atropos Documentation

Release 1.1.5+6.g1efc0a8.dirty

Marcel Martin

May 22, 2017

Contents

CHAPTER 1

Installation

Quickstart

The easiest way to install atropos is to use `pip` on the command line:

```
pip install --user --upgrade atropos
```

This will download the software from PyPI (the Python packaging index), and install the atropos binary into `$HOME/.local/bin`. If an old version of atropos exists on your system, the `--upgrade` parameter is required in order to install a newer version. You can then run the program like this:

```
~/.local/bin/atropos --help
```

If you want to avoid typing the full path, add the directory `$HOME/.local/bin` to your `$PATH` environment variable.

Conda

Atropos can also be installed using conda:

```
conda install atropos
```

Docker

We provide a Docker container that can be executed using a Docker or Singularity engine:

    docker run jdidion/atropos <args>

or

    singularity run docker://jdidion/atropos

Manual installation

Dependencies

Atropos requires this software to be installed:

- One of Python 3.3+.
- A C compiler.
- Cython 0.25.2+

Under Ubuntu, you may need to install the packages `build-essential` and `python-dev`.

Installation

If you have already downloaded and unpacked the `.tar.gz` file, then installation is done like this:

```
python3 setup.py install --user
```

If you get an error message:

```
error: command 'gcc' failed with exit status 1
```

Then check the entire error message. If it says something about a missing `Python.h` file, then you need to install the Python development packages. The appropriate package is called `python3-dev` in Ubuntu.sour

We also provide a Makefile that can optionally run unit tests for you. For this you will need to have make installed and also the Python pytest library (if you want to run the tests). To both test and install, run:

```
make installargs='--user'
```

To only install, run:

```
make install installargs='--user'
```

System-wide installation

If you have root access, then you can install atropos system-wide by running:

```
sudo pip install atropos
```

This installs atropos into */usr/local/bin*.

If you want to upgrade from an older version, use this command instead:

```
sudo pip install --upgrade atropos
```

To use the Makefile, simply run:

```
make
```

or (to skip running tests)

    make install

Use without installation

Build the C extension module (you can try to skip this step – a compiled version of the module for Linux x86_64 is already included):

```
python setup.py build_ext -i
```

Then simply run the script from where it is, similar to this:

```
bin/atropos --help
```

If you get any errors, first try to explicitly request a specific Python version by running atropos like this:

```
python3 bin/atropos --help
```

CHAPTER 2

User guide

Basic usage

If you just want to trim a 3' adapter, the basic command-line for atropos is:

```
atropos -a AACCGGTT -o output.fastq -se input.fastq
```

The sequence of the adapter is given with the `-a` option. Of course, you need to replace `AACCGGTT` with your actual adapter sequence. Reads are read from the input file `input.fastq` and written to the output file `output.fastq`.

Atropos searches for the adapter in all reads and removes it when it finds it. All reads that were present in the input file will also be present in the output file, some of them trimmed, some of them not. Even reads that were trimmed entirely (because the adapter was found in the very beginning) are output. All of this can be changed with command-line options, explained further down.

A report is printed after atropos has finished processing the reads.

Input and output file formats

Input files for atropos need to be in one the these formats:

- FASTA (file name extensions: `.fasta`, `.fa`, `.fna`, `.csfasta`, `.csfa`)
- FASTQ (extensions: `.fastq`, `.fq`)
- A pair of a FASTA file and a `.(cs)qual` file
- SAM/BAM (extensions: `.sam`, `.bam`)

The latter format is (or was) used for colorspace data from the SOLiD instruments.

The input file format is recognized from the file name extension (given in parentheses in the list above). You can also explicitly specify which format the input has by using the `--format` option.

Paired-end FASTQ files are supported using `-pe1` and `-pe2` to specify the read 1 and read 2 files, respectively. Interleaved files (FASTQ, SAM, and BAM) are supported using the `-l` option. Note that interleaved files must be sorted such that both reads appear successively for each read pair. For SAM/BAM, the two reads can be in any order since their identity is detected from the flags. You must have the `pysam` library installed

for SAM/BAM support. If you only want to process one of the read pairs from an interleaved file, you can set `--single-input-read <1|2>`.

The output file format is determined by the input format. By default, paired-end output will be written into two files, one for each read. Set the `-L` option to write inteleaved output instead. Also, atropos does not check the output file name: If you input FASTQ data, but use `-o output.fasta`, then the output file will actually be in FASTQ format.

Compressed files

Atropos supports compressed input and output files. Whether an input file needs to be decompressed or an output file needs to be compressed is detected automatically by inspecting the file name: If it ends in `.gz`, then gzip compression is assumed. You can therefore run atropos like this and it works as expected:

```
atropos -a AACCGGTT -o output.fastq.gz -se input.fastq.gz
```

All of atropos's options that expect a file name support this.

Files compressed with bzip2 (`.bz2`) or xz (`.xz`) are also supported.

Standard input and output

If no output file is specified via the `-o` option, then the output is sent to the standard output stream. Instead of the example command line from above, you can therefore also write:

```
atropos -a AACCGGTT -se input.fastq > output.fastq
```

There is one difference in behavior if you use atropos without `-o`: The report is sent to the standard error stream instead of standard output. You can redirect it to a file like this:

```
atropos -a AACCGGTT -se input.fastq > output.fastq 2> report.txt
```

Wherever Atropos expects a file name, you can also write a dash (`-`) in order to specify that standard input or output should be used. For example:

```
tail -n 4 input.fastq | atropos -a AACCGGTT -se - > output.fastq
```

The `tail -n 4` prints out only the last four lines of `input.fastq`, which are then piped into Atropos. Thus, Atropos will work only on the last read in the input file.

In most cases, you should probably use `-` at most once for an input file and at most once for an output file, in order not to get mixed output.

You cannot combine `-` and gzip compression since Atropos needs to know the file name of the output or input file. if you want to have a gzip-compressed output file, use `-o` with an explicit name.

One last "trick" is to use `/dev/null` as an output file name. This special file discards everything you send into it. If you only want to see the statistics output, for example, and do not care about the trimmed reads at all, you could use something like this:

```
atropos -a AACCGGTT -o /dev/null -se input.fastq
```

Subsampling

If you only want to processes some of the reads in your input file, you have two options. First, you can use the `--max-reads <N>` option to only process the first `N` reads/pairs in the input. Second, you can use the `--subsample <p>` option to sample a (pseudo)random fraction of the reads from the file, where the fraction is defined by probability `0 < p <= 1`. You can set a specific seed using `--subsample-seed` to guarantee identical subsampling between runs.

Read processing

Atropos can do a lot more in addition to removing adapters. There are various command-line options that make it possible to modify and filter reads and to redirect them to various output files. Each read is processed in the following way:

1. *Read modification options* are applied. This includes *adapter removal*, *quality trimming*, read name modifications etc.
2. *Filtering options* are applied, such as removal of too short or untrimmed reads. Some of the filters also allow to redirect a read to a separate output file.
3. If the read has passed all the filters, it is written to the output file.

Read modifications are applied in a specific order (below), and steps not requested on the command-line are skipped.

1. *Removing a fixed number of bases* with `-c` (C).
2. *NextSeq polyG trimming* with `--nextseq-trim` (G).
3. *Quality trimming* with `-q`/`-Q` (Q).
4. *Adapter removal* with `-a`/`-A`, `-b`/`-B`, and `-g`/`-G` (A).
5. *Bisulfite sequencing-specific trimming* with `--bisulfite`.
6. *N trimming* with `--trim-n`.
7. *Ensuring at least a fixed number of bases have been trimmed* with `-i`/`-I`.
8. *Length tag modification* with `--length-tag`.
9. *Read name suffix removal* with `--strip-suffix`.
10. *Addition of prefix and suffix to read name* with `-x`/`--prefix` and `-y`/`--suffix`.
11. Double-encode the sequence (only colorspace).
12. Replace negative quality values with zero (zero capping, only colorspace).
13. Trim primer base (only colorspace).

The user can have some control over the order in which operations are applied. The `--op-order` option takes a string of characters (in parentheses above) that controls the order in which the first four operations are applied. By default, `--op-order CGQA` to maintain compatibility with Cutadapt; however, this is likely to change to 'GACQ' in the near future.

