axe Documentation

Release 0.3.3-dirty

Kevin Murray

Apr 16, 2018

Contents

Axe is a read de-multiplexer, useful in situations where sequence reads contain the indexes that uniquely distinguish samples. Axe uses a rapid and accurate algorithm based on hamming mismatch tries to competitively match the prefix of a sequencing read against a set of indexes. Axe supports combinatorial indexing schemes.

Contents:

CHAPTER 1

Axe Tutorial

In this tutorial, we'll use Axe to demultiplex some paired-end, combinatorially-index Genotyping-by-Sequencing reads. The data for this tutorial is available from figshare: https://figshare.com/articles/axe-tutorial_tar/6143720 .

Axe should be run as the initial step of any analysis: don't use sequence QC tools like AdapterRemoval or Trimmomatic before using axe, as indexes may be trimmed away, or pairing information removed.

1.1 Step 0: Download the trial data

This will download the trial data, and extract it on the fly:

```
curl -LS https://ndownloader.figshare.com/files/11094782 | tar xv
```

1.2 Step 1: prepare a key file

The key file associates index sequences with sample names. A key file can be prepared in a spreadsheet editor, like LibreOffice Calc, or Excel. The format is quite strict, and is described in detail in the online usage documentation.

Let's now inspect the keyfile I have provided for the tutorial.

```
head axe-keyfile.tsv
```

1.3 Step 2: Demultiplex with Axe

In this step, we will demultiplex our interleaved input file to per-sample interleaved output files. To see a full range of Axe's options, please run `axe-demux -h`, or inspect the online usage documentation.

First, let's inspect the input.

```
zcat axe-tutorial.fastq.gz | head -n 8
```

Then, we need to ensure that axe has somewhere to put the demultiplexed reads. Axe outputs one file (or more, depending on pairing) per sample. Axe does so by appending the sample name to some prefix (as given by the `-I`, `-F`, and/or `-R` options). If this prefix is a directory, then sample fastq files will be created in that sub-directory, but the directory must exist. Let's make an output directory:

```
mkdir -p output
```

Now, let's demultiplex the reads!

```
axe-demux -i axe-tutorial.fastq.gz -I output/ \
   -c -b axe-keyfile.tsv -t demux-stats.tsv -z 1
```

The command above demultiplexes reads from `axe-tutorial.fastq.gz` into separate files under `output`, based on the combinatorial (`-c`) sample-to-index-sequence mapping described in `axe-keyfile.tsv`, and saves a file of statistics as `demux-stats.tsv`. Note that we have enabled compression of output files using the `-z` option, in case you don't have much disk space available. This will make Axe slightly slower.

CHAPTER 2

Axe Usage

Note: For arcane reasons, the name of the `axe` binary changed to `axe-demux` with version 0.3.0. Apologies for the inconvenience, this was required to make `axe` installable in Debian and its derivatives. Command-line usage did not change.

Axe has several usage modes. The primary distinction is between the two alternate indexing schemes, single and combinatorial indexing. Single index matching is used when only the first read contains index sequences. Combinatorial indexing is used when both reads in a read pair contain independent (typically different) index sequences.

For concise reference, the command-line usage of `axe-demux` is reproduced below:

```
USAGE:
axe-demux [-mzc2pt] -b (-f [-r] | -i) (-F [-R] | -I)
axe-demux -h
axe-demux -v

OPTIONS:
    -m, --mismatch        Maximum hamming distance mismatch. [int, default 1]
    -z, --ziplevel        Gzip compression level, or 0 for plain text [int, default 0]
    -c, --combinatorial        Use combinatorial barcode matching. [flag, default OFF]
    -p, --permissive        Don't error on barcode mismatch confict, matching only
                            exactly for conficting barcodes. [flag, default OFF]
    -2, --trim-r2        Trim barcode from R2 read as well as R1. [flag, default OFF]
    -b, --barcodes        Barcode file. See --help for example. [file]
    -f, --fwd-in        Input forward read. [file]
    -F, --fwd-out        Output forward read prefix. [file]
    -r, --rev-in        Input reverse read. [file]
    -R, --rev-out        Output reverse read prefix. [file]
    -i, --ilfq-in        Input interleaved paired reads. [file]
    -I, --ilfq-out        Output interleaved paired reads prefix. [file]
    -t, --table-file        Output a summary table of demultiplexing statistics to␣
↪file. [file]
    -h, --help                Print this usage plus additional help.
    -V, --version        Print version string.
```

```
  -v, --verbose         Be more verbose. Additive, -vv is more vebose than -v.
  -q, --quiet                 Be very quiet.
```

2.1 Inputs and Outputs

Regardless of read mode, three input and output schemes are supported: single-end reads, paired reads (separate R1 and R2 files) and interleaved paired reads (one file, with R1 and R2 as consecutive reads). If single end reads are inputted, they must be output as single end reads. If either paired or interleaved paired reads are read, they can be output as either paired reads or interleaved paired reads. This applies to both successfully de-multiplexed reads and reads that could not be de-multiplexed.

The `-z` flag can be used to specify that outputs should be compressed using gzip compression. The `-z` flag takes an integer argument between 0 (the default) and 9, where 0 indicates plain text output (`gzopen` mode "wT"), and 1-9 indicate that the respective compression level should be used, where 1 is fastest and 9 is most compact.

The output flags should be prefixes that are used to generate the output file name based on the index's (or index pair's) ID. The names are generated as: `prefix` + _ + `index ID` + _ + `read number` + `.extension`. The output file for reads that could not be demultiplexed is `prefix` + _ + `unknown` + _ + `read number` + `.extension`. The read number is omitted unless the paired read file scheme is used, and is "il" for interleaved output. The extension is "fastq"; ".gz" is appended to the extension if the `-z` flag is used.

