axe Documentation

Release 0.0.1

Kevin Murray

October 18, 2015

Contents

Axe is a read de-multiplexer, useful in situations where sequence reads contain the barcodes 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 barcodes. Axe supports combinatorial barcoding schemes.

Contents:

CHAPTER 1

Axe Usage

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

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

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

1.1 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 barcode's (or barcode pair's) ID. The names are generated as: `prefix` + _ + `barcode 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.

1.2 The barcode file

The barcode 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 barcoding mode, and is described further in the sections below. If a header is present, the header line must start with either *Barcode* or `barcode`, or it will be interpreted as a barcode line, leading to a parsing error. Any line starting with ';' or '#' is ignored, allowing comments to be added in line with barcodes. Please ensure that the software used to produce the barcode 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 barcode tables; Microsoft Excel can also be used.

1.3 Mismatch level selection

Independent of barcode mode, the `-m` flag is used to select the maximum allowable hamming distance between a read's prefix and a barcode to be considered as a match. As "mutated" barcodes must be unique, a hamming distance of one is the default as typically barcodes 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 barcode will only be matched exactly. This allows one to process datasets with barcodes that don't have a sufficiently high distance between them.

1.4 Single barcode mode

Single barcode mode is the default mode of operation. Barcodes are matched against read one (hereafter the forward read), and the barcode 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 barcode is also trimmed from the second or reverse read. Note that sequence of this second read is not checked before trimming.

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

1.5 Combinatorial barcode mode

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

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

1.6 The Demultipexing 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 unbarcoded reads.

CHAPTER 2

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' barcodes.

2.1 Hamming distance matching

While for most applications in high-throughput sequencing hamming distances are a frowned-upon metric, it is typical for HTS read barcodes 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 barcodes 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.

2.2 Hamming mismatch tries

Typically, reads are matched to a set of barcodes by calculating the hamming distance between the barcode, and the first l bases of a read for a barcode of length l. The "correct" barcode is then selected by recording either the barcode with the lowest hamming distance to the read (competitive matching) or by simply accepting the first barcode 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 barcodes of differing length and combinatorial barcoding 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 barcode 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 barcodes, verbatim, and two tries where every sequence within a hamming distance of 1 and 2 of each barcode 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-barcoded" output file.

This algorithm ensures optimal barcode matching in many ways, but is also extremely fast. In situations with barcodes of differing length, we ensure that the *longest* acceptable barcode at a given hamming distance is chosen; assuming that sequence is random after the barcode, 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 barcodes with no error, then 1 error, and so on. This ensures that reads with barcodes that are followed by random sequence that happens to inexactly match a longer barcode in the set are not falsely assigned to this longer barcode.

The speed of this algorithm is largely due to the constant time matching algorithm with respect to the number of barcodes to match. The time taken to match each read is proportional instead to the length of the barcodes, as for a barcode 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 barcodes as is typical for traditional matching algorithms

CHAPTER 3

Indices and tables

- genindex