**Commet**

Comparing and combining metagenomic datasets V1.0

User’s guide – July 2014

|  |  |
| --- | --- |
| **Authors:**  Nicolas Maillet  Guillaume Collet  Thomas Vannier  Dominique Lavenier  Pierre Peterlongo | **Contact:**  [pierre.peterlongo@inria.fr](mailto:pierre.peterlongo@inria.fr) |

# Licence

Copyright (C) 2014 INRIA

This program is free software: you can redistribute it and/or modify it under the terms of the GNU Affero General Public License as published by the Free Software Foundation, either version 3 of the License, or any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

# Publication

In prep.

# Commet description

Commet enables the comparison of huge metagenomic datasets represented as files containing reads from metagenomic experiments. Given two read sets *A* and *B*, read from *A* that are similar enough with at least a read from *B* are outputed. Commet stores vectors of bits (called **bit vector**) that represent the selected reads. This representation saves space and enables efficient and easy logical operation between sets of selected reads.

# Quick Start

Given a set of read sets formatted as follow, just use the python command for filtering and comparing all read sets:

python Commet.py read\_sets.txt

with read\_sets.txt file formatted as follows:

*Name\_set1: path\_to\_read\_set1.1.fq.gz; path\_to\_read\_set\_1.2.fq.gz; path\_to\_read\_set1.3.fq.gz;…*

*Name\_set2: path\_to\_read\_set2.1.fq.gz; path\_to\_read\_set\_2.2.fq.gz*

*Name\_set3: path\_to\_read\_set3.1.fq.gz; path\_to\_read\_set\_3.2.fq.gz; path\_to\_read\_set3.3.fq.gz;…*

*…*

This will create an *“output\_commet”* directory containing vectors (.bv files) corresponding to read shared between read sets and containing dendrograms and heatmat matrices summing-up results.

To get the real corresponding reads in a fasta or fastq fashion, use the *extract\_reads* tool.

# Commet software

For comparing *n* sets of reads, the Commet approach consists in two main mandatory steps:

1. Filter the reads given some parameters (create a .bv file per read file).
2. Select reads that are similar between each set pairs (create a .bv file per intersection).

After these two mandatory steps, two optional steps may be performed:

1. Intersect results using simple Boolean operations (basically AND, OR and NOT).
2. The .bv file of interest may be transformed back to real read files (fastq or fasta, gzipped or not).

These four tasks are implemented in four programs called **filter\_reads** (step 1), **index\_and\_search** (step 2), **bvop** (step 3) and **extract\_reads** (step 4).

**Filter\_reads** corresponds to the Commet filtering step. Each read is filtered on three parameters: its size, its N content and its Shannon entropy index (see below). Filter\_reads produces a bit vector that represents the selected reads. Additionally, Filter\_reads can limit the number of reads of each read set. This is useful when sub-sampling read sets.

**Index\_and\_search** corresponds to the comparison step. It inputs a set of query read files (*Q1, Q2, …, Qm*) and a bank read file *B*. Reads from *B* are indexed then each read from each query file is conserved if it is considered as similar to at least a read from set *B*.

**bvop** (bit vector operators) allows to combine bit vectors through the following Boolean operators: AND, OR, NOT, ANDNOT.

**Extract\_reads** extracts reads from a given file based on a given bit vector, and writes them in an output file.

# Pipelining filter\_reads and index\_and\_search

The Commet core task combines the usage of the filtering and the comparison steps. To this end, we propose the *Commet.py* tool.

*Commet.py* **inputs** a file containing sets of reads (each set of reads being possibly a virtual concatenation of read files – see next section). It filters each set of reads given user-defined parameters. For each couple of read sets *A* and *B*, it computes the reads of *A* similar to a read of *B* and *vice versa.* The Commet.py tool factorizes when possible indexation of read files (if *A* has to be compared to more than two read sets, then it is indexed only once).

*Commet.py* manages SGE clusters parallelizing the independent computations, dealing with job dependencies.

