Tutorial 1 – A complete pipeline to process chromatogram files in a cDNA sequencing project

Introduction

In this tutorial, we describe the specification of a pipeline that processes raw chromatogram files from a cDNA sequencing project. This pipeline is similar to the one that our group uses for ORESTES reads. ORESTES (ORF ESTs) are cDNA fragments synthesized by RT-PCR using 18 to 25-mer arbitrary primers under a low stringency (Dias-Neto *et al.* - Shotgun sequencing of the human transcriptome with ORF expressed sequence tags. *Proc. Natl. Acad. Sci. USA* **97**(7): 3491-3496, 2000). In this kind of ESTs, there is no need to mask and trim the polyA tails, since most of the reads span the central part of the transcripts. PolyA trimming will be covered and discussed in Tutorial 4. On the other hand, because ORESTES technique is based on the use of arbitrary primers to generate the cDNAs, the corresponding primer sequences should be masked.

The following steps constitute this pipe:

1. Uploading trace files and performing base calling and quality evaluation;
2. Masking primer sequences;
3. Masking vector sequences;
4. Filtering low quality sequences;
5. Saving sequences invalidated by the quality filter;
6. Trimming the bases that present a low Phred quality value and those that are masked;
7. Filtering sequences considered too small;
8. Saving sequences invalidated by the size filter;
9. Filtering mitochondrial sequences;
10. Saving sequences invalidated by the contaminant filter;
11. Filtering ribosomal sequences;
12. Saving sequences invalidated by the contaminant filter;
13. Saving sequences not previously invalidated by any filter;
14. Generating a report of all filtering steps;
15. Creating an XML snapshot recording all the processing steps that were performed;
16. Assembling the valid sequences using CAP3;
17. Generating an HTML page with graphical reports;
18. Generating a complete graphical report.

We have previously constructed a pipeline for this tutorial using CoEd, EGene's graphical configuration editor. The EGene's configuration file (`trace_file_complete.gen`) and its counterpart text file (`trace_file_complete.cnf`) can be found at the `config_files` directory. In order to run the pipeline, go to the `/examples/traces_complete_pipe` directory. This directory contains the subdirectory `chromat_dir`, which presents a set of trace files, and the file `primer_table.txt`, composed by a list of the primers used in the sequencing.

To run the pipe, you should type the following command:

```
bigou.pl –c ../config_files/trace_file_complete.cnf
```

If everything goes well, you should now find the following additional files in this directory:

```
filtered_by_quality.fasta
filtered_by_size.fasta
filtered_by_mitochondria.fasta
filtered_by_ribosome.fasta
filtering_report.html
redundancy_report.html
report_graphic_simple.html
final_snapshot.xml
good_sequences.fasta
```

and the following additional directories:

```
assembly_dir/
complete_report/
images_dir/
```

Understanding the pipeline and the component parameters:

The configuration file is actually a concatenation of the specification for all steps of the pipeline. A text section that starts with two parameters describes every step:

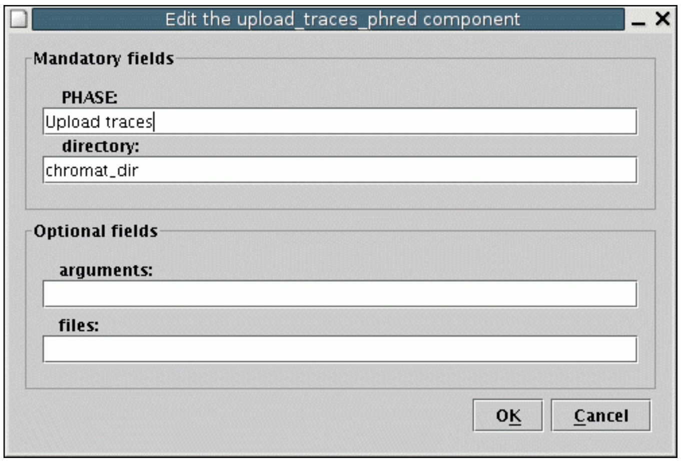
- `PHASE`: specifies a name for the pipeline step. Used by CoEd to label the icon representing the step.
- `program`: specifying the name of the component to be run at the step.

After this preamble, the parameters for the component are specified. Below we describe the parameters for each step of this pipeline. In this tutorial we will show, for each step of the pipeline, the text section of the configuration file corresponding to the step.

1. Uploading sequences in trace file format

Configuration parameters in the .cnf file:

```
#=============================================================
PHASE=Upload traces
program = upload_traces_phred.pl
#-------------------------------------------------------------
directory = chromat_dir
#=============================================================
```



Mandatory fields

PHASE: Upload traces

directory: chromat_dir

Optional fields

arguments:

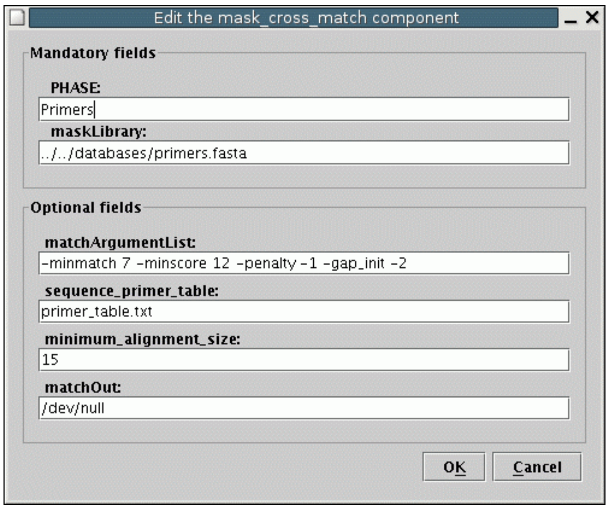
files:

This step uses the software Phred to perform base calling in a set of trace files. The only argument to this component is the directory that contains the trace files (in our case `chromat_dir`). It is assumed that `bigou.pl` is run while the shell is in the directory that contains `chromat_dir`. Alternatively, the user can specify a complete path for the directory (e.g. `/home/test/chromat_dir`).

