hairpins of self-dimerizations) are considered. However, this option is computationally expensive, so we will begin by selecting a reasonable set of all potential best oligonucleotides and then apply a thorough test on them (**Fig. 3**).

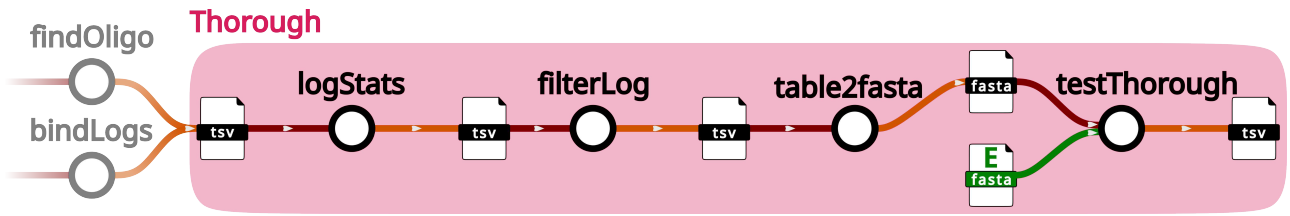


Figure 3. Schematic representation of the so-called thorough workflow suggested herein.

We can profit from the previously created log file (resulting from the function '`bindLogs`' from the '**basic pipeline**'). So let's explore it with the function '`logStats`' (page 24):

```
logStats -f ${PREFIX}_log.tsv
```

This will print to the console some summary statistics of all columns in the table. Based on these statistics we can already apply a more stringent selection of the oligonucleotides and exclude those that resulted in a poor search. For example, we will select oligonucleotides with a GC content in between 40 and 60%, that hits at least 95% of the sequences in the target file, at most 0.1% in the excluding file, at most 0.1% of hits allowing 1 mismatch to the

excluding file, at most 0.01% of hits allowing 2 mismatches, and with maximum accessibility class of 'IV'. We can do that with the function '`filterLog`' (page 16) as follows:

```
filterLog -l ${PREFIX}_log.tsv -s 0.4 0.6 -t 0.95 -R 0.001 -m 0.001 -M 0.0001
-k IV
```

Note that such selection can also be done with the function '`selectLog`', and restrict the thorough search considerably. However, the function '`selectLog`' most of the times selects different oligonucleotides within the same region, and now we are interested in exploring further tests (i.e., hairpins or position of the mismatches). This is why a slightly broader selection will allow selecting different regions of the gene, apply additional tests, and probably resulting in a different final selection of the best oligonucleotides.

Let's export a fasta file from the selected table with the selected oligonucleotides with the function '' (page 32):

```
table2fasta -l ${PREFIX}_log_filtered.tsv
```

Once we have a reasonably small selection of oligonucleotides in fasta format, we can start the through search with the function '`testThorough`' (page 34). This is probably the slowest function of the whole tool set, so unless you are running the pipeline in a cluster and you don't want to spend several hours running it, it is better to select a reasonably small number of oligonucleotides. For example, testing around 20 oligonucleotides (of lengths ~15 to ~25 bases) against the PR2 reference database may take from ~1 to 7 hours. In addition, this function will export a detailed log file, with all possible hairpins, self-dimers, and the identity of the hits allowing mismatches, among others. Continuing with the example of PR2, most of the reference databases contain a given structure to simplify extracting information. In this example, PR2 uses the pipe symbol ('`|`') to delineate different hierarchical taxonomic categories (i.e., class, order, family). Let's suppose we are interested only on the broad identity of the hits and we don't really care about the specific sequences at this point. Then we can summarize the identity of the hits by extracting specific taxonomic levels (e.g., 7, 8 and 9). Lastly, let's also suppose we are designing probes for *in situ* hybridization, so we are interested that the given oligonucleotide contains mismatches in the center. Then such a test would look like:

```
testThorough -f ${PREFIX}_log_filtered.fasta -e ${EXCLUDING} -c 0.6 -u 7-9 -d
'|'
```

With the '`${PREFIX}_log_filtered_testThorough.tsv`' output you can now visually inspect your selection of oligonucleotides and select that one that best fit your requirements.

