Commet

Comparing and combining metagenomic datasets V1.0

User's guide – April 2015

Authors:
Nicolas Maillet
Guillaume Collet
Thomas Vannier
Dominique Lavenier
Pierre Peterlongo

Contact:
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: comparing and combining multiple metagenomic datasets"

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 one read from *B* are output. Commet stores vectors of bits (called **bit vector**) that represent the selected reads. This representation saves space and enables efficient and easy logical operations 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 a dendrogram and three heatmap matrices summing-up results.
To get the real corresponding reads in a fasta or fastq fashion, use the *extract_reads* tool.

Commet software

To compare *N* sets of reads, the Commet approach consists in two main mandatory steps:

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

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

3. Intersect results using simple logical operations (basically AND, OR and NOT).
4. 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 and each read from each query file is processed. A read from a 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. 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, it is indexed only once.

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

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

These matrices result are provided as:

- **plain**: total number of shared reads
- **percentage**: directed measure. For instance for reads of a set *A* in a set *B: number of reads of A shared with B / number of reads in A*

- **normalized**: undirected measure. For instance for read sets *A* and *B:*
  100 * (*number of reads of A shared with B + number of reads of B shared with A*) / (*number of reads in A + number of 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	*k-mer size [default: 33]*
-t T	*minimal number of shared k-mers [default: 2]*
-l L	*minimal length a read should have to be kept [default: k*t]*
	*if l=0: no filtration, else l must be bigger or equal to k*t*
-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, 2.32]. [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 colon (:), then the list of related read files separated by semicolons (;). 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 comma (,) 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. Moreover, file's path must be absolute.

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. Moreover, *filter_reads* can also limit the number of output reads.

Usage:

```
./filter_reads <input_file> [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).

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` extension. 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.

Index_and_search

Index_and_search takes two files containing read sets. One contains the reference read set - only the first line (i.e. the first read set) is taken into consideration. The other contains the queries read sets (all of them are compared to the reference read set). *Index_and_search* finds from queries the reads detected as similar to a read from the reference. Two reads are considered similar if they share a minimal 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 *index_and_search*, neither on the size nor on the complexity of reads.

Usage:

```
./index_and_search –i <file.txt> -s <file.txt> [options]
```

Input:

The input files are given by the `–i` and `–s` options. `–i` specifies the reference read set file (index). `–s` specifies the query read set file (search). Each input file must be formatted as presented in previous section (virtual concatenation of read sets).

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

Output:

For each file in query read set, a bit vector corresponding to reads similar to at least a read from the reference read set.

Options:

- `-l string`: path to write log files [default=./].
- `-o string`: path to write output files [default=./].
- `-k int`: size of k-mers (value of k) [default=33].
- `-t int`: minimal number of shared non overlapping k-mers [default=2].
- `-f`: full comparison of the index set and the first search set [default=false].
- `-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 output 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 logical 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 *filter_reads*, *index_and_search* 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 bit vector contains the result of the logical operation applied to the input bit vector(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.