The corresponding CLI flags are:

- `-f` and `-F`: Single end or paired R1 file input and output respectively.
- `-r` and `-R`: Paired R2 file input and output.
- `-i` and `-I`: Interleaved paired input and output.

2.2 The index file

The index file is a tab-separated file with an optional header. It is mandatory, and is always supplied using the `-b` command line flag. The exact format is dependent on indexing mode, and is described further in the sections below. If a header is present, the header line must start with either *Barcode* or `index`, or it will be interpreted as a index line, leading to a parsing error. Any line starting with ';' or '#' is ignored, allowing comments to be added in line with indexes. Please ensure that the software used to produce the index uses ASCII encoding, and does not insert a Byte-order Mark (BoM) as many text editors can silently use Unicode-based encoding schemes. I recommend the use of LibreOffice Calc (part of a free and open source office suite) to generate index tables; Microsoft Excel can also be used.

2.3 Mismatch level selection

Independent of index mode, the `-m` flag is used to select the maximum allowable hamming distance between a read's prefix and a index to be considered as a match. As "mutated" indexes must be unique, a hamming distance of one is the default as typically indexes are designed to differ by a hamming distance of at least two. Optionally, (using the `-p` flag), axe will allow selective mismatch levels, where, if clashes are observed, the index will only be matched exactly. This allows one to process datasets with indexes that don't have a sufficiently high distance between them.

2.4 Single index mode

Single index mode is the default mode of operation. Barcodes are matched against read one (hereafter the forward read), and the index is trimmed from only the forward read, unless the `-2` command line flag is given, in which case a prefix the same length as the matched index is also trimmed from the second or reverse read. Note that sequence of this second read is not checked before trimming.

In single index mode, the index file has two columns: `Barcode` and `ID`.

2.5 Combinatorial index mode

Combinatorial index mode is activated by giving the `-c` flag on the command line. Forward read indexes are matched against the forward read, and reverse read indexes are matched against the reverse read. The optimal indexes are selected independently, and the index pair is selected from these two indexes. The respective indexes are trimmed from both reads; the `-2` command line flag has no effect in combinatorial index mode.

In combinatorial index mode, the index file has three columns: `Barcode1`, `Barcode2` and `ID`. Individual indexes can occur many times within the forward and reverse indexes, but index pairs must be unique combinations.

2.6 The Demultiplexing Statistics File

The `-t` option allows the output of per-sample read counts to a tab-separated file. The file will have a header describing its format, and includes a line for reads which could not be demultiplexed.

CHAPTER 3

Axe's matching algorithm

Axe uses an algorithm based on longest-prefix-in-trie matching to match a variable length from the start of each read against a set of 'mutated' indexes.

3.1 Hamming distance matching

While for most applications in high-throughput sequencing hamming distances are a frowned-upon metric, it is typical for HTS read indexes to be designed to tolerate a certain level of hamming mismatches. Given these sequences are short and typically occur at the 5' end of reads, insertions and deletions rarely need be considered, and the increased rate of assignment of reads with many errors is offset by the risk of falsely assigning indexes to an incorrect sample. In any case, reads with more than 1-2 sequencing errors in their first several bases are likely to be poor quality, and will simply be filtered out during downstream quality control.

3.2 Hamming mismatch tries

Typically, reads are matched to a set of indexes by calculating the hamming distance between the index, and the first l bases of a read for a index of length l. The "correct" index is then selected by recording either the index with the lowest hamming distance to the read (competitive matching) or by simply accepting the first index with a hamming distance below a certain threshold. These approaches are both very computationally expensive, and can have lower accuracy than the algorithm I propose. Additionally, implementations of these methods rarely handle indexes of differing length and combinatorial indexing well, if at all.

Central to Axe's algorithm is the concept of hamming-mismatch tries. A trie is a N-ary tree for an N letter alphabet. In the case of high-throughput sequencing reads, we have the alphabet `AGCT`, corresponding to the four nucleotides of DNA, plus `N`, used to represent ambiguous base calls. Instead of matching each index to each read, we pre-calculate all allowable sequences at each mismatch level, and store these in level-wise tries. For example, to match to a hamming distance of 2, we create three tries: One containing all indexes, verbatim, and two tries where every sequence within a hamming distance of 1 and 2 of each index respectively. Hereafter, these tries are referred to as the 0, 1 and 2-mm tries, for a hamming distance (mismatch) of 0, 1 and 2. Then, we find the longest prefix in each sequence read in the 0mm trie. If this prefix is not a valid leaf in the 0mm trie, we find the longest prefix in the 1mm trie, and so on

for all tries in ascending order. If no prefix of the read is a complete sequence in any trie, the read is assigned to an "non-indexed" output file.

This algorithm ensures optimal index matching in many ways, but is also extremely fast. In situations with indexes of differing length, we ensure that the *longest* acceptable index at a given hamming distance is chosen; assuming that sequence is random after the index, the probability of false assignments using this method is low. We also ensure that short perfect matches are preferred to longer inexact matches, as we firstly only consider indexes with no error, then 1 error, and so on. This ensures that reads with indexes that are followed by random sequence that happens to inexactly match a longer index in the set are not falsely assigned to this longer index.

The speed of this algorithm is largely due to the constant time matching algorithm with respect to the number of indexes to match. The time taken to match each read is proportional instead to the length of the indexes, as for a index of length l, at most $l + 1$ trie level descents are required to find an entry in the trie. As this length is more-or-less constant and small, the overall complexity of axe's algorithm is $O(n)$ for n reads, as opposed to $O(nm)$ for n reads and m indexes as is typical for traditional matching algorithms

CHAPTER 4

Indices and tables

- genindex