Although, despite the output of '`testThorough`' is meant for a detailed check and quantitative assessment, it is possible to also export a summary table with the argument '`-t`', or even to transform the long detailed output into the same table (so there is no need to run the long

test again) with the function 'detailed2table' (page 14). This will allow an automatic selection of the oligonucleotides and include the 'testThorough' function in your custom high-through put pipeline. So just to complete the example, we can do this as we have already seen:

```
detailed2table -f ${PREFIX}_log_filtered_testThorough.tsv

bindLogs -f ${PREFIX}_log_filtered.tsv $
{PREFIX}_log_filtered_testThorough_table.tsv -o $
{PREFIX}_log_filtered_tested.tsv -r

selectLog -t ${PREFIX}_log_filtered_tested.tsv -c hitsT hitsE
average_brightness self-dimer_count hairpin_count mismatch1_thorough
mismatch1_central0.6 mismatch2_thorough mismatch2_central0.6
```

This will result in the ranking of all filtered oligonucleotides from the file 'oligos_log_filtered_tested.tsv'.

6.4. Expert design

For the last example it is required some theoretical and empirical experience designing specific oligonucleotides, and such experience should be preferably manual. Rather than designing final specific oligonucleotides, here we will identify specific regions of interest and compare them with the excluding file, for a visual (or manual) identification of oligonucleotides (**Fig. 4**). Basically, in this example we will stream-line the "looking at the alignment" approach by taking profit of HMM profiles.

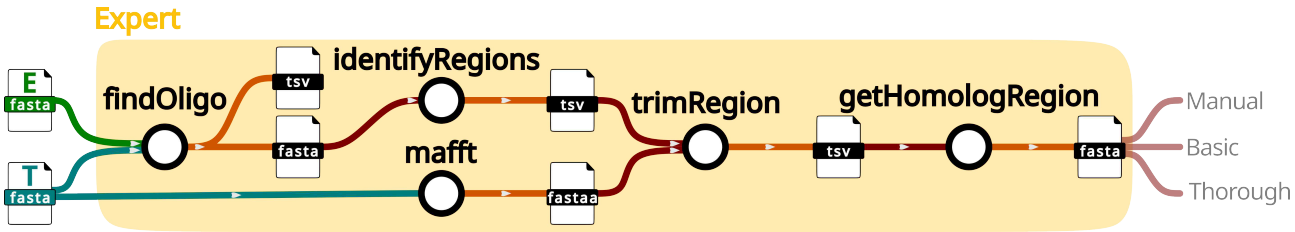


Figure 4. Schematic representation of the so-called expert workflow suggested herein.

To do so, we will begin by identifying regions of interest with the already known 'findOligo' function, but instead we will select length of 30, 25, 20 and 15. Such an inverse order will avoid redundancy of smaller oligonucleotides nested within larger oligonucleotides:

```
TARGET='target.fasta'
EXCLUDING='excluding.fasta'
PREFIX='expert'
findOligo -t ${TARGET} -e ${EXCLUDING} -l 30 25 20 15 -o ${PREFIX}_find.tsv -f
${PREFIX}_find.fasta
```

Now we will merge the found oligonucleotides into regions with the function ‘identifyRegions’ (page 24), since most of the found oligonucleotides are sliding windows of length *k* within a given region of length >*k*.

```
identifyRegions -f ${PREFIX}_find.fasta -o ${PREFIX}_regions.tsv -e $
{PREFIX}_regions.fasta
```

This will export a tab delimited table with the regions, the region length, the number of oligonucleotides within each region (or *size*) and the identifiers of the oligonucleotides. It will also export a fasta file with the regions and their size. It is already possible to explore the regions further by aligning the target file, and then the so-called regions to it with mafft as follow:

```
mafft ${TARGET} > ${TARGET/.fasta/_align.fasta}
mafft --addfragment ${PREFIX}_regions.fasta ${TARGET/.fasta/_align.fasta} > $
{TARGET/.fasta/_align_regions.fasta}
```

These regions will now be extract from the aligned target file (‘target_align.fasta’) with the function ‘trimRegion’ (page 36). Although to keep the directory somehow organized, we will export all extracted regions to a directory called ‘regions’ (with the argument ‘-d’), and named these files after the region name (with the argument ‘-n’), as in ‘oligos_regions.fasta’. We can do all of this as follows:

```
trimRegion -f ${TARGET/.fasta/_align.fasta} -s ${PREFIX}_regions.fasta -d
regions -n
```