2. Masking primer sequences

Configuration parameters in the .cnf file:

```
#==============================================================
PHASE=Primers
program = mask_cross_match.pl
#--------------------------------------------------------------
maskLibrary = ../../databases/primers.fasta
matchArgumentList = -minmatch 7 -minscore 12 -penalty -1 -
gap_init -2
sequence_primer_table = primer_table.txt
minimum_alignment_size = 15
matchOut = /dev/null
#==============================================================
```



Mandatory fields

PHASE: Primers

maskLibrary: ../../databases/primers.fasta

Optional fields

matchArgumentList: -minmatch 7 -minscore 12 -penalty -1 -gap_init -2

sequence_primer_table: primer_table.txt

minimum_alignment_size: 15

matchOut: /dev/null

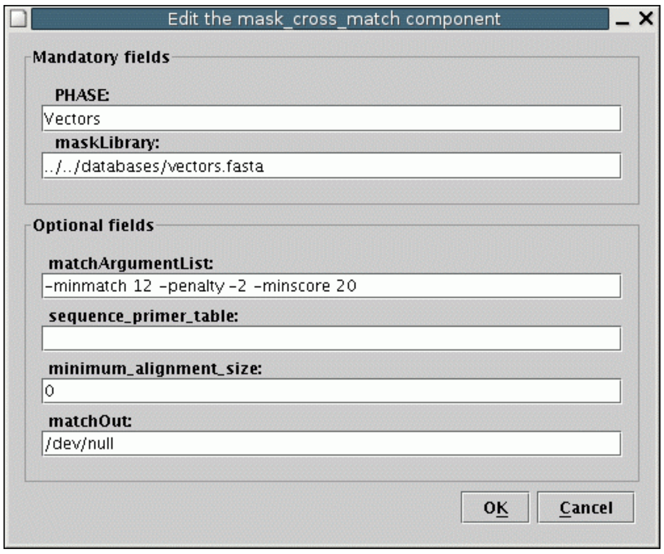
There are 5 parameters for this component:

- `mask_Library`: relative or complete path to the multiFASTA file that contains the primer sequences.
- `MatchArgumentList`: command line arguments for Cross_match program.
- `sequence_primer_table`: when the database of primer sequences contains more than the primers specifically used for the particular set of sequences being processed, the user can create a small file presenting a list of the names of the primers that should be looked for in the particular run of the pipeline. This argument describes where to find this file.
- `minimum_alignment_size`: setting appropriate arguments for Cross_match can be very problematic for small sequences, as stringent parameters would not identify the primer sequence, while loose parameters would lead to a unacceptable rate of false positive matches. This parameter specifies the minimum size of the alignment block for a primer sequence to be considered, allowing for less stringent command line arguments for Cross_match.
- `match_Out`: it is possible to save Cross_match's output in a specified file. This argument describes the name of the file and the path. Saving similarity search results from Cross_match can be particularly useful for monitoring purposes, especially when standardizing the parameters of the filtering stringency of a pipeline.

3. Masking vector sequences

Configuration parameters in the .cnf file:

```
#==============================================================
PHASE=Vectors
program = mask_cross_match.pl
#--------------------------------------------------------------
maskLibrary = ../../databases/vectors.fasta
matchArgumentList = -minmatch 12 -penalty -2 -minscore 20
minimum_alignment_size = 0
matchOut = /dev/null
#==============================================================
```



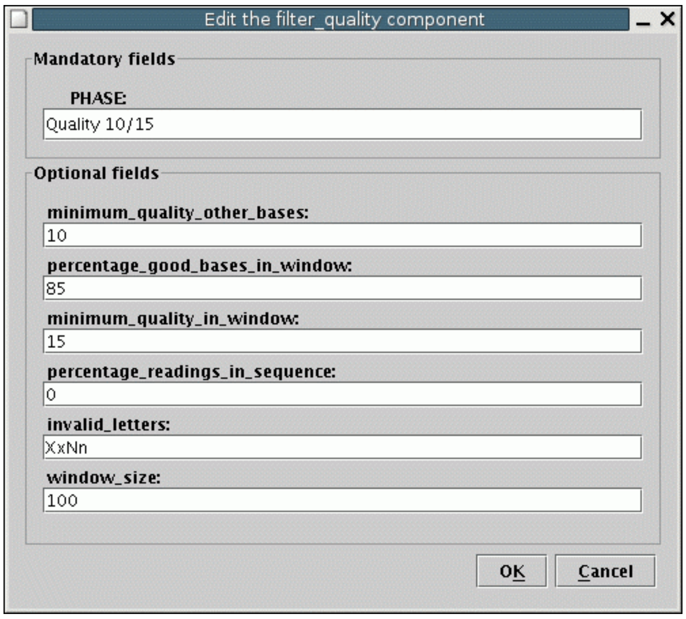
PHASE: Vectors
maskLibrary: ../../databases/vectors.fasta
matchArgumentList: -minmatch 12 -penalty -2 -minscore 20
sequence_primer_table:
minimum_alignment_size: 0
matchOut: /dev/null

Here we run the same component as in the previous step, but with different parameters. Since vector sequences are long, there is no need limit a minimum size for the alignment block. Thus, the `minimum_alignment_size` parameter can be set to zero.

4. Filtering low quality sequences

Configuration parameters in the .cnf file:

```
#==============================================================
PHASE=Quality 10/15
program = filter_quality.pl
#--------------------------------------------------------------
minimum_quality_other_bases = 10
percentage_readings_in_sequence = 0
percentage_good_bases_in_window = 85
minimum_quality_in_window = 15
invalid_letters = XxNn
window_size = 100
#==============================================================
```


Edit the filter_quality component

Mandatory fields

PHASE:
Quality 10/15

Optional fields

minimum_quality_other_bases:
10
percentage_good_bases_in_window:
85
minimum_quality_in_window:
15
percentage_readings_in_sequence:
0
invalid_letters:
XxNn
window_size:
100

This step filters out low quality sequences. The `filter_quality.pl` program uses a two-step approach. First, it checks the overall quality rate. For this purpose, the Phred quality values are computed for all bases and compared to the quality threshold (`minimum_quality_other_bases`). Those bases that present quality values above the threshold are classified as valid. The read itself is considered valid if the percentage of valid bases is higher than the value previously established by the parameter `percentage_readings_in_sequence`. As a second step, the program uses a sliding window and looks for a stretch of bases presenting a minimum percentage of valid bases, where valid bases are those presenting Phred quality above the number defined by the parameter `minimum_quality_in_window`. The second step is used to guarantee that the read includes a continuous region of high quality bases.