*Commet.py* **outputs** the obtained bit vectors. Additionally, it proposes several matrices summing up the distances between sets of reads either as csv files and as heatmaps in pdf files. It also clusterizes read sets and output dendograms of input read sets.

These results are provided as:

* **plain**: total number of shared reads
* **percentage**: directed measure. For instance for reads of a set *A* in a set *B: 100 \* nb reads of A shared with B / nb reads in A*
* **normalized**: undirected measure. For instance for read sets *A* and *B:* 100 \* (*nb reads of A shared with B + nb reads of B shared with A*) / (*nb reads in A + nb reads in B*)

**Commet.py Usage:**

*Commet.py [-h] [--sge] [-b] [-o] [-k K] [-t T] [-l L] [-n N] [-e E] [-m M] input\_file*

***positional arguments:***

*input\_file input read sets (a line = a set composed by “set\_name : read\_file ; read\_file...")*

***optional arguments:***

*-h, --help shows this help message and exit*

*--sge indicates the usage of SGE cluster commands*

*-b B binary directory [default: "./bin"]*

*-o O output directory [default: "output\_commet"]*

*-k K kmer size [default: 32]*

*-t T minimal number of shared k-mers [default: 2]*

*-l L minimal length a read should have to be kept [default: 0]*

*-n N maximal number of Ns a read should contain to be kept. [default: any]*

*-e E minimal Shannon index a read should have to be kept. Float in [0,2]. [default: 0]*

*-m M maximum number of selected reads in sets [default: all]*

**Be careful:** the *-m* option applies to a full set of reads: if a set is composed by 3 read files, and m=600, then the first 200 reads from each read file will be treated.

# Virtual concatenation of read sets

Often, a set of reads specific to an experiment is spread over several read files. A classical dirty solution consists in explicitly concatenating the related read sets, generating large and possibly numerous files. We propose a solution avoiding such a concatenation. The Commet input consists in a text file containing on each line a set of reads composed by a *virtual concatenation* of any number of read files. In practice, each line contains first the read set name (user defined), a ‘:’ symbol and then the list of related read files separated by a ‘;’ character. For instance:

*Name\_set1: set1.1.fq.gz; set\_1.2.fq.gz; read\_set1.3.fq.gz;…*

*Name\_set2: set2.1.fq.gz; set\_2.2.fq.gz*

*Name\_set3: set3.1.fq.gz; set\_3.2.fq.gz; set3.3.fq.gz;…*

*…*

Thus the read files indicated on each line are considered as a unique set called *Name\_set1, Name\_set2, Name\_set3..*.

If a user already disposes the bit vectors corresponding to read files sub-sampling, they can be indicated by adding for each read file a ‘,’ character and the corresponding .bv file:

*Name\_set1: set1.1.fq.gz, set1.1.bv ; set\_1.2.fq.gz, set1.2.bv; set1.3.fq.gz, \_set1.3.bv;…*

*Name\_set2: set2.1.fq.gz, set2.1.bv ; set\_2.2.fq.gz, set2.2.bv*

*Name\_set3: set3.1.fq.gz, set3.1.bv ; set\_3.2.fq.gz, set3.2.bv; set3.3.fq.gz, \_set3.3.bv;…*

**Be careful:** the read set names should not contain the ‘~’ symbol.

## Filter\_reads

Filter\_reads inputs a file containing reads (fasta or fastq, gzipped or not) and filters them on three parameters, their length, their N content (number of unknown bases), and their Shannon entropy index. The output file contains the bit vector corresponding to selected reads.

**Usage:**

./filter\_reads <input\_file> [options]

**Input:**

The input file need to be in a well-formed **fasta or fatsq** format, compressed with **gzip or not** (errors often comes from bad formatted files).

**Output:**

The output file is a bit vector that represents the selected reads in the input file. The size of the bit vector is the number of reads in the input file. The default output file name is the input file name with .bv extention. The user may also specify the output file name with –o option.