We can now compare these regions to the excluding file with HMM profiles using the function ‘getHomolog’ (page 20). And to run efficiently this function in all different extracted regions, we can do so in a for loop as follows:

```
for FILE in $(ls regions/*); do
      getHomolog -f ${FILE} -e ${EXCLUDING}
done
```

This will export different files containing the region from the target file, the HMM profile from the target file, and all aligned regions from the excluding file that shared at least 90% of the aligned positions. While for some experts these files might already be enough to start designing oligonucleotides, it is possible to still go further and explore the outcome with the function ‘getHomologStats’ (page 21).

```
getHomologStats -f regions/*complementRegion* -t -p
```

And now it’s really up to your imagination. For example, you could chose the most appropriate region and start a thorough design on it (as seen in page 44), or even create a new target and excluding file from one of the files generated by the function ‘getHomolog’.

Just to finish setting the basics of the oligoN-design tool, let's imagine we are interested in the second region with 20 oligonucleotides and we want to create a new target and excluding file from it. If you followed this example, it should be called '`region2_size20_complementRegion.fasta`'. Then we could do the following (replacing '`PATTERN`' by the pattern you used at the beginning in the Basic pipeline, page 42):

```
FILE='regions/region2_size20_complementRegion.fasta'
TARGET='region2_target.fasta'
EXCLUDING='region2_excluding.fasta'
sequenceSelect -f ${FILE} -o ${TARGET2} -p PATTERN
sequenceSelect -f ${FILE} -o ${EXCLUDING2} -p PATTERN HMM_profile -r
```

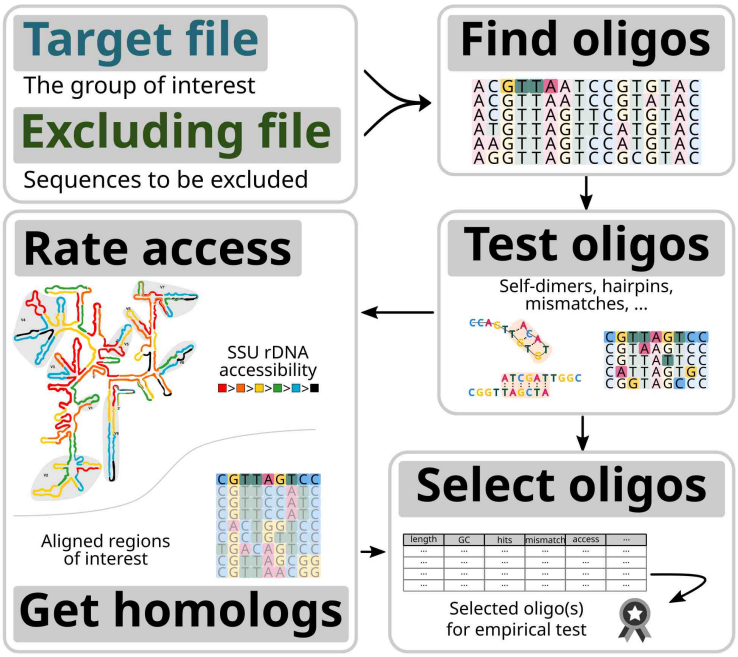
We can explore the new target file with the function '`breakFasta`' (page 13), as follows:

```
breakFasta -f ${TARGET} -p
```

And finally proceed with your favorite pipeline.

7. Concluding remarks

The oligoN-design tool was designed to help the design of specific oligonucleotides accommodating the large environmental datasets and with a simple and versatile approach. Here we have suggested several workflows to integrate the OligoN-design tools and ease the design of specific oligonucleotides.



However, oligonucleotide design is a tedious work and any *in silico* result should be interpreted as a starting point that needs to be empirically optimized and tested. Therefore, bioinformatic workflows will only provide theoretical candidate regions.

8. References

Beliveau, B.J. *et al.* (2018) 'OligoMiner provides a rapid, flexible environment for the design of genome-scale oligonucleotide in situ hybridization probes', *Proceedings of the National Academy of Sciences*, 115(10). Available at: https://doi.org/10.1073/pnas.1714530115.

Edgar, R.C. (2022) 'Muscle5: High-accuracy alignment ensembles enable unbiased assessments of sequence homology and phylogeny', *Nature Communications*, 13(1), p. 6968. Available at: https://doi.org/10.1038/s41467-022-34630-w.