Removing adapters

Atropos supports trimming of multiple types of adapters:

Adapter type	Command-line option
3' adapter	`-a ADAPTER`
5' adapter	`-g ADAPTER`
Anchored 3' adapter	`-a ADAPTER$`
Anchored 5' adapter	`-g ^ADAPTER`
5' or 3' (both possible)	`-b ADAPTER`
Linked adapter	`-a ADAPTER1...ADAPTER2`

Here is an illustration of the allowed adapter locations relative to the read and depending on the adapter type:

By default, all adapters *are searched error-tolerantly*. Adapter sequences *may also contain the "N" wildcard character*.

In addition, it is possible to *remove a fixed number of bases* from the beginning or end of each read, and to *remove low-quality bases (quality trimming)* from the 3' and 5' ends.

Alignment algorithms

Cutadapt was developed when single-end reads were common, and thus its alignment algorithm was optimized for that data. It uses a very high-performance Implementation of semi-global alignment to identify adapters within reads. However, most current data sets are paired-end, which enables much better adapter trimming. With most paired-end read data, adapter contamination should be symmetric, because the sequencer reads the same number of bases in either direction. So, for example, if you have 150 bp paired-end reads and you have a read pair with an insert size of 130, the sequencer will read the 130 bp from the forward direction and then read an additional 20 bp of the forward adapter, and will then read the same 130 bp in the reverse direction followed by 20 bp of the reverse adapter. This means that adapter contamination can be more accurately detected by first aligning the reads to each other and then examining the overhangs for adapter sequences. This procedure is called insert alignment, as opposed to adapter alignment. Atropos implements a version of the algorithm described in Strum et al. (DOI: 10.1186/s12859-016-1069-7) that first attempts insert alignment (leveraging the dynamic programming model that was implemented in Cutadapt). If the insert match is successful, then a less stringent adapter match is performed. Otherwise, the normal Cutadapt-style adapter matching is performed.

This new algorithm only works with paired-end data that contains a single 3' adapter in each direction. To enable this aligner, use the *–aligner insert* option.

> atropos –aligner insert -a AGATCGGAAGAGCACACGTCTGAACTCCAGTCACACAGTGATCTCGTATGCCGTCTTCTGCTTG -A AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGTAGATCTCGGTGGTCGCCGTATCATT -o trimmed.1.fq.gz -p trimmed.2.fq.gz -pe1 reads.1.fq.gz -pe2 reads.2.fq.gz

There are three parameters that can be used to fine-tune insert matching:

- `--insert-match-error-rate`: Similar to `-e/--error-rate`, but specifically

for matching inserts. We find generally that it is safe and leads to greater accuracy when the insert-match error rate is set higher than the global error rate. For example `-e 0.1 --insert-match-error-rate 0.2` works well for high-quality data. * `--insert-match-adapter-error-rate`: Maximum allowed error rate for matching adapters after successful insert match. This is typically used when you want less stringent adapter matching when there is already evidence of an insert match. The global error rate (`-e`) is still used when the algorithm is not able to make an insert match and falls back to adapter matching. * `--insert-max-rmp`: Random match probability (RMP) is the probability that two different sequences of length N will match each other by chance. You can specify an RMP threshold in addition to/instead of error rate and minimum overlap.

Error correction

Read pairs that overlap are derived from the same sequence, and are thus expected to be identical. However, all sequencing technologies are associated with some level of error that may lead to mismatches. The insert aligner is able to correct mismatches between overlapping reads using one of three strategies (set using the `--correct-mismatches` option):

- conservative: When the qualities of the bases are different, set the base to be the one with the highest quality/lowest error probability. When the qualities are equal, do not change them.

- liberal: Same policy as 'conservative', except that when qualities are the same set the base to be the one from the read with the highest median quality.
- N: Set the base to 'N' in both pairs.

Note that in all cases where error correction is enabled, if exactly one base is an N, it is set to be the base from the other pair.

3' adapters

A 3' adapter is a piece of DNA ligated to the 3' end of the DNA fragment you are interested in. The sequencer starts the sequencing process at the 5' end of the fragment and sequences into the adapter if the read is long enough. The read that it outputs will then have a part of the adapter in the end. Or, if the adapter was short and the read length quite long, then the adapter will be somewhere within the read (followed by other bases).

For example, assume your fragment of interest is *MYSEQUENCE* and the adapter is *ADAPTER*. Depending on the read length, you will get reads that look like this:

```
MYSEQUEN
MYSEQUENCEADAP
MYSEQUENCEADAPTER
MYSEQUENCEADAPTERSOMETHINGELSE
```

Use atropos' `-a ADAPTER` option to remove this type of adapter. This will be the result:

```
MYSEQUEN
MYSEQUENCE
MYSEQUENCE
MYSEQUENCE
```

As can be seen, atropos correctly deals with partial adapter matches, and also with any trailing sequences after the adapter. Atropos deals with 3' adapters by removing the adapter itself and any sequence that may follow. If the sequence starts with an adapter, like this:

```
ADAPTERSOMETHING
```

Then the sequence will be empty after trimming. By default, empty reads are kept and will appear in the output.

5' adapters

Note: Unless your adapter may also occur in a degraded form, you probably want to use an anchored 5' adapter, described in the next section.

A 5' adapter is a piece of DNA ligated to the 5' end of the DNA fragment of interest. The adapter sequence is expected to appear at the start of the read, but may be partially degraded. The sequence may also appear somewhere within the read. In all cases, the adapter itself and the sequence preceding it is removed.

Again, assume your fragment of interest is *MYSEQUENCE* and the adapter is *ADAPTER*. The reads may look like this:

```
ADAPTERMYSEQUENCE
DAPTERMYSEQUENCE
TERMYSEQUENCE
SOMETHINGADAPTERMYSEQUENCE
```

All the above sequences are trimmed to `MYSEQUENCE` when you use *-g ADAPTER*. As with 3' adapters, the resulting read may have a length of zero when the sequence ends with the adapter. For example, the read

```
SOMETHINGADAPTER
```

will be empty after trimming.

Anchored 5' adapters

In many cases, the above behavior is not really what you want for trimming 5' adapters. You may know, for example, that degradation does not occur and that the adapter is also not expected to be within the read. Thus, you always expect the read to look like the first example from above:

```
ADAPTERSOMETHING
```

If you want to trim only this type of adapter, use `-g ^ADAPTER`. The `^` is supposed to indicate the the adapter is "anchored" at the beginning of the read. In other words: The adapter is expected to be a prefix of the read. Note that cases like these are also recognized:

```
ADAPTER
ADAPT
ADA
```

The read will simply be empty after trimming.

Be aware that Atropos still searches for adapters error-tolerantly and, in particular, allows insertions. So if your maximum error rate is sufficiently high, even this read will be trimmed:

```
BADAPTERSOMETHING
```

The `B` in the beginnig is seen as an insertion. If you also want to prevent this from happening, use the option `--no-indels` to disallow insertions and deletions entirely.

Anchored 3' adapters

It is also possible to anchor 3' adapters to the end of the read. This is rarely necessary, but if you have merged, for example, overlapping paired-end reads, then it is useful. Add the `$` character to the end of an adapter sequence specified via `-a` in order to anchor the adapter to the end of the read, such as `-a ADAPTER$`. The adapter will only be found if it is a *suffix* of the read, but errors are still allowed as for 5' adapters. You can disable insertions and deletions with `--no-indels`.

Anchored 3' adapters work as if you had reversed the sequence and used an appropriate anchored 5' adapter.

As an example, assume you have these reads:

```
MYSEQUENCEADAP
MYSEQUENCEADAPTER
MYSEQUENCEADAPTERSOMETHINGELSE
```

Using `-a ADAPTER$` will result in:

```
MYSEQUENCEADAP
MYSEQUENCE
MYSEQUENCEADAPTERSOMETHINGELSE
```

Only the middle read is trimmed at all.

Linked adapters

This is a combination of a 5' and a 3' adapter. Use `-a ADAPTER1...ADAPTER2` to search for a linked adapter. ADAPTER1 is interpreted as an anchored 5' adapter, which is searched for first. Only if ADAPTER1 is found will then ADAPTER2 be searched for, which is a regular 3' adapter.

This feature is experimental and will probably break when used in combination with some other options, such as `--info-file`, `--mask-adapter`.

5' or 3' adapters

The last type of adapter is a combination of the 5' and 3' adapter. You can use it when your adapter is ligated to the 5' end for some reads and to the 3' end in other reads. This probably does not happen very often, and this adapter type was in fact originally implemented because the library preparation in an experiment did not work as it was supposed to.