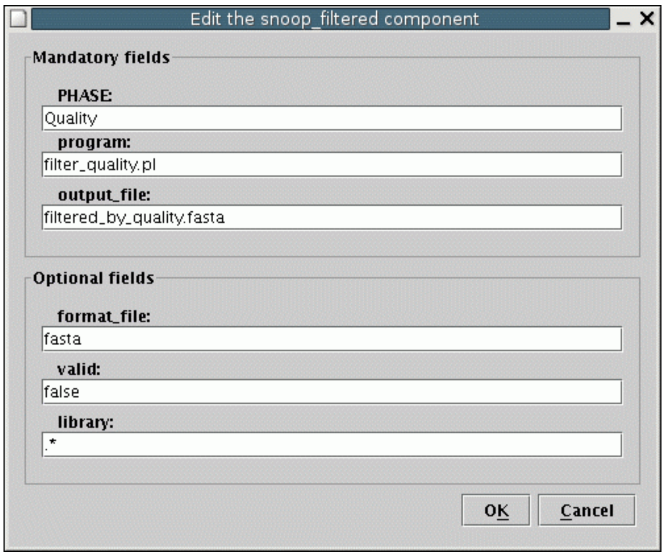
The parameters are:

- `minimum_quality_other_bases`: indicates what is the quality threshold for the overall evaluation of the sequence.
- `percentage_readings_in_sequence`: indicates what percentage of the bases need to have a quality value above the threshold in an overall evaluation. In this example, we have set the value to zero, indicating no restriction on the overall quality evaluation.
- `minimum_quality_in_window`: indicates the quality value threshold for the second step of the evaluation. This number is generally larger than that used for `minimum_quality_other_bases`.
- `percentage_good_bases_in_window`: minimum percentage of valid bases for the second step of the evaluation.
- `window_size`: indicates the window size to be used at the second step of the program. Summarizing, in this particular example, valid sequences will be those that present a continuous stretch of at least 100 bp, where at least 85% of the bases are not masked and present a Phred quality value above 15.
- `invalid_letters`: `filter_quality.pl` considers as "bad" those bases that present a Phred quality value below the appropriate threshold, as well as those that are explicitly indicated as invalid. In this example, we consider invalid bases those that are represented by the characters "x" (masked) or "n" (ambiguities). The "xXnN" characters indicate that either lower or upper case letters are considered.

5. Saving sequences invalidated by the quality filter

Configuration parameters in the .cnf file:

```
#==============================================================
PHASE=Quality
program = snoop_filtered.pl
#--------------------------------------------------------------
program = filter_quality.pl
output_file = filtered_by_quality.fasta
format_file = fasta
valid = false
library = .*
#==============================================================
```



Edit the snoop_filtered component

Mandatory fields

PHASE: Quality

program: filter_quality.pl

output_file: filtered_by_quality.fasta

Optional fields

format_file: fasta

valid: false

library: .*

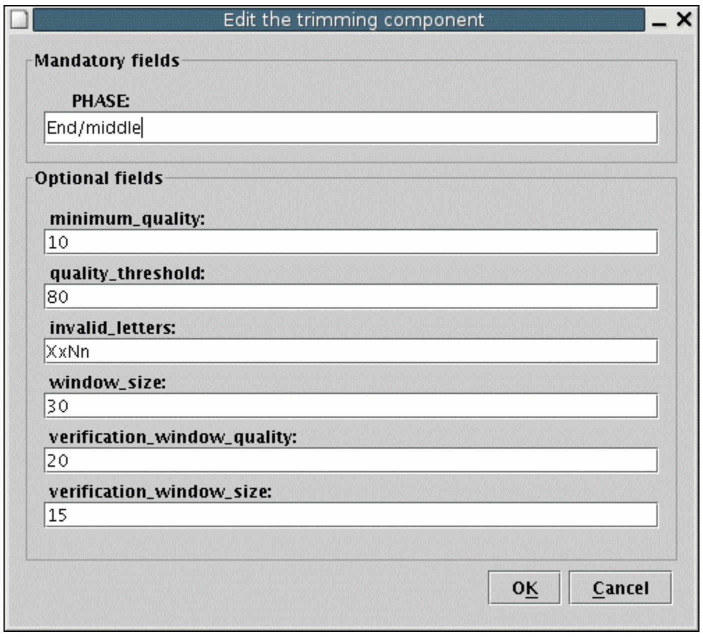
This step generates a FASTA file with the sequences of all reads invalidated in step four (quality filter). To do this, the parameters of the `snoop_filtered.pl` program are:

- `program`: `snoop_filtered.pl` can check for sequences invalidated by any filtering component. This argument specifies if we are interested on sequences that have been processed by some specific filtering program. In this case, we want to look at the sequences examined by `filter_quality.pl`.
- `output_file`: name of the output file.
- `format_file`: format of the output file. In this case the chosen format is `fasta`, but `xml` (XML) and `phd` (Phred PHD file) are also available options.
- `valid`: this argument specifies the output file should contain the sequences that were invalidated by the filtering program (`value = false`) or those that were considered valid (`value = true`).
- `library`: this argument is used when the filtering program uses some database (e.g. `filter_blast.pl`). This argument specifies the name of the database that originated the filtering. The default value is ".*", which indicates any database (the value is a Perl regular expression).

6. Trimming sequences

Configuration parameters in the .cnf file:

```
#=============================================================
PHASE=End/middle
program = trimming.pl
#-------------------------------------------------------------
minimum_quality = 10
quality_threshold = 80
invalid_letters = XxNn
window_size = 30
verification_window_quality = 20
verification_window_size = 15
#=============================================================
```


Edit the trimming component

Mandatory fields

PHASE:
End/middle

Optional fields

minimum_quality:
10
quality_threshold:
80
invalid_letters:
XxNn
window_size:
30
verification_window_quality:
20
verification_window_size:
15

After masking, we can now trim the bases that present either low quality, or masked, or have not ambiguity (n”) known value. `Trimming.pl` also works in two steps. The first step is end-trimming, where the goal is to trim low quality and invalid bases on the 5' and 3' ends of the sequence. The second step is a search for a region with no significant stretches of low quality bases in the middle. Both phases use sliding windows and are optional. The parameters are:

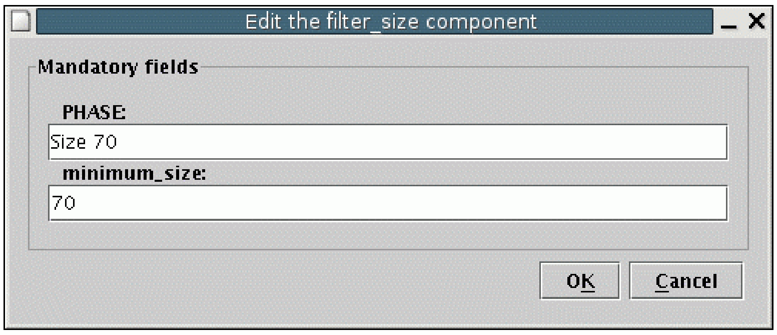
- `minimum_quality`: minimum Phred quality value for a base to be considered acceptable.
- `quality_threshold`: indicates the minimum percentage of good bases in a window for it to be considered acceptable.
- `window size`: this is the size of the sliding window at the first step. The program scans the sequence starting, respectively, at the 5' and 3' ends with a window of 30 bp. In either case the window is moved towards the center while the number of good bases is below the threshold. The program trims all bases between the window and the 5' (or 3') end of the sequence.
- `verification_window_quality`: this is the minimum quality value for the second step.
- `verification_window_size`: size of the window used in the second step.
- `verification_window_quality`: quality threshold for the second step. Here the sequence remaining after the first step is investigated with the second window. In this second step the regions where the sliding window finds the number of good bases below the threshold are marked. One of the remaining “good” parts is chosen, and the rest of the sequence is trimmed. The algorithm adds up the quality of all bases for each “good” part. The part with the highest sum is chosen.