Gouy, M., Guindon, S. and Gascuel, O. (2010) 'SeaView Version 4: A Multiplatform Graphical User Interface for Sequence Alignment and Phylogenetic Tree Building', *Molecular Biology and Evolution*, 27(2), pp. 221–224. Available at: https://doi.org/10.1093/molbev/msp259.

Guillou, L. *et al.* (2013) 'The Protist Ribosomal Reference database (PR2): A catalog of unicellular eukaryote Small Sub-Unit rRNA sequences with curated taxonomy', *Nucleic Acids Research*, 41(D1), pp. 597–604. Available at: https://doi.org/10.1093/nar/gks1160.

Hall, T.A. (1999) 'BioEdit: A User-Friendly Biological Sequence Alignment Editor and Analysis Program for Windows', 41(41), pp. 95–98.

Hendling, M. *et al.* (2018) 'Oli2go: an automated multiplex oligonucleotide design tool', *Nucleic Acids Research*, 46(W1), pp. W252–W256. Available at: https://doi.org/10.1093/nar/gky319.

Hendling, M. and Barišić, I. (2019) 'In-silico Design of DNA Oligonucleotides: Challenges and Approaches', *Computational and Structural Biotechnology Journal*, 17, pp. 1056–1065. Available at: https://doi.org/10.1016/j.csbj.2019.07.008.

Katoh, K. and Standley, D.M. (2013) 'MAFFT multiple sequence alignment software version 7: Improvements in performance and usability', *Molecular Biology and Evolution*, 30(4), pp. 772–780. Available at: https://doi.org/10.1093/molbev/mst010.

Kozlov, A.M. *et al.* (2019) 'RAxML-NG: a fast, scalable and user-friendly tool for maximum likelihood phylogenetic inference', *Bioinformatics*. Edited by J. Wren, 35(21), pp. 4453–4455. Available at: https://doi.org/10.1093/bioinformatics/btz305.

Larsson, A. (2014) 'AliView: a fast and lightweight alignment viewer and editor for large datasets', *Bioinformatics*, 30(22), pp. 3276–3278. Available at: https://doi.org/10.1093/bioinformatics/btu531.

Ludwig, W. *et al.* (2004) 'ARB: a software environment for sequence data', *Nucleic Acids Research*, 32(4), pp. 1363–1371. Available at: https://doi.org/10.1093/nar/gkh293.

Minh, B.Q. *et al.* (2020) 'IQ-TREE 2: New Models and Efficient Methods for Phylogenetic Inference in the Genomic Era', *Molecular Biology and Evolution*. Edited by E. Teeling, 37(5), pp. 1530–1534. Available at: https://doi.org/10.1093/molbev/msaa015.

Quast, C. *et al.* (2013) 'The SILVA ribosomal RNA gene database project: improved data processing and web-based tools', *Nucleic Acids Research*, 41(D1), pp. D590–D596. Available at: https://doi.org/10.1093/nar/gks1219.

Sievers, F. *et al.* (2011) 'Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega', *Molecular Systems Biology*, 7(1), p. 539. Available at: https://doi.org/10.1038/msb.2011.75.

Simonsen, M., Mailund, T. and Pedersen, C.N.S. (2008) 'Rapid Neighbour-Joining', in K.A. Crandall and J. Lagergren (eds) *Algorithms in Bioinformatics*. Berlin, Heidelberg: Springer Berlin Heidelberg (Lecture Notes in Computer Science), pp. 113–122. Available at: https://doi.org/10.1007/978-3-540-87361-7_10.

Untergasser, A. *et al.* (2012) 'Primer3—new capabilities and interfaces', *Nucleic Acids Research*, 40(15), pp. e115–e115. Available at: https://doi.org/10.1093/nar/gks596.

Vaulot, D. *et al.* (2022) 'pr2-primers: An 18S rRNA primer database for protists', *Molecular Ecology Resources*, 22(1), pp. 168–179. Available at: https://doi.org/10.1111/1755-0998.13465.

Wright, E., S. (2016) 'Using DECIPHER v2.0 to Analyze Big Biological Sequence Data in R', *The R Journal*, 8(1), p. 352. Available at: https://doi.org/10.32614/RJ-2016-025.