For this type of adapter, the sequence is specified with `-b ADAPTER` (or use the longer spelling `--anywhere ADAPTER`). The adapter may appear in the beginning (even degraded), within the read, or at the end of the read (even partially). The decision which part of the read to remove is made as follows: If there is at least one base before the found adapter, then the adapter is considered to be a 3' adapter and the adapter itself and everything following it is removed. Otherwise, the adapter is considered to be a 5' adapter and it is removed from the read, but the sequence after it remains.

Here are some examples.

Read before trimming	Read after trimming	Detected adapter type
MYSEQUENCEADAPTERSOMETHING	MYSEQUENCE	3' adapter
MYSEQUENCEADAPTER	MYSEQUENCE	3' adapter
MYSEQUENCEADAP	MYSEQUENCE	3' adapter
MADAPTER	M	3' adapter
ADAPTERMYSEQUENCE	MYSEQUENCE	5' adapter
PTERMYSEQUENCE	MYSEQUENCE	5' adapter
TERMYSEQUENCE	MYSEQUENCE	5' adapter

The `-b` option cannot be used with colorspace data.

Error tolerance

All searches for adapter sequences are error tolerant. Allowed errors are mismatches, insertions and deletions. For example, if you search for the adapter sequence `ADAPTER` and the error tolerance is set appropriately (as explained below), then also `ADABTER` will be found (with 1 mismatch), as well as `ADAPTR` (with 1 deletion), and also `ADAPPTER` (with 1 insertion).

The level of error tolerance is adjusted by specifying a *maximum error rate*, which is 0.1 (=10%) by default. Use the `-e` option to set a different value. To determine the number of allowed errors, the maximum error rate is multiplied by the length of the match (and then rounded off).

What does that mean? Assume you have a long adapter `LONGADAPTER` and it appears in full somewhere within the read. The length of the match is 11 characters since the full adapter has a length of 11, therefore 11·0.1=1.1 errors are allowed with the default maximum error rate of 0.1. This is rounded off to 1 allowed error. So the adapter will be found within this read:

```
SEQUENCELONGADUPTERSOMETHING
```

If the match is a bit shorter, however, the result is different:

```
SEQUENCELONGADUPT
```

Only 9 characters of the adapter match: `LONGADAPT` matches `LONGADUPT` with one substitution. Therefore, only 9·0.1=0.9 errors are allowed. Since this is rounded off to zero allowed errors, the adapter will not be found.

The number of errors allowed for a given adapter match length is also shown in the report that Atropos prints:

```
Sequence: 'LONGADAPTER'; Length: 11; Trimmed: 2 times.

No. of allowed errors:
0-9 bp: 0; 10-11 bp: 1
```

This tells us what we now already know: For match lengths of 0-9 bases, zero errors are allowed and for matches of length 10-11 bases, one error is allowed.

The reason for this behavior is to ensure that short matches are not favored unfairly. For example, assume the adapter has 40 bases and the maximum error rate is 0.1, which means that four errors are allowed for full-length matches. If four errors were allowed even for a short match such as one with 10 bases, this would mean that the error rate for such a case is 40%, which is clearly not what was desired.

Insertions and deletions can be disallowed by using the option `--no-indels`.

Multiple adapter occurrences within a single read

If a single read contains multiple copies of the same adapter, the basic rule is that the leftmost match is used for both 5' and 3' adapters. For example, when searching for a 3' adapter in

```
cccccADAPTERgggggADAPTERttttt
```

the read will be trimmed to

```
ccccc
```

When the adapter is a 5' adapter instead, the read will be trimmed to

```
gggggADAPTERttttt
```

The above applies when both occurrences of the adapter are *exact* matches, and it also applies when both occurrences of the adapter are *inexact* matches (that is, it has at least one indel or mismatch). However, if one match is exact, but the other is inexact, then the exact match wins, even if it is not the leftmost one! The reason for this behavior is that Atropos searches for exact matches first and, to improve performance, skips the error-tolerant matching step if an exact match was found.

Reducing random matches

Since Atropos allows partial matches between the read and the adapter sequence, short matches can occur by chance, leading to erroneously trimmed bases. For example, roughly 25% of all reads end with a base that is identical to the first base of the adapter. To reduce the number of falsely trimmed bases, Atropos uses a threshold based on random-match probability (RMP). The default threshold is 1x10^-6, but you can change this with the `--adapter-max-rmp` option.

Another way you can control random matches is by specifying a minimum overlap between the adapter and the read. The minimum overlap length can be changed with the parameter `--overlap` (or its short version `-O`). Shorter matches are simply ignored, and the bases are not trimmed. Note that we generally find the RMP-based control to be sufficient, and thus setting a minimum overlap is usually not necessary.

Requiring at least three bases to match is quite conservative. Even if no minimum overlap was required, we can compute that we lose only about 0.44 bases per read on average, see Section 2.3.3 in my thesis. With the default minimum overlap length of 3, only about 0.07 bases are lost per read.

When choosing an appropriate minimum overlap length, take into account that true adapter matches are also lost when the overlap length is higher than zero, reducing Atropos' sensitivity.

Wildcards

All IUPAC nucleotide codes (wildcard characters) are supported. For example, use an `N` in the adapter sequence to match any nucleotide in the read, or use `-a YACGT` for an adapter that matches both `CACGT` and `TACGT`. The wildcard character `N` is useful for trimming adapters with an embedded variable barcode:

```
atropos -a ACGTAANNNNTTAGC -o output.fastq -se input.fastq
```

Wildcard characters in the adapter are enabled by default. Use the option `-N` to disable this.

Matching of wildcards in the reads is also possible, but disabled by default in order to avoid matches in reads that consist of many (often low-quality) `N` bases. Use `--match-read-wildcards` to enable wildcards also in reads.

If wildcards are disabled entirely (that is, you use `-N` and *do not* use `--match-read-wildcards`), then Atropos compares characters by ASCII value. Thus, both the read and adapter can be arbitrary strings (such as `SEQUENCE` or `ADAPTER` as used here in the examples).

Wildcards do not work in colorspace.

Repeated bases in the adapter sequence

If you have many repeated bases in the adapter sequence, such as many N``s or many ``A``s, you do not have to spell them out. For example, instead of writing ten ``A in a row (AAAAAAAAAA), write `A{10}` instead. The number within the curly braces specifies how often the character that preceeds it will be repeated. This works also for IUPAC wildcard characters, as in `N{5}`.

It is recommended that you use quotation marks around your adapter sequence if you use this feature. For poly-A trimming, for example, you would write:

```
atropos -a "A{100}" -o output.fastq -se input.fastq
```

Specifying adapters by name

Rather than enter your adapter sequences each time or maintain separate adapter files for each of your datasets, you can maintain a single adapter file (in FASTA format) and specify adapters by name:

```
atropos -F myadapters.fa -a TruSeq1 -A TruSeq2 ...
```

If you are using standard adapter sequences, it is likely that they are already included in the Atropos adapter definition file, in which case you can omit the `-F` option and Atropos will automatically fetch the file from the GitHub repository (assuming you have an internet connection). Atropos will cache the adapters found in either the default or the user-specified adapter file (in a .adapters file, unless you specify a different path using the `--adapter-cache-file` option). You can disable the fetching and use of the default adapter file with the `--no-default-adapters` option. You can disable adapter caching using the `--no-cache-adapters` option.

Modifying reads

This section describes in which ways reads can be modified other than adapter removal.

Removing a fixed number of bases

By using the `--cut` option or its abbreviation `-u`, it is possible to unconditionally remove bases from the beginning or end of each read. If the given length is positive, the bases are removed from the beginning of each read. If it is negative, the bases are removed from the end.

For example, to remove the first five bases of each read:

```
atropos -u 5 -o trimmed.fastq -se reads.fastq
```

To remove the last seven bases of each read:

```
atropos -u -7 -o trimmed.fastq -se reads.fastq
```

The `-u`/`--cut` option can be combined with the other options, but the desired bases are removed *before* any adapter trimming.