You should consider the parameters used here only as a guideline. In our experience, each project needs a different setting due to different notions of what part of the sequence needs to be kept.

7. Filtering sequences by size

Configuration parameters in the .cnf file:

```
#==============================================================
PHASE=Size 70
program = filter_size.pl
#--------------------------------------------------------------
minimum_size = 70
#==============================================================
```



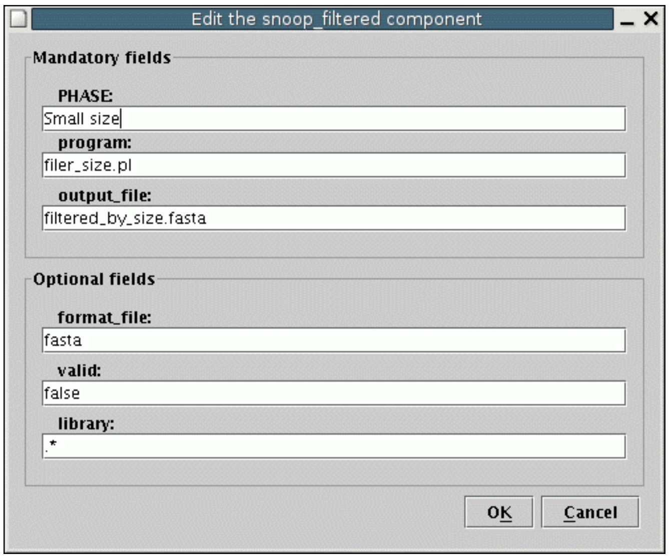
PHASE: Size 70
minimum_size: 70

After trimming, some sequences may have too few bases left. This component invalidates sequences that are too small to be useful. The only argument is `mininum_size`, stating the minimum acceptable size for a sequence.

8. Saving the sequences filtered by size

Configuration parameters in the .cnf file:

```
#==============================================================
PHASE=Small size
program = snoop_filtered.pl
#--------------------------------------------------------------
program = filter_size.pl
output_file = filtered_by_size.fasta
format_file = fasta
valid = false
library = .*
#==============================================================
```


Edit the snoop_filtered component

Mandatory fields

PHASE:
Small size

program:
filer_size.pl

output_file:
filtered_by_size.fasta

Optional fields

format_file:
fasta

valid:
false

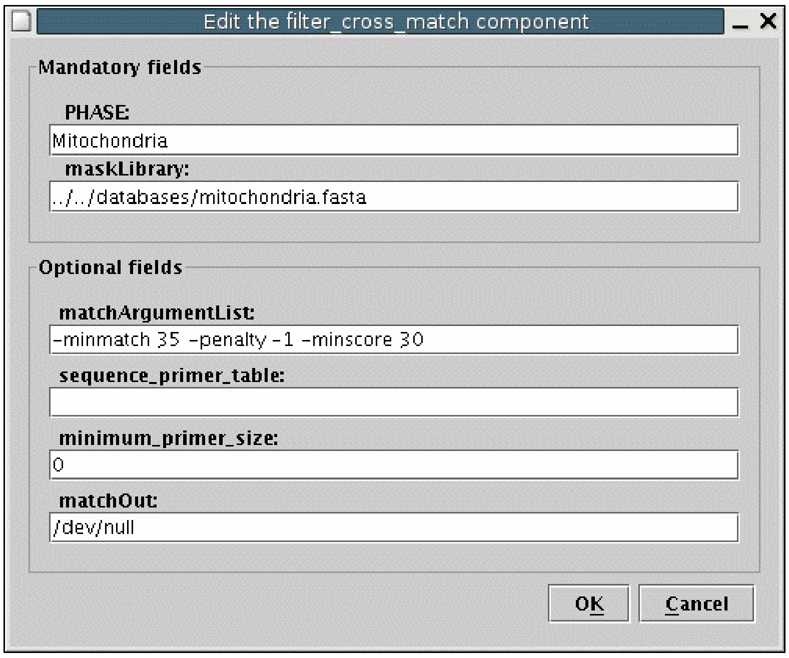
library:
.*

Here we use `snoop_filtered.pl` to save the sequences invalidated by the program `filter_size.pl` into the file `filtered_by_size.fasta`. Thus, the first two parameters are different than those used in the fifth step of the pipeline. All the other parameters are the same.

9. Filtering contaminants: mitochondrial sequences

Configuration parameters in the .cnf file:

```
#==============================================================
PHASE=Mitochondria
program = filter_cross_match.pl
#--------------------------------------------------------------
maskLibrary = ../../databases/mitochondria.fasta
matchArgumentList = -minmatch 35 -penalty -1 -minscore 30
matchOut = /dev/null
#==============================================================
```



Mandatory fields

PHASE: Mitochondria

maskLibrary: ../../databases/mitochondria.fasta

Optional fields

matchArgumentList: -minmatch 35 -penalty -1 -minscore 30

sequence_primer_table:

minimum_primer_size: 0

matchOut: /dev/null

This step invalidates reads originated from potential contaminant sources or undesired sequences. In this particular example, a pipeline designed for an EST sequencing project, we consider mitochondrial sequences as undesired. The program `filter_cross_match.pl` runs Cross_match and performs a similarity search against a database of undesired sequences. If a match is found, it invalidates the corresponding sequences. In this particular case, we chose to use `filter_cross_match.pl` because Cross_match is more sensitive than BLAST, and also because our mitochondria database is small enough to be used by Cross_match with an acceptable performance.