**Options:**

* -l int : minimal length a read should have to be kept [default=0].
* -n int : maximal number of Ns a read should contains to be kept [default=infinite].
* -e float : minimal Shannon index a read should have to be kept [default=0].
* -c string : the given string will be paste in the header of the output file.
* -m int : maximum number of selected reads [default=all].
* -o string : the output file name [default=stdout].
* -h : prints this help.
* -v : prints the version number.

## Compare\_reads

Compare\_reads takes two sets of files containing reads and finds the common reads between the two sets. Two reads are considered similar if they share a given number of identical non-overlapping k-mers. Each file may be associated to a .bv file (bit vector) that represents the previously filtered reads.

**Be careful:** The sets of reads given in input are supposed to be filtered by filter\_reads. No filter is made in compare\_reads on the size nor the complexity of reads.

**Usage:**

./compare\_reads –a <file[,bv]> -b <file[,bv]> [options]

**Input:**

Compare\_reads takes two sets (*A* and *B*) of files containing reads.

Input files need to be in a well-formed **fasta or fatsq** format, compressed with **gzip or not** (errors often comes from bad formatted files).

The files of set *A* are declared with the –a flag. The files of set *B* are declared with the –b flag.

Input files may have an associated bit vector. A bit vector associated to a file is declared after a comma.

**Output:**

For each input file *a* in *A*, a bit vector is written in an output file named **a\_in\_B.bv** that corresponds to reads from *a* found in *B*. Similarly, for each input file *b* in *B*, a bit vector is written in an output file named **b\_in\_A.bv** that corresponds to reads from *b* found in *A*.

**Be careful:** If two input files have the same basename, one will write on the other because the basename only is used to generate the output file names. This may happen for files having the same name in different directories.

A log file, containing information about the comparison, is also written in file A\_VS\_B.txt.

**Options:**

* -k int : size of k-mers (value of k) [default=33].
* -t int : minimal number of shared non overlapping k-mers [default=2].
* -m in t : maximum number of reads to read per file [default=all]
* -l string : path to write log file [default=./].
* -o string : path to write output files [default=./].
* -h : prints this help.
* -v : prints the version number.

## Extract\_reads

Extract\_reads inputs a file containing reads and its associated bit vector, then it outputs the selected reads in an output file in the same format than the input file.

**Usage:**

./extract\_reads <input\_file> <input\_bv> [options]

**Input:**

The input file needs to be in a well-formed **fasta or fatsq** format, compressed with **gzip or not** (errors often comes from bad formatted files).

The input\_bv is the associated bit vector file. The bit vector size must be exactly the number of reads in the input file.

**Output:**

Extract\_reads outputs reads, from the input file, that are selected in the bit vector. The default ouput is the standard output, use –o option to specify an output file.

**Options:**

* -o string : name of the output file [default=stdout].
* -h : prints this help.
* -v : prints the version number.

## Bvop

Bvop is designed to perform Boolean operations between bit vectors. It takes a bit vector file and an optional operation to perform on a second bit vector file. If no operation specified, it just does nothing. Option –I prints the comment and some statistics about the input file.

**Usage:**

./bvop <input\_file.bv> [options]

**Input:**

Input files are bit vector files generated by compare\_reads, filter\_reads or bvop. A bit vector contains a header with comments, then a line with a # and the size of the vector (number of reads), finally the vector of bits (binary format).

**Output:**

The output file contains the result of the Boolean operation applied to the input file(s).

**Options:**

* -n : performs **NOT** on the input\_file.bv.
* -a <file2.bv> : performs **AND** between input\_file.bv and file2.bv.
* -o <file2.bv> : performs **OR** between input\_file.bv and file2.bv.
* -d <file2.bv> : performs **ANDNOT** between input\_file.bv and file2.bv.
* -p <output.bv> : print result in file output.bv [Default=stdout].
* -i : prints information about input\_file.bv.
* -h : prints this help.
* -v : prints the version number.