The `--cut-min` option (`-i`) works identically to the `--cut` option, except that bases are removed after all other modifications have been applied, and only if the required number of bases have not already been removed. For example, if the following sequence is in reads.fastq:

```
ACGTACGTACGTADAP
```

The following command will first trim the `ADAP` part of the adapter off the end. Then, since only 4 bases were trimmed, the `-i 5` option will cause a 5th base to be removed.

```
atropos -A ADAPTER -i 5 -o trimmed.fastq -se reads.fastq
```

Quality trimming

The `-q` (or `--trim-qualities`) parameter can be used to trim low-quality ends from reads before adapter removal. For this to work correctly, the quality values must be encoded as ascii(phred quality + 33). If they are encoded as ascii(phred quality + 64), you need to add `--quality-base=64` to the command line.

Quality trimming can be done without adapter trimming, so this will work:

```
atropos -q 10 -o output.fastq -se input.fastq
```

By default, only the 3' end of each read is quality-trimmed. If you want to trim the 5' end as well, use the `-q` option with two comma-separated cutoffs:

```
atropos -q 15,10 -o output.fastq -se input.fastq
```

The 5' end will then be trimmed with a cutoff of 15, and the 3' will be trimmed with a cutoff of 10. If you only want to trim the 5' end, then use a cutoff of 0 for the 3' end, as in `-q 10,0`.

Quality trimming algorithm

The trimming algorithm is the same as the one used by BWA, but applied to both ends of the read in turn (if requested). That is: Subtract the given cutoff from all qualities; compute partial sums from all indices to the end of the sequence; cut the sequence at the index at which the sum is minimal. If both ends are to be trimmed, repeat this for the other end.

The basic idea is to remove all bases starting from the end of the read whose quality is smaller than the given threshold. This is refined a bit by allowing some good-quality bases among the bad-quality ones. In the following example, we assume that the 3' end is to be quality-trimmed.

Assume you use a threshold of 10 and have these quality values:

42, 40, 26, 27, 8, 7, 11, 4, 2, 3

Subtracting the threshold gives:

32, 30, 16, 17, -2, -3, 1, -6, -8, -7

Then sum up the numbers, starting from the end (partial sums). Stop early if the sum is greater than zero:

(70), (38), 8, -8, -25, -23, -20, -21, -15, -7

The numbers in parentheses are not computed (because 8 is greater than zero), but shown here for completeness. The position of the minimum (-25) is used as the trimming position. Therefore, the read is trimmed to the first four bases, which have quality values 42, 40, 26, 27.

Modifying read names

If you feel the need to modify the names of processed reads, some of the following options may be useful.

Use `-y` or `--suffix` to append a text to read names. The given string can contain the placeholder `{name}`, which will be replaced with the name of the adapter found in that read. For example, writing

```
atropos -a adapter1=ACGT -y ' we found {name}' -se input.fastq
```

changes a read named `read1` to `read1 we found adapter1` if the adapter `ACGT` was found. The options `-x`/`--prefix` work the same, but the text is added in front of the read name. For both options, spaces need to be specified explicitly, as in the above example. If no adapter was found in a read, the text `no_adapter` is inserted for `{name}`.

In order to remove a suffix of each read name, use `--strip-suffix`.

Some old 454 read files contain the length of the read in the name:

```
>read1 length=17
ACGTACGTACAAAAAA
```

If you want to update this to the correct length after trimming, use the option `--length-tag`. In this example, this would be `--length-tag 'length='`. After trimming, the read would perhaps look like this:

```
>read1 length=10
ACGTACGTAC
```

NextSeq-specific trimming

Some data from the new Illumina NextSeq platform generates base calls that have high quality scores but are incorrect due to the use of only two fluorescent tags (rather than the 4 used in the MiSeq and HiSeq sequencers). These base calls appear as a polyG string at the end of the read. The `--nextseq-trim` option will remove these bases.

Filtering reads

By default, all processed reads, no matter whether they were trimmed are not, are written to the output file specified by the `-o` option (or to standard output if `-o` was not provided). For paired-end reads, the second read in a pair is always written to the file specified by the `-p` option.

The options described here make it possible to filter reads by either discarding them entirely or by redirecting them to other files. When redirecting reads, the basic rule is that *each read is written to at most one file*. You cannot write reads to more than one output file.

In the following, the term "processed read" refers to a read to which all modifications have been applied (adapter removal, quality trimming etc.). A processed read can be identical to the input read if no modifications were done.

`--minimum-length N` or `-m N` Throw away processed reads shorter than *N* bases.

`--too-short-output FILE` Instead of throwing away the reads that are too short according to `-m`, write them to *FILE* (in FASTA/FASTQ format).

`--maximum-length N` or `-M N` Throw away processed reads longer than *N* bases.

`--too-long-output FILE` Instead of throwing away the reads that are too long (according to `-M`), write them to *FILE* (in FASTA/FASTQ format).

`--untrimmed-output FILE` Write all reads without adapters to *FILE* (in FASTA/FASTQ format) instead of writing them to the regular output file.

`--discard-trimmed` Throw away reads in which an adapter was found.

--discard-untrimmed Throw away reads in which *no* adapter was found. This has the same effect as specifying `--untrimmed-output /dev/null`.

The options `--too-short-output` and `--too-long-output` are applied first. This means, for example, that a read that is too long will never end up in the `--untrimmed-output` file when `--too-long-output` was given, no matter whether it was trimmed or not.

The options `--untrimmed-output`, `--discard-trimmed` and `-discard-untrimmed` are mutually exclusive.

Trimming paired-end reads

Atropos supports trimming of paired-end reads, trimming both reads in a pair at the same time.

Assume the input is in `reads.1.fastq` and `reads.2.fastq` and that `ADAPTER_FWD` should be trimmed from the forward reads (first file) and `ADAPTER_REV` from the reverse reads (second file).

The basic command-line is:

```
atropos -a ADAPTER_FWD -A ADAPTER_REV -o out.1.fastq -p out.2.fastq -pe1 reads.1.
↪fastq -pe2 reads.2.fastq
```

`-p` is the short form of `--paired-output`. The option `-A` is used here to specify an adapter sequence that Atropos should remove from the second read in each pair. There are also the options `-G`, `-B`. All of them work just like their lowercase counterparts, except that the adapter is searched for in the second read in each paired-end read. There is also option `-U`, which you can use to remove a fixed number of bases from the second read in a pair.

While it is possible to run Atropos on the two files separately, processing both files at the same time is highly recommended since the program can check for problems in your input files only when they are processed together.

When you use `-p`/`--paired-output`, Atropos checks whether the files are properly paired. An error is raised if one of the files contains more reads than the other or if the read names in the two files do not match. Only the part of the read name before the first space is considered. If the read name ends with `/1` or `/2`, then that is also ignored. For example, two FASTQ headers that would be considered to denote properly paired reads are:

```
@my_read/1 a comment
```

and:

```
@my_read/2 another comment
```

As soon as you start to use one of the filtering options that discard reads, it is mandatory you process both files at the same time to make sure that the output files are kept synchronized: If a read is removed from one of the files, Atropos will ensure it is also removed from the other file.

The following command-line options are applied to *both* reads:

- `-q` (along with `--quality-base`)
- `--times` applies to all the adapters given
- `--no-trim`
- `--trim-n`
- `--mask`
- `--length-tag`
- `--prefix`, `--suffix`
- `--strip-f3`
- `--colorspace`, `--bwa`, `-z`, `--no-zero-cap`, `--double-encode`, `--trim-primer`

The following limitations still exist:

- The `--info-file`, `--rest-file` and `--wildcard-file` options write out information only from the first read.
- Demultiplexing is not yet supported with paired-end data.

Filtering paired-end reads

The *filtering options listed above* can also be used when trimming paired-end data. Since there are two reads, however, the filtering criteria are checked for both reads. The question is what to do when a criterion applies to only one read and not the other.

By default, the filtering options discard or redirect the read pair if *any* of the two reads fulfill the criteria. That is, `--max-n` discards the pair if one of the two reads has too many `N` bases; `--discard-untrimmed` discards the pair if one of the reads does not contain an adapter; `--minimum-length` discards the pair if one of the reads is too short; and `--maximum-length` discards the pair if one of the reads is too long. Note that the `--discard-trimmed` filter would also apply because it is also the case that at least one of the reads is *trimmed*!

To require that filtering criteria must apply to *both* reads in order for a read pair to be considered "filtered", use the option `--pair-filter=both`.