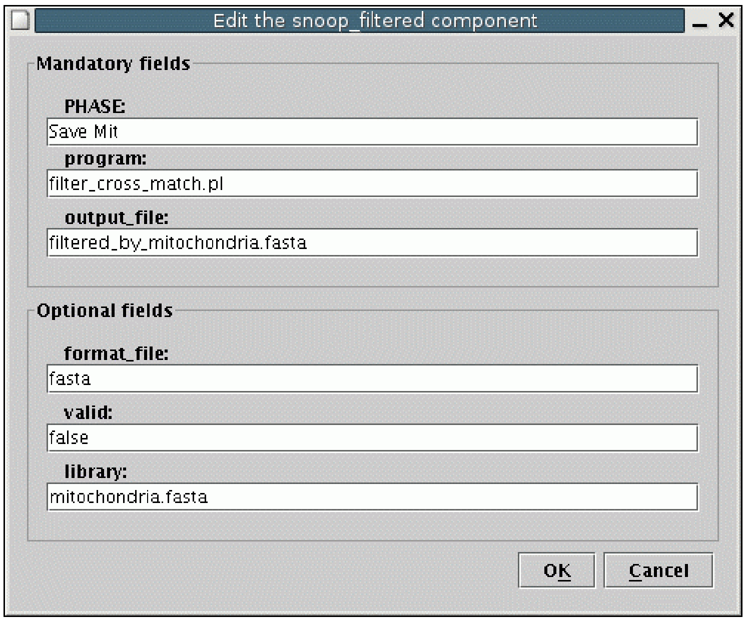
The parameters are:

- `maskLibrary`: name (relative or full path) of a multiFASTA file that contains the mitochodrial sequences.
- `matchArgumentList`: command line argument for Cross_match. These arguments were set using our previous experience, but can be modified by the user according to the desired stringency.
- `matchOut`: this argument specifies a file to store the Cross_match output of each search. In this case, we are discarding the results by redirecting them to `/dev/null`.

10. Saving mitochondrial sequences

Configuration parameters in the .cnf file:

```
#==============================================================
PHASE=Save Mit
program = snoop_filtered.pl
#--------------------------------------------------------------
program = filter_cross_match.pl
output_file = filtered_by_mitochondria.fasta
format_file = fasta
valid = false
library = mitochondria.fasta
#==============================================================
```


Edit the snoop_filtered component

Mandatory fields

PHASE: Save Mit

program: filter_cross_match.pl

output_file: filtered_by_mitochondria.fasta

Optional fields

format_file: fasta

valid: false

library: mitochondria.fasta

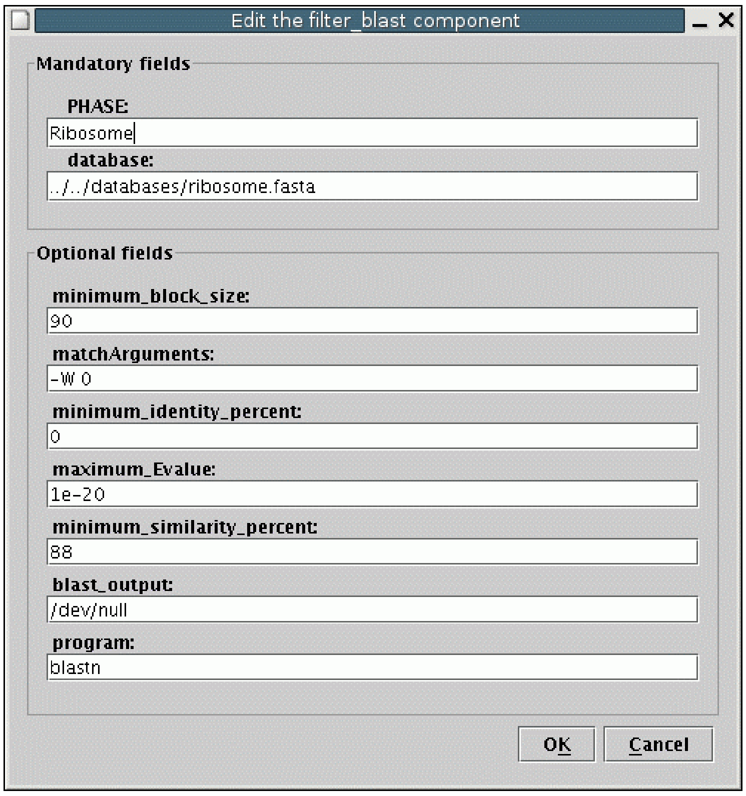
Here we use `snoop_filtered.pl` to store the mitochondria-derived sequences, invalidated in the previous step. Three parameters have settings different from those used previously (steps 5 and 8):

- `program`: this argument specifies the program that was used in the previous step to invalidate the sequences. In this example, we used `filter_cross_match.pl`.
- `output_file`: the invalidated sequences will be stored in the file named `filtered_by_mithocondria.fasta`.
- `library`: this corresponds to the name of the file containing the database of undesired sequences. In this case, we use `mitochondria.fasta` file as the database of mitochondrial sequences.

11. Filtering contaminants: ribosomal sequences

Configuration parameters in the .cnf file:

```
#==============================================================
PHASE=Ribosome
program = filter_blast.pl
#--------------------------------------------------------------
database = ../../databases/ribosome.fasta
minimum_block_size = 90
matchArguments = -W 0
minimum_identity_percent = 0
maximum_Evalue = 1e-20
minimum_similarity_percent = 88
blast_output = /dev/null
program = blastn
#==============================================================
```



Mandatory fields

PHASE:
Ribosome

database:
../../databases/ribosome.fasta

Optional fields

minimum_block_size:
90

matchArguments:
-W 0

minimum_identity_percent:
0

maximum_Evalue:
1e-20

minimum_similarity_percent:
88

blast_output:
/dev/null

program:
blastn

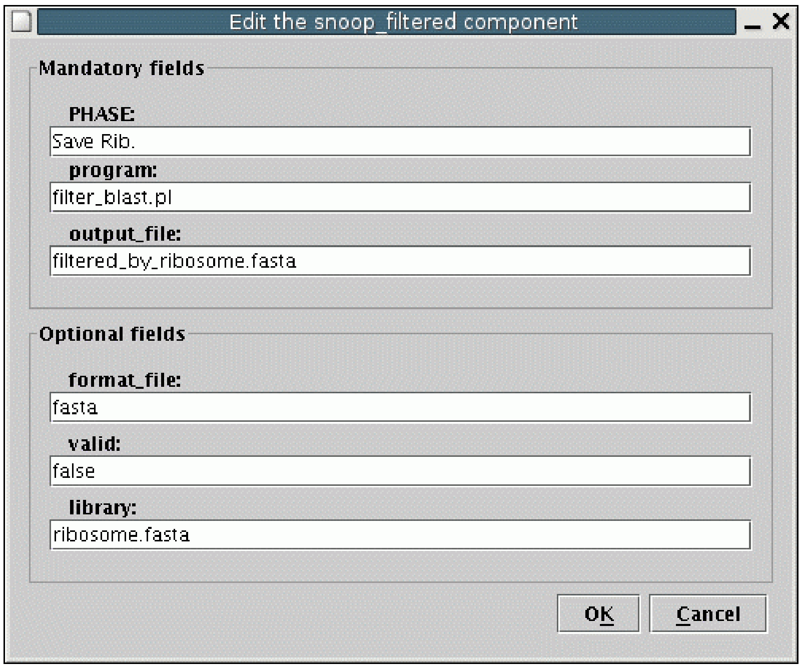
Again, we will perform a similarity-based filtering. However, this time the database is quite large, and Cross_match would present a poor performance compared to BLAST. Therefore, we employ `filter_blast.pl`, a component that uses BLAST as the third-party similarity search tool. The parameters are:

- `database`: indicates the name of the database file containing the ribosomal sequences. Notice that this file must be previously formatted by `formatdb`, a companion tool of BLAST package. The index files, created by `formatdb`, must be present at the same path as the database file, otherwise BLAST will abort execution.
- `minimum_block_size`: minimum size of the alignment block for it to be considered as a match by the filter.
- `mininum_similarity_percent`: minimum value for the similarity of the alignment for it to be considered as a match by the filter. In case of filtering based on protein alignments, one can also use the parameter `minimum_identity_percent`.
- `maximum_Evalue`: maximum E-value of a match for it to be considered as a match by the filter. In this case, any sequences presenting alignment with E-values lower than 10^{-20} will be filtered out.
- `Match_arguments`: any additional BLAST command line arguments can be used here.
- `blast_output`: this argument specifies a file to store the BLAST output of each search. In this case, we are discarding the results by redirecting them to `/dev/null`.
- `program`: program to be run by `blastall` (BLAST command line program). In this case, we use `blastn` (query and subject correspond to nucleotide sequences). Additional available options include `blastp`, `blastx`, `tblastx` and `tblastn`.

12. Saving ribosomal sequences

Configuration parameters in the .cnf file:

```
#==============================================================
PHASE=Save Rib.
program = snoop_filtered.pl
#--------------------------------------------------------------
program = filter_blast.pl
output_file = filtered_by_ribosome.fasta
format_file = fasta
valid = false
library = ribosome.fasta
#==============================================================
```


Edit the snoop_filtered component

Mandatory fields

PHASE: Save Rib.

program: filter_blast.pl

output_file: filtered_by_ribosome.fasta

Optional fields

format_file: fasta

valid: false

library: ribosome.fasta

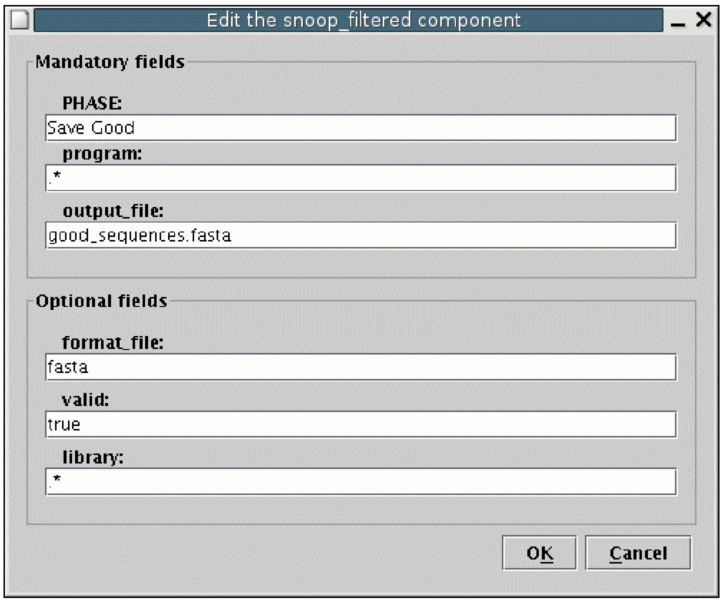
The parameters are:

- Here we use `snoop_filtered.pl` to store the ribosome-derived sequences, invalidated in the previous step. The arguments are similar to the previous use of this component, but the filtering program now is `filter_blast.pl`.
- `library:` the database file is `ribosome.fasta.`
- `output_file:` the invalidated sequences will be stored in the file named `filtered_by_ribosome.fasta`.
- `valid`: we are interested again in sequences invalidated by `filter_blast.pl`, so this parameter should be set to `false.`
- `format:` we want a multiFASTA file, so we choose `fasta` in this parameter.

13. Saving valid sequences

Configuration parameters in the .cnf file:

```
#==============================================================
PHASE=Save Good
program = snoop_filtered.pl
#--------------------------------------------------------------
program = .*
output_file = good_sequences.fasta
format_file = fasta
valid = true
library = .*
#==============================================================
```


Edit the snoop_filtered component

Mandatory fields

PHASE: Save Good

program: .*

output_file: good_sequences.fasta

Optional fields

format_file: fasta

valid: true

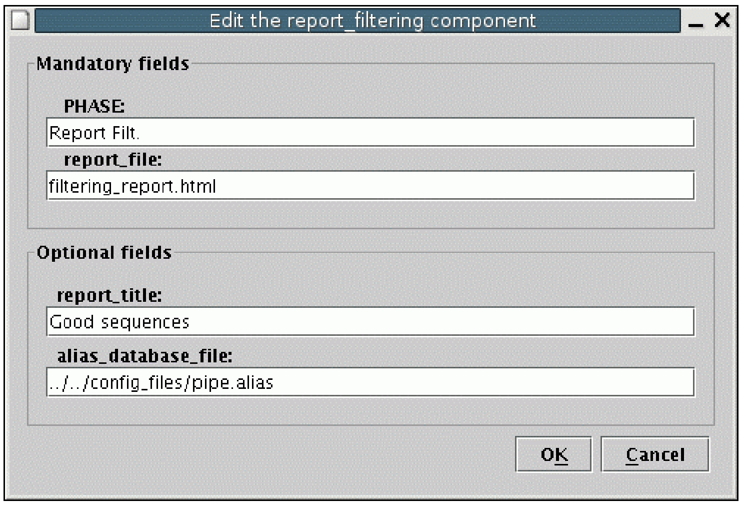
library: .*

- In this step, `snoop_filtered.pl` is used to save the valid sequences. In order to save the valid sequences, we set the `valid` parameter to the value `true`. The valid sequences will be stored in the file named `good_sequences.fasta`.
- Because there is no other program nor database associated with this step, the `program` and `library` parameters should both be set to ".*".

14. Producing a report of all filtering steps

Configuration parameters in the .cnf file:

```
#==============================================================
PHASE=Report Filt.
program = report_filtering.pl
#--------------------------------------------------------------
report_file = filtering_report.html
report_title = Good sequences
alias_database_file = ../../config_files/pipe.alias
#==============================================================
```



Optional fields

PHASE: Report Filt.
report_file: filtering_report.html
report_title: Good sequences
alias_database_file: ../../config_files/pipe.alias

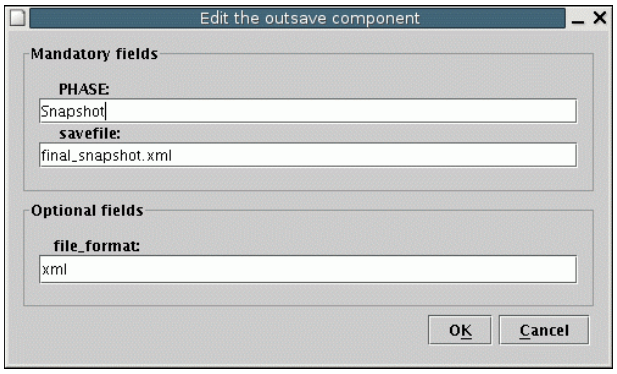
The component `report_filtering.pl` produces an HTML report indicating how many sequences were filtered in each previous filtering steps of the pipeline. The parameters are:

- `report_file`: name of the HTML file that will store the results.
- `report_title`: title of the last entry in the table, indicating the number of files that have not been filtered.
- `alias_database_file`: name of the database alias file. Database files that are used by the similarity filtering steps not always have nice names. The user can create a file indicating how each database name should be printed in the filtering report. An example of an alias file is `pipe.alias`, located in the `config_files` directory. Like all parameters specifying file names, the names can indicate the complete or relative path of the file. In this example, a relative pathname has been specified.

15. Creating an XML snapshot recording all the processing steps that were performed;

Configuration parameters in the .cnf file:

```
#==============================================================
PHASE=Snapshot
program = outsave.pl
#--------------------------------------------------------------
savefile = final_snapshot.xml
file_format = xml
#==============================================================
```



PHASE: Snapshot
savefile: final_snapshot.xml
file_format: xml

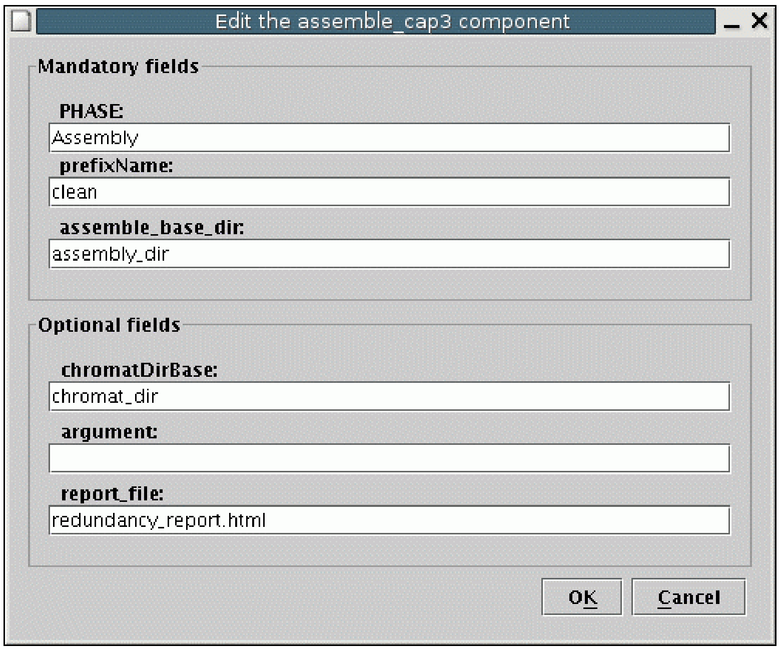
The component `outsave.pl` creates a snapshot of all the processing steps that have been performed along the pipeline. This component can be used to generate a flatfile database. The resulting file can be fed into a new pipeline using the `upload_xml.pl` component (see Tutorial 3). The result is the same as continuing the processing from the point where the snapshot has been generated. This component can be used to implement forks in the pipeline. We create a snapshot just before the "fork" of the workflow, generate a separate pipe specification for each of the "branches" of the fork, and run each of these new pipelines using the snapshot produced. The parameters are:

- `savefile`: name of the snapshot file.
- `file_format`: format of the output file. The default value, specified here, is XML, but `fasta` and `phd` (Phred PHD file) are also available options. When we use `xml`, in fact we are generating a flatfile database of all processing done before this step.

16. Assembling the valid sequences

Configuration parameters in the .cnf file:

```
#==============================================================
PHASE=Assembly
program = assemble_cap3.pl
#--------------------------------------------------------------
assemble_base_dir = assembly_dir
prefixName = clean
chromatDirBase = chromat_dir
report_file = redundancy_report.html
#==============================================================
```


Edit the assemble_cap3 component

Mandatory fields

PHASE: Assembly

prefixName: clean

assemble_base_dir: assembly_dir

Optional fields

chromatDirBase: chromat_dir

argument:

report_file: redundancy_report.html

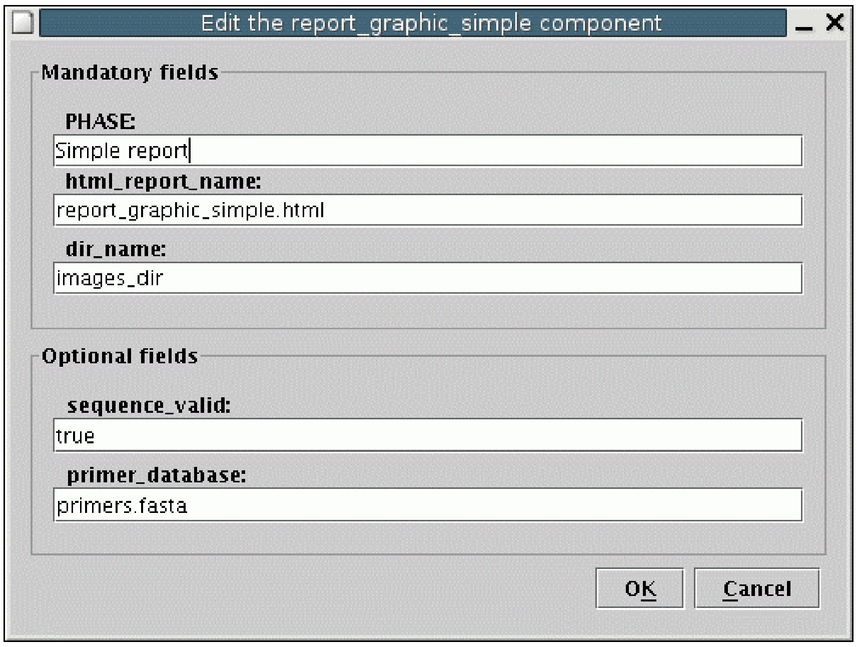
In this step, we will assemble the valid sequences using the program CAP3. The component that performs this task is `assemble_cap3.pl`. Alternatively, the user can choose Phrap as the assembler. For doing so, the component `assemble_phrap.pl` should be used instead (see Tutorial 3). The component `assemble_cap3.pl` does not alter any sequence, but rather generates new files. Thus, it can also be seen as a report component. The parameters are:

- `assemble_base_dir`: the output of the assembly is stored in a directory structure. The topmost directory is specified in this parameter. Therefore, in our case, all the assembly output will be found in the `assembly_dir` directory. Three sub-directories will be also generated: `chromat_dir`, `edit_dir` and `phd_dir`. These directories follow the structure required by Consed, a viewer and editor for assembled sequences.
- `prefixName`: during assembly, CAP3 generates various files containing assembly data. All these files will present the name specified in this argument as a prefix (e.g. `clean` in this example).
- `chromatDirBase`: if chromatogram files are available, they can be copied into `chromat_dir` directory, so the traces can be displayed by Consed. In fact, in order to save disk space, we use a UNIX command to create symbolic links.
- `report_file`: an HTML report file is generated after the assembly. This file contains information on the number of contigs and singlets, total number of distinct assembled events and the overall redundancy. The `report_file` parameter specifies the name of that file, in this example `redundancy_report.html`.

17. Generating an HTML page with graphical reports

Configuration parameters in the .cnf file:

```
#==============================================================
PHASE=Simple report
program = report_graphic_simple.pl
#--------------------------------------------------------------
html_report_name = report_graphic_simple.html
dir_name = images_dir
sequence_valid = true
primer_database = primers.fasta
#==============================================================
```


Edit the report_graphic_simple component

Mandatory fields

PHASE:
Simple report

html_report_name:
report_graphic_simple.html

dir_name:
images_dir

Optional fields

sequence_valid:
true

primer_database:
primers.fasta

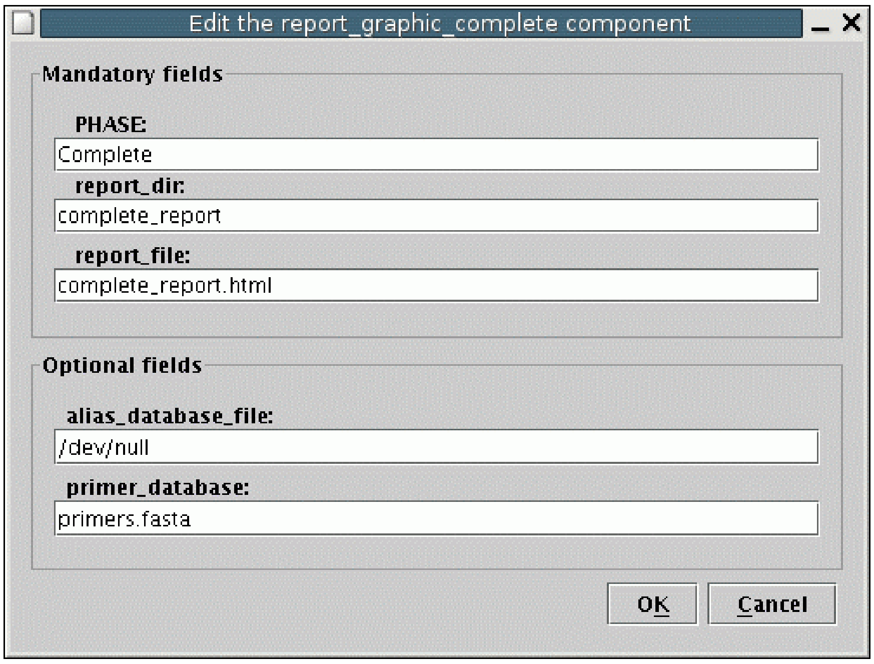
This component generates an HTML file containing a graphical report of each read. The report displays a base quality graph with horizontal bars that indicate the masked and trimmed regions of sequence. The parameters are:

- `html_report_name`: name of the HTML output file.
- `dir_name`: this argument specifies the name of the directory that will store all the graphic files (created in PNG format).
- `sequence_valid`: the report can be generated for both valid and invalid sequences. In our example, we want the graphic files for the valid sequences, so the paramater is set to `true`.
- `primer_database`: the graphical report indicates all the maskings with horizontal bars. When masking primers, the orientation of the respective oligonucleotide is displayed. This argument states the name of the primer database. All masking performed due a match against this database will be displayed as a red horizontal bar, with the arrowhead indicating the primer orientation. Notice that this parameter refers solely to the name of the database specified in the masking component. There is no need to declare the path.

18. Generating a complete graphical report

Configuration parameters in the .cnf file:

```
#=============================================================
PHASE=Complete
program = report_graphic_complete.pl
#-------------------------------------------------------------
report_dir = complete_report
report_file = complete_report.html
alias_database_file = /dev/null
primer_database = primers.fasta
#=============================================================
```



Edit the report_graphic_complete component

Mandatory fields

PHASE: Complete

report_dir: complete_report

report_file: complete_report.html

Optional fields

alias_database_file: /dev/null

primer_database: primers.fasta

The complete graphical report is an HTML page containing, for each sequence processed by the pipeline, a link to a page that contains all the information of the simple report described above, with the addition of the following data:

- An indication if the sequence is valid or not.
- For invalid sequences, the program responsible for the invalidation and the configuration parameters that were used.
- The names of the sequences that presented a match during the masking process and the coordinates of the masked regions.
- A bases report presenting the following information: read size and number of clean, masked and trimmed bases.

The parameters are:

- `report_dir`: directory that will contain all the files used in this report (including the main report file).
- `report_file`: name of the HTML report file. This file contains a list of read names, with an indication if the sequence is valid or not. Each read name is also a link to a page displaying the corresponding graphical report.
- `alias_database_file`: name of the database alias file (see step 14).
- `primer_database`: similar to the previous step (17).