demultiplex Documentation

Release stable

Mar 23, 2020

Contents:

Versatile NGS demultiplexer with the following features:

- Support for FASTA and FASTQ files.
- Support for multiple reads per fragment, e.g., paired-end.
- Handles barcodes in the header and in the reads.
- Handles barcodes at *unknown* locations in reads (e.g., PacBio or Nanopore barcodes).
- Support for selection of part of a barcode.
- Allows for mismatches, insertions and deletions.
- Barcode guessing by frequency or fixed amount.
- Handles large numbers (over one million) of barcodes.

Please see ReadTheDocs for the latest documentation.

CHAPTER 1

Installation

The software is distributed via PyPI, it can be installed with `pip`:

```
pip install demultiplex
```

1.1 From source

The source is hosted on GitHub, to install the latest development version, use the following commands.

```
git clone https://github.com/jfjlaros/demultiplex
cd demultiplex
pip install .
```

CHAPTER 2

Usage

The `demultiplex` program provides several ways to demultiplex any number of FASTA or a FASTQ files based on a list of barcodes. This list can either be provided via a file or guessed from the data. The demultiplexer can be set to search for the barcodes in the header, or in the read itself. To allow for mismatches, two distance functions (edit distance and Hamming distance) are available.

2.1 Illumina FASTQ files

For Illumina FASTQ files, the barcodes can usually be found in the header of each FASTQ record. Currently, the `demultiplex` program supports two types of headers, the classical Illumina headers and the newer HiSeq X headers. These headers are detected automatically.

Demultiplexing is done with the `demux` subcommand by providing a list of barcodes. The barcodes file is formatted as follows:

```
name sequence
```

So a typical barcodes file might look like this:

```
index1 ACGTAA
index2 GTAAGG
```

To use this to demultiplex two FASTQ files, where we assume that the barcode can be found in the header of the first file, we use the following command:

```
demultiplex demux barcodes.csv file_1.fq file_2.fq
```

This will generate six files:

```
file_1_index1.fq
file_2_index1.fq
file_1_index2.fq
file_2_index2.fq
```

(continues on next page)

(continued from previous page)

```
file_1_UNKNOWN.fq
file_2_UNKNOWN.fq
```

the first four files will contain records assigned to index1 and index2, the last two will contain anything that could not be assigned.

If the list of barcodes is not known beforehand, the `guess` subcommand can be used to search for a top list of barcodes. For example, if we want to search for the top five barcodes in the first 1000 records, we use the following:

```
demultiplex guess -o barcodes.csv -t 5 -n 1000 file.fq
```

This will generate the barcodes file that can be used for the `demux` subcommand.

If the number of barcodes is not known beforehand, an alternative selection method can be used which selects all barcodes with a minimum number of occurrences. The following command will generate a barcode file of all barcodes that occur at least five times in the first 1000 reads:

```
demultiplex guess -o barcodes.csv -f -t 5 -n 1000 file.fq
```

2.2 Other files

For platforms other than Illumina, or for alternative sequencing runs, like those coming from 10X experiments, barcodes sometimes end up at a specific location in each read. It can also be that the barcode is in the header, but only part of this barcode is used, this happens when dual indexing is used for example. To deal with these cases, both the `guess` as well as the `demux` subcommand can be instructed to look voor the barcode in the read with the `-r` option and a selection can be made by providing a start- and end coordinate via the `-s` and `-e` options. For example, if we want to search for barcodes in the first six nucleotides of a read, we use the following command:

```
demultiplex demux -r -e 6 barcodes.csv file.fq
```

2.3 Multiple barcodes

Suppose we have two files containing barcodes that are used for dual indexing:

- `A.csv` for barcode 1 and 2.
- `B.csv` for barcode 3 and 4.

Furthermore, suppose that the first barcode can be found in the header of read 1 and the second one in the header of read 2.

We can then demultiplex in two steps:

```
demultiplex demux A.csv read_1.fq read_2.fq
```

This will result in two new pairs of files:

- `read_1_1.fq`, `read_2_1.fq`
- `read_1_2.fq`, `read_2_2.fq`

We can now demultiplex each of these pairs as follows:

```
demultiplex demux B.csv read_2_1.fq read_1_1.fq
demultiplex demux B.csv read_2_2.fq read_1_2.fq
```

Which will result in the final list of pairs:

- `read_1_1_3.fq`, `read_2_1_3.fq`
- `read_1_1_4.fq`, `read_2_1_4.fq`
- `read_1_2_3.fq`, `read_2_2_3.fq`
- `read_1_2_4.fq`, `read_2_2_4.fq`

2.4 Barcodes at unknown locations

Libraries for long read platforms like PacBio and Oxford Nanopore often have barcodes in the reads at unknown locations. To demultiplex these kind of datasets, it is necessary to align the barcode to the read to find it. This is exactly what the `match` subcommand does.

The barcodes file is very similar to the one in the `demux` command, except that it allows for multiple barcodes per sample. If dual barcoding is used, the formatting is as follows.

```
name sequence1 sequence2
```

So a typical barcodes file might look like this:

```
index1 ACGTAA TTGCAA
index2 GTAAGG GTGTAA
```

Demultiplexing is done as follows.

```
demultiplex match barcodes.csv reads.fq
```

CHAPTER 3

Library

Warning: Under construction.

CHAPTER 4

Contributors

- Jeroen F.J. Laros <J.F.J.Laros@lumc.nl> (Original author, maintainer)

Find out who contributed:

```
git shortlog -s -e
```