To further complicate matters, Atropos switches to a backwards compatibility mode ("legacy mode") when none of the uppercase modification options (`-A`/`-B`/`-G`/`-U`) are given. In that mode, filtering criteria are checked only for the *first* read. Atropos will also tell you at the top of the report whether legacy mode is active. Check that line if you get strange results!

These are the paired-end specific filtering and output options:

`--paired-output FILE` or `-p FILE` Write the second read of each processed pair to *FILE* (in FASTA/FASTQ format).

`--untrimmed-paired-output FILE` Used together with `--untrimmed-output`. The second read in a pair is written to this file when the processed pair was *not* trimmed.

`--pair-filter=(any|both)` Which of the reads in a paired-end read have to match the filtering criterion in order for it to be filtered.

Note that the option names can be abbreviated as long as it is clear which option is meant (unique prefix). For example, instead of `--untrimmed-output` and `--untrimmed-paired-output`, you can write `--untrimmed-o` and `--untrimmed-p`.

Interleaved paired-end reads

Paired-end reads can be read from a single FASTQ file in which the entries for the first and second read from each pair alternate. The first read in each pair comes before the second. Enable this file format by specifying an input file using the `-l` (`--interleaved-input`) option. For example:

```
atropos -q 20 -a ACGT -A TGCA -o trimmed.fastq -l reads.fastq
```

The output can also be interleaved, using the `-L` (`--interleaved-output`) option. Atropos will detect if the input file is not properly interleaved by checking whether read names match and whether the file contains an even number of entries.

When interleaved input is used, legacy mode is disabled (that is, read-modification options such as `-q` always apply to both reads).

Legacy paired-end read trimming

Note: This section describes the way paired-end trimming was done in Cutadapt before 1.8, where the `-A`, `-G`, `-B` options were not available. It is less safe and more complicated, but you can still use it.

If you do not use any of the filtering options that discard reads, such as `--discard`, `--minimum-length` or `--maximum-length`, you can run Atropos on each file separately:

```
atropos -a ADAPTER_FWD -o trimmed.1.fastq -se reads1.fastq
atropos -a ADAPTER_REV -o trimmed.2.fastq -se reads2.fastq
```

You can use the options that are listed under 'Additional modifications' in atropos' help output without problems. For example, if you want to quality-trim the first read in each pair with a threshold of 10, and the second read in each pair with a threshold of 15, then the commands could be:

```
atropos -q 10 -a ADAPTER_FWD -o trimmed.1.fastq -se reads1.fastq
atropos -q 15 -a ADAPTER_REV -o trimmed.2.fastq -se reads2.fastq
```

If you use any of the filtering options, you must use Atropos in the following way (with the `-p` option) to make sure that read pairs remain sychronized.

First trim the forward read, writing output to temporary files (we also add some quality trimming):

```
atropos -q 10 -a ADAPTER_FWD --minimum-length 20 -o tmp.1.fastq -p tmp.2.fastq -
↪pe1 reads.1.fastq -pe2 reads.2.fastq
```

Then trim the reverse read, using the temporary files as input:

```
atropos -q 15 -a ADAPTER_REV --minimum-length 20 -o trimmed.2.fastq -p trimmed.1.
↪fastq -pe1 tmp.2.fastq -pe2 tmp.1.fastq
```

Finally, remove the temporary files:

```
rm tmp.1.fastq tmp.2.fastq
```

Please see the previous section for a much simpler way of trimming paired-end reads!

In legacy paired-end mode, the read-modifying options such as `-q` only apply to the first file in each call to Atropos (first `reads.1.fastq`, then `tmp.2.fastq` in this example). Reads in the second file are not affected by those options, but by the filtering options: If a read in the first file is discarded, then the matching read in the second file is also filtered and not written to the output given by `--paired-output` in order to keep both output files synchronized.

Multiple adapters

It is possible to specify more than one adapter sequence by using the options `-a`, `-b` and `-g` more than once. Any combination is allowed, such as five `-a` adapters and two `-g` adapters. Each read will be searched for all given adapters, but **only the best matching adapter is removed**. (But it is possible to *trim more than one adapter from each read*). This is how a command may look like to trim one of two possible 3' adapters:

```
atropos -a TGAGACACGCA -a AGGCACACAGGG -o output.fastq -se input.fastq
```

The adapter sequences can also be read from a FASTA file. Instead of giving an explicit adapter sequence, you need to write `file:` followed by the name of the FASTA file:

```
atropos -a file:adapters.fasta -o output.fastq -se input.fastq
```

All of the sequences in the file `adapters.fasta` will be used as 3' adapters. The other adapter options `-b` and `-g` also support this. Again, only the best matching adapter is trimmed from each read.

When Atropos has multiple adapter sequences to work with, either specified explicitly on the command line or via a FASTA file, it decides in the following way which adapter should be trimmed:

- All given adapter sequences are matched to the read.
- Adapter matches where the overlap length (see the `-O` parameter) is too small or where the error rate is too high (`-e`) are removed from further consideration.
- Among the remaining matches, the one with the **greatest number of matching bases** is chosen.
- If there is a tie, the first adapter wins. The order of adapters is the order in which they are given on the command line or in which they are found in the FASTA file.

If your adapter sequences are all similar and differ only by a variable barcode sequence, you should use a single adapter sequence instead that *contains wildcard characters*.

NOTE: The insert-match algorithm currently only supports using a single pair of 3' adapters.

Named adapters

Atropos reports statistics for each adapter separately. To identify the adapters, they are numbered and the adapter sequence is also printed:

```
=== Adapter 1 ===

Sequence: AACCGGTT; Length 8; Trimmed: 5 times.
```

If you want this to look a bit nicer, you can give each adapter a name in this way:

```
atropos -a My_Adapter=AACCGGTT -o output.fastq -se input.fastq
```

The actual adapter sequence in this example is `AACCGGTT` and the name assigned to it is `My_Adapter`. The report will then contain this name in addition to the other information:

```
=== Adapter 'My_Adapter' ===

Sequence: TTAGACATATCTCCGTCG; Length 18; Trimmed: 5 times.
```

When adapters are read from a FASTA file, the sequence header is used as the adapter name.

Adapter names are also used in column 8 of *info files*.

Demultiplexing

Atropos supports demultiplexing, which means that reads are written to different output files depending on which adapter was found in them. To use this, include the string `{name}` in the name of the output file and give each adapter a name. The path is then interpreted as a template and each trimmed read is written to the path in which `{name}` is replaced with the name of the adapter that was found in the read. Reads in which no adapter was found will be written to a file in which `{name}` is replaced with `unknown`.

Example:

```
atropos -a one=TATA -a two=GCGC -o trimmed-{name}.fastq.gz -se input.fastq.gz
```

This command will create the three files `demulti-one.fastq.gz`, `demulti-two.fastq.gz` and `demulti-unknown.fastq.gz`. You can *also provide adapter sequences in a FASTA file*.

In order to not trim the input files at all, but to only do multiplexing, use option `--no-trim`. And if you want to output the reads in which no adapters were found to a different file, use the `--untrimmed-output` parameter with a file name. Here is an example that uses both parameters and reads the adapters from a FASTA file (note that `--untrimmed-output` can be abbreviated):

```
atropos -a file:barcodes.fasta --no-trim --untrimmed-o untrimmed.fastq.gz -o␣
↪trimmed-{name}.fastq.gz -se input.fastq.gz
```

Trimming more than one adapter from each read

By default, at most one adapter sequence is removed from each read, even if multiple adapter sequences were provided. This can be changed by using the `--times` option (or its abbreviated form `-n`). Atropos will then search for all the given adapter sequences repeatedly, either until no adapter match was found or until the specified number of rounds was reached.

As an example, assume you have a protocol in which a 5' adapter gets ligated to your DNA fragment, but it's possible that the adapter is ligated more than once. So your sequence could look like this:

```
ADAPTERADAPTERADAPTERMYSEQUENCE
```

To be on the safe side, you assume that there are at most 5 copies of the adapter sequence. This command can be used to trim the reads correctly:

```
atropos -g ^ADAPTER -n 5 -o output.fastq -se input.fastq
```

This feature can also be used to search for *5'/3' linked adapters*. For example, when the 5' adapter is *FIRST* and the 3' adapter is *SECOND*, then the read could look like this:

```
FIRSTMYSEQUENCESECOND
```

That is, the sequence of interest is framed by the 5' and the 3' adapter. The following command can be used to trim such a read:

```
atropos -g ^FIRST -a SECOND -n 2 ...
```

Support for linked adapters is currently incomplete. For example, it is not possible to specify that SECOND should only be trimmed when FIRST also occurs. See also this feature request, and comment on it if you would like to see this implemented.

