**Commet**

comparing and combining metagenomic datasetsV1.0

User’s guide – July 2014

|  |  |
| --- | --- |
| **Authors:**  Nicolas Maillet  Claire Lemaître  Rayan Chikhi  Dominique Lavenier  Pierre Peterlongo  Guillaume Collet | **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.

# Software description

Commet has been developed to enable the comparison of huge metagenomic datasets (files containing reads from metagenomic experiments). Commet is based on the Compareads methology whose efficiency is due to a compact data structure: the **Bloom filter** (Bloom, 1970). Each read is split in k-mers, which are then stored in a Bloom filter by 4 bits, thus reducing the memory footprint. Commet stores a vector of bits (called **bit vector**) that represents the selected reads. This representation saves space and enable efficient and easy logical operation between sets of selected reads.

To compare n sets of reads, the Commet approach consists in two 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 steps, the user has the possibility to intersect results using simple Boolean operations (basically AND, OR and NOT). Finally, the .bv file of interest (and others!) can be transformed back to real read files (fastq or fasta, gzipped or not depending on the input files format)

These four tasks are implemented in four programs called **filter\_reads** (setp I), **index\_and\_search** (step II), **bvop** (logical operations) and **extract\_reads** (get back to the reads).

**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 and 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 is composed by the usage of the filtering and the comparison steps. … BLA BLA BLA

# Concatenating read sets

BLA BLA (posisiblité defaire une concatenation “virtuelle”) 🡪 donner format

## Filter\_reads

Filter\_reads takes a file containing reads and filters them on three parameters, their length, their contents in N (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 or 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

**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 takes 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.