Illumina TruSeq

If you have reads containing Illumina TruSeq adapters, follow these steps.

Single-end reads as well as the first reads of paired-end data need to be trimmed with `A` + the "TruSeq Indexed Adapter". Use only the prefix of the adapter sequence that is common to all Indexed Adapter sequences:

```
atropos -a AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC -o trimmed.fastq.gz -se reads.fastq.
↪gz
```

If you have paired-end data, trim also read 2 with the reverse complement of the "TruSeq Universal Adapter". The full command-line looks as follows:

```
atropos \
            -a AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC \
            -A AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGTAGATCTCGGTGGTCGCCGTATCATT \
            -o trimmed.1.fastq.gz -p trimmed.2.fastq.gz \
            -pe1 reads.1.fastq.gz -pe2 reads.2.fastq.gz
```

See also the *section about paired-end adapter trimming above*.

If you want to simplify this a bit, you can also use the common prefix `AGATCGGAAGAGC` as the adapter sequence in both cases:

```
atropos \
            -a AGATCGGAAGAGC -A AGATCGGAAGAGC \
            -o trimmed.1.fastq.gz -p trimmed.2.fastq.gz \
            -pe1 reads.1.fastq.gz -pe2 reads.2.fastq.gz
```

The adapter sequences can be found in the document Illumina TruSeq Adapters De-Mystified.

Warning about incomplete adapter sequences

Sometimes Atropos' report ends with these lines:

```
WARNING:
    One or more of your adapter sequences may be incomplete.
    Please see the detailed output above.
```

Further up, you'll see a message like this:

```
Bases preceding removed adapters:
  A: 95.5%
  C: 1.0%
  G: 1.6%
  T: 1.6%
  none/other: 0.3%
WARNING:
    The adapter is preceded by "A" extremely often.
    The provided adapter sequence may be incomplete.
    To fix the problem, add "A" to the beginning of the adapter sequence.
```

This means that in 95.5% of the cases in which an adapter was removed from a read, the base coming *before* that was an `A`. If your DNA fragments are not random, such as in amplicon sequencing, then this is to be expected and the warning can be ignored. If the DNA fragments are supposed to be random, then the message may be genuine: The adapter sequence may be incomplete and should include an additional `A` in the beginning.

This warning exists because some documents list the Illumina TruSeq adapters as starting with `GATCGGA...`. While that is technically correct, the library preparation actually results in an additional `A` before that sequence, which also needs to be removed. See the *previous section* for the correct sequence.

Dealing with `N` bases

Atropos supports the following options to deal with `N` bases in your reads:

`--max-n COUNT` Discard reads containing more than *COUNT* `N` bases. A fractional *COUNT* between 0 and 1 can also be given and will be treated as the proportion of maximally allowed `N` bases in the read.

`--trim-n` Remove flanking `N` bases from each read. That is, a read such as this:

```
NNACGTACGTNNNN
```

Is trimmed to just `ACGTACGT`. This option is applied *after* adapter trimming. If you want to get rid of `N` bases before adapter removal, use quality trimming: `N` bases typically also have a low quality value associated with them.

Read merging and overwriting

Overlapping reads can lead to double-counting of sites in down-stream quantitative analyses that treat each read in a pair independently. Atropos can merge overlapping reads, which means that the pair will be collapsed into a single read.

Read merging is enable by the `--merge-overlapping` option. During merging, any mismatches are corrected according to the mismatch correction policy (determined by `--correct-mismatches`, see Error correction). The following parameters enable fine-tuning of the merge process:

- `--merge-min-overlap`: The minimum overlap between reads required for merging. If this number is (0,1.0], it specifies the minimum length as the fraction of the length of the *shorter* read in the pair; otherwise it specifies the minimum number of overlapping base pairs (with an absolute minimum of 2 bp).
- `--merge-error-rate`: The maximum error rate when aligning reads for merging.

IMPORTANT: By default, merged reads are discarded. Merged reads cannot be written to FASTQ output because it breaks the required read pairing between files. You can write merged reads to a separate, "single-end" FASTQ file using the `--merged-output` option.

Overwriting low-quality reads

In some atypical situations, you may end up with data in which one read in a pair is of substantially worse quality than the other read. In such cases, you may want to ignore the poor-quality read entirely and, rather than throw out the read pair, substitute the better quality read for the worse-quality read. This is done using `--overwrite-low-quality LOWQ,HIGHQ,WINDOW`, meaning that when one read has mean quality < LOWQ and the other read has mean quality >= HIGHQ over the first WINDOW bases, the better-quality read overwrites the worse-quality read.

Note that if read merging is enabled, such reads will be treated as merged and either discarded or written to `--merged-output`.

Whether read-merging occurs also depends on the order of trimming operations.

Order of modifications

Read trimming tools that support both adapter and quality trimming differ in the order in which the apply these operations. Cutadapt performs adapter trimming first, followed by quality trimming. Atropos enables you to specify the order in which you'd like operations applied using `--op-order`. The order of the following 5 operations can be customized:

- A = adapter trimming
- C = cutting (unconditional)
- G = NextSeq trimming
- Q = quality trimming
- W = overwrite poor quality reads

The default order of operations is "CGQAW" to maintain backward-compatibility with Cutadapt. However, we find that using the order "GAWCQ" generally leads to better results; this is likely to become the default in Atropos 2.0+.

Bisulfite sequencing

Proper trimming of Methyl-Seq reads is critical to accurate downstream analysis. When trimming reads that come from bisulfite-converted DNA, it is necessary to trim certain bases to avoid bias in estimating methylation. The trimming parameters are different for each type of library preparation. Atropos provides an option to enable automated trimming of Methyl-Seq reads according to the best practices recommended by library construction kit manufacturers or in the literature:

- Reduced-representation bisulfite sequencing (RRBS): RRBS relies on a

restriction enzyme (MspI) for genome fragmentation. MspI leaves a 2 bp overhang that is filled in during end-repair prior to adapter ligation. The filled-in cytosine will not be reflective of the true methylation state, and thus needs to be trimmed away. For reads in which the adapter sequence is detected, Atropos ensures that at least two additional bases are trimmed after the adapter sequence is removed. Alternatively, you can use the method that was previously recommended for use with Cutadapt:

```
atropos -a NNADAPTER -o output.fastq -se input.fastq
```

- Non-directional bisulfite sequencing: Early bisulfite sequencing protocols,

including paired-end RRBS and whole-genome bisulfite libraries constructed prior to current-generation protocols (see below), can generate strand-complementary reads whose 5' ends begin with CAA or CGA tri-nucleotides, which are also an artifact of MspI digestion. For reads in which the first three 5' bases are CAA or CGA, Atropos ensures that at least 2 bases are trimmed from the 5' end. For non-directional RRBS, the 3' 2 bp of adapter-trimmed reads are removed only if the 5' end does not start with CAA or CGA. * EpiGnome Methyl-Seq and TruSeq DNA Methylation kits: These kits introduce adapters by tagmentation of bisulfite-converted reads. Trimming of these reads beyond adapter trimming is not required. * Accel-NGS Methyl-Seq: Accel-NGS (Swift Biosciences) is a recently introduced library construction kit for directional RRBS, WGBS, and other Methyl-Seq protocols. An artifact of adding the adapter sequences is that up to 10 bp of low-complexity sequence are introduced into the 3' end of the template DNA, and thus must be trimmed away. Atropos removes 10 bp from the end of read 1 and the beginning of read 2, as recommended by the manufacturer.

Additionally, in bisulfite mode, Atropos uses an expected nucleotide frequency of 0.33 rather than 0.25 for computing random-match probabilities, since 'C' nucleotides are very infrequent. While it would be more technically correct to use nucleotide-specific probabilities for each species and assay type, in practice this level of complexity would have an impact on performance and would be unlikely to change the results substantially, as observed by Sturm et al (2016).

Micro-RNA sequencing

Several default options are recommended for miRNA-seq, including allowing a higher error rate for matching the adapter (0.12), requiring a minimum sequence length to retain reads (16), and using a minimum quality threshold for end trimming (20). Using the –mirna option sets these defaults, and, in addition, sets the adapter sequence to the Illumina small RNA adapter by default.

Multi-threading

The main advance in Atropos relative to Cutadapt is the ability to utilize multiple processors to speed up read trimming. By default, Atropos uses a single thread to read input, process reads, and write output. To use multiple cores, specify the `-T` (or `--threads`) option with the number of threads to use for trimming:

> atropos -T 4 -a ADAPTER -o output.fastq -se input.fastq

It is important to note that, whatever number of threads you give Atropos, it will use one of those for reading input and, if you use the `--compression writer` option (or if writer compression is used automatically), another for writing output. For example, the above command would use 4 cores, 1 reader thread, 1 writer thread, and 2 trimming threads. It is recommended that you have at least 2 available cores when using multi-threading.

Also note that the multi-threading model in Atropos is programmed to not make any assumptions about or impose any limiations on the runtimes of individual tasks. Instead, there is a "timeout" period, after which the log messages reporting on the status of the processing escalate from "debug" to "error." You can change the timeout length using the `--process-timeout` option. But don't be alarmed if you see messages such as the following:

> ERROR: Waiting on worker summaries for 65.0 seconds ERROR: Workers are still alive and haven't returned summaries: 1,4

This would be interpreted as "All of the data has been loaded, but worker threads 1 and 4 are still working on processing some of it." These messages are only to keep you abreast of the status – a few minutes delay is normal,

but if it starts to be tens of minutes or hours, something probably went wrong and you should kill the program (using Ctrl-C).

Parallel writing

In many cases, it is not actually necessary to write all results to the same file. For example, if the next processing step after trimming is alignment, and your aligner supports reading from multiple FASTQ files (or you are on a linux-based system and can use process substitution) to concatenate multiple files to a single input stream) then it can be much faster to have worker threads write results directly to separate files. This mode is enabled by specifying the `--no-writer-process` option, and is compatible with both local and cluster modes.

Technical details

When the `--threads` option is set, the main process loads batches of reads from the input file(s) and posts them to a Queue. One or more worker processes take batches from the Queue and process them in much the same manner as cutadapt.

If compressed output is requested, the –compression option determines if compression is performed by the worker processes or by the writer process, and the default is based on how many threads you specify and whether you are using a system with a gzip program (which includes linux and OSX) – using system-level compression is about 2x faster than compressing files through the python interface. However, we find that with 8 or more threads, having the workers perform compression leads to an overall performance gain. Thus, the default is to use worker compression unless you have both fewer than 8 threads and system-level gzip. If worker compression is used, the worker processes compress the data before placing it on the result queue, and the writer process then takes batches of results from the result queue and writes them to disk. On the other hand, if writer compression is used, the workers place uncompressed results in the result queue, and the writer compresses them (if necessary) before writing them to disk.

One thread is reserved for the reader process, but once all reads are loaded an additional worker process is started since the reader process becomes idle. With writer compression, one thread is also reserved for the writer process, so you must have more than two available threads.

Other options

`--preserve-order` Preserve order of reads in input files (ignored if –no-writer-process is set). By default, there is no guarantee as to how reads will be ordered in the output files (although read pairs are always guaranteed to be at identical positions in their respective files).

`--read-queue-size` and `--result-queue-size` Communication between the reader thread and the trimmer threads, and between the trimmer threads and the writer thread, is all done using queues. Queue sizes are limited to prevent storing too much data in memory at once. The sizes of the queues can be controlled with these two parameters. Note that queue sizes are in numbers of batches, not numbers of reads.

`--batch-size` The maximum number of reads in each batch. In our experience, this parameter tends not to have much effect on performance.

`--process-timeout` When one party tries to do a read operation on an empty queue, or a write operation on a full queue, it will wait until either something is added to/removed from the queue, or until it is told to stop (because there was an error, or because the program finished running). Each time the operation polls the queue and is told to wait, it writes a DEBUG message. However, if the wait time exceeds the value set for this parameter, then those messages are escalated to ERROR level, which suggests that the user might want to investigate.

`--compression` If ‘worker’, perform data compression in the worker (trimmer) processes; if ‘writer’, perform compression in the writer process. Otherwise, Atropos makes a choice based on whether system-level gzip is available.

Optimization

If you are going to be processing lots of data, we recommend taking some time to optimize Atropos for your particular environment. You can do this by using the `--max-reads` parameter to limit the number of input reads (we recommend 1-10 M reads to get a good idea of average processing time per read) and then experiment with multiple parameter combinations. Turning on DEBUG log messages (*–log-level DEBUG*) can also be helpful for this task. Note that increasing the number of available threads has diminishing returns, and should certainly not exceed the number of cores available on your system. We generally find 8 threads to offer the best trade-off between speed and resource usage, though this may differ for your own environment.

Atropos's output

How to read the report

After every run, Atropos prints out per-adapter statistics. The output starts with something like this:

```
Sequence: 'ACGTACGTACGTTAGCTAGC'; Length: 20; Trimmed: 2402 times.
```

The meaning of this should be obvious.

The next piece of information is this:

```
No. of allowed errors:
0-9 bp: 0; 10-19 bp: 1; 20 bp: 2
```

The adapter has, as was shown above, has a length of 20 characters. We are using the default error rate of 0.1. What this implies is shown above: Matches up to a length of 9 bp are allowed to have no errors. Matches of lengths 10-19 bp are allowd to have 1 error and matches of length 20 can have 2 errors. See also *the section about error-tolerant matching*.

Finally, a table is output that gives more detailed information about the lengths of the removed sequences. The following is only an excerpt; some rows are left out:

```
Overview of removed sequences
length  count   expect  max.err error counts
3       140     156.2   0       140
4       57      39.1    0       57
5       50      9.8     0       50
6       35      2.4     0       35
...
100     397     0.0     3       358 36 3
```

The first row tells us the following: Three bases were removed in 140 reads; randomly, one would expect this to occur 156.2 times; the maximum number of errors at that match length is 0 (this is actually redundant since we know already that no errors are allowed at lengths 0-9 bp).

The last column shows the number of reads that had 0, 1, 2 ... errors. In the last row, for example, 358 reads matched the adapter with zero errors, 36 with 1 error, and 3 matched with 2 errors.

The "expect" column gives only a rough estimate of the number of sequences that is expected to match randomly (it assumes a GC content of 50%, for example), but it can help to estimate whether the matches that were found are true adapter matches or if they are due to chance. At lengths 6, for example, only 2.4 reads are expected, but 35 do match, which hints that most of these matches are due to actual adapters.

Note that the "length" column refers to the length of the removed sequence. That is, the actual length of the match in the above row at length 100 is 20 since that is the adapter length. Assuming the read length is 100, the adapter was found in the beginning of 397 reads and therefore those reads were trimmed to a length of zero.

The table may also be useful in case the given adapter sequence contains an error. In that case, it may look like this:

```
...
length  count   expect  max.err error counts
10      53      0.0     1       51 2
11      45      0.0     1       42 3
12      51      0.0     1       48 3
13      39      0.0     1       0 39
14      40      0.0     1       0 40
15      36      0.0     1       0 36
...
```

We can see that no matches longer than 12 have zero errors. In this case, it indicates that the 13th base of the given adapter sequence is incorrect.

Saving to file

You can re-direct the summary report to a file using the `--report-file` option.

Serialized formats

Atropos collects statistics on all aspects of the trimming process. These statistics are used in generating the summary report. However, you can also choose to output the statistics in raw form into a serialized file format. Currently, three formats are supported:

- JSON
- YAML
- Pickle

Note that JSON and YAML are language-independent, text-based formats, while Pickle is a python-specific binary format.

You can save one of these formats in one of two ways:

- Change the file extension of the `--report-file`:

```
atropos --report-file summary.yaml ...
```

- Set the `--report-formats` option. When doing this, you can specify more than

one type of output file, and you use the `--report-file` option to specify the file name prefix:

```
atropos --report-file summary --report-formats txt yaml json
```

We provide an Atropos module in the MultiQC package that reads the JSON summary output directly.

TODO: Describe the format of the JSON/YAML/Pickle dictionary.

TODO: Document using custom report formats via Jinja2 templates.

Format of the info file

When the `--info-file` command-line parameter is given, detailed information about the found adapters is written to the given file. The output is a tab-separated text file. Each line corresponds to one read of the input file (unless *–times* is used, see below). The fields are:

1. Read name
2. Number of errors
3. 0-based start coordinate of the adapter match
4. 0-based end coordinate of the adapter match

5. Sequence of the read to the left of the adapter match (can be empty)
6. Sequence of the read that was matched to the adapter
7. Sequence of the read to the right of the adapter match (can be empty)
8. Name of the found adapter.
9. Quality values corresponding to sequence left of the adapter match (can be empty)
10. Quality values corresponding to sequence matched to the adapter (can be empty)
11. Quality values corresponding to sequence to the right of the adapter match (can be empty)

The concatenation of the fields 5-7 yields the full read sequence. Column 8 identifies the found adapter. *The section about named adapters <named-adapters>* describes how to give a name to an adapter. Adapters without a name are numbered starting from 1. Fields 9-11 are empty if quality values are not available. Concatenating them yields the full sequence of quality values.

If no adapter was found, the format is as follows:

1. Read name
2. The value -1
3. The read sequence
4. Quality values

When parsing the file, be aware that additional columns may be added in the future. Note also that some fields can be empty, resulting in consecutive tabs within a line.

If the `--times` option is used and greater than 1, each read can appear more than once in the info file. There will be one line for each found adapter, all with identical read names. Only for the first of those lines will the concatenation of columns 5-7 be identical to the original read sequence (and accordingly for columns 9-11). For subsequent lines, the shown sequence are the ones that were used in subsequent rounds of adapter trimming, that is, they get successively shorter.

Quality control statistics

FastQC is a popular program for generating quality control (QC) metrics on raw read data. It is common practice to examine QC metrics both before and after trimming to identify any systematic data quality issues, to observe the improvements in data quality due to trimming, and to ensure that trimming does not introduce any unintended side-effects. Since both read trimming and QC involve iterating over all reads in the dataset, we reasoned that implementing both operations in the same tool would reduce the overall processing time, and also eliminate the need to install two separate tools.

You can enable QC using the `--stats` option of the `trim` subcommand. This option takes one of three values controlling at what point(s) QC metrics are collected: pre, post, and both. The `--stats` option can take additional arguments to customize which metrics are collected. Currently, only the `tiles` parameter is supported, which enables collecting tile-level metrics (Illumina only):

```
atropos --stats pre:tiles
```

To collect tile-level metrics, Atropos must parse the read name to obtain the tile ID. By default it uses a regular expression that matches standard Illumina read names:

```
^(?:[^\:]+\:){4}([^\:]+)
```

If you need to modify this regular expression, you can pass it as an argument to the tiles argument:

```
atropos --stats "pre:tiles=<myregexp>"
```

Additionally, there is a `qc` subcommand that only collects QC metrics (i.e. it does not perform trimming).

QC metrics are added to the summary reports. Additionally, the Atropos MultiQC module can display a summary of the QC metrics.

Adapter detection

Often, details of sequencing library construction are not fully communicated from the individual or facility that generated the library to the individual(s) performing data analysis. Manual determination of sequencing adapters and other potential library contaminants can be a tedious and error-prone task. Atropos provides the `detect` subcommand to guess the most likely adapter sequence from a sample of your data:

```
atropos detect -pe1 read1.fq -pe2 read2.fq
```

There are three algorithms you can use for detection. Atropos choses one by default based on the other command line options, but you can specify an algorithm using the `-d/--detector` option. The available detectors are:

- heuristic: Use a heuristic algorithm to detect adapter sequences. This is the

slowest and most memory-intensive algorithm, but also the most accurate. * khmer: Use the khmer library to identify frequent contaminants. This requires the optional khmer dependency to be installed. This algorithm is able to detect more rare contaminants than the heuristic algorithm, and is also more memory-efficient, but it also has higher false-positive and false-negative error rates. It is recommended to only use this algorithm if the heuristic algorithm fails. * known: Only match reads against known adapter sequences. The previous two algorithms can also match detected contaminant sequences against known adapters.

Because adapter sequences have been designed not to match any known sequence in nature, a sequence (or pair of sequences) that occurs at high frequency and matches a known adapter sequence is likely to be the true sequence(s) used as adapters in the dataset. Thus, our algorithm optionally matches the high-abundance `k`-mers to a list of known adapters/contaminants. By default, Atropos uses a curated list of commonly used adapter sequences. You can specify your own file (in FASTA format) using the `--known-adapters-file` option (see "Specifying adapters by name for details). When a contaminant list is not provided, or when the adapter does not match a known sequence, we advise you to take caution when using the results of this detection process, as a highly abundant sequence might simply be derived from a frequently repeated element in the genome.

Error rate estimation

One of the most important parameters for adapter trimming is the error rate (`-e`). For high-quality Illumina data, the default settings are sufficient. However, it's difficult to know the data quality a priori. Atropos provides the `error` subcommand to estimate the error rate for setting the `-e` option:

```
atropos error -pe1 read1.fq -pe2 read2.fq
```

There are two error rate estimation algorithms provided. The default algorithm simply averages the base quality scores in a sample of reads. This is likely to be an overestimation of the true error rate, but computing it is very fast. A more accurate but *much* slower algorithm is Shadow Regression (Wang et al., "Estimation of sequencing error rates in short reads", BMC Bioinformatics 2012 13:185, DOI: 10.1186/1471-2105-13-185). Using the shadow regression algorithm requires for R to be installed, as well as the `ShadowRegression` R package.

Once you've estimated the error rate, we recommend setting the `-e` option to ~10X the error rate. For example, if the estimated error is 0.9% (0.009), a good value for `-e` is 0.1.

CHAPTER 3

Colorspace reads

Atropos was designed to work with colorspace reads from the ABi SOLiD sequencer. Colorspace trimming is activated by the `--colorspace` option (or use `-c` for short). The input reads can be given either:

- in a FASTA file
- in a FASTQ file
- in a `.csfasta` and a `.qual` file (this is the native SOLiD format).

In all cases, the colors must be represented by the characters 0, 1, 2, 3. Example input files are in the atropos distribution at `tests/data/solid.*`. The `.csfasta`/`.qual` file format is automatically assumed if two input files are given to atropos.

In colorspace mode, the adapter sequences given to the `-a`, `-b` and `-g` options can be given both as colors or as nucleotides. If given as nucleotides, they will automatically be converted to colorspace. For example, to trim an adapter from `solid.csfasta` and `solid.qual`, use this command-line:

```
atropos -c -a CGCCTTGGCCGTACAGCAG solid.csfasta solid.qual > output.fastq
```

In case you know the colorspace adapter sequence, you can also write `330201030313112312` instead of `CGCCTTGGCCGTACAGCAG` and the result is the same.

Ambiguity in colorspace

The ambiguity of colorspace encoding leads to some effects to be aware of when trimming 3' adapters from colorspace reads. For example, when trimming the adapter `AACTC`, atropos searches for its colorspace-encoded version `0122`. But also `TTGAG`, `CCAGA` and `GGTCT` have an encoding of `0122`. This means that effectively four different adapter sequences are searched and trimmed at the same time. There is no way around this, unless the decoded sequence were available, but that is usually only the case after read mapping.

The effect should usually be quite small. The number of false positives is multiplied by four, but with a sufficiently large overlap (3 or 4 is already enough), this is still only around 0.2 bases lost per read on average. If inspecting k-mer frequencies or using small overlaps, you need to be aware of the effect, however.

Double-encoding, BWA and MAQ

The read mappers MAQ and BWA (and possibly others) need their colorspace input reads to be in a so-called "double encoding". This simply means that they cannot deal with the characters 0, 1, 2, 3 in the reads, but require that the letters A, C, G, T be used for colors. For example, the colorspace sequence `0011321` would be `AACCTGC` in double-encoded form. This is not the same as conversion to basespace! The read is still in colorspace, only letters are used instead of digits. If that sounds confusing, that is because it is.

Note that MAQ is unmaintained and should not be used in new projects.

BWA's colorspace support was dropped in versions more recent than 0.5.9, but that version works well.

When you want to trim reads that will be mapped with BWA or MAQ, you can use the `--bwa` option, which enables colorspace mode (`-c`), double-encoding (`-d`), primer trimming (`-t`), all of which are required for BWA, in addition to some other useful options.

The `--maq` option is an alias for `--bwa`.

Colorspace examples

To cut an adapter from SOLiD data given in `solid.csfasta` and `solid.qual`, to produce MAQ- and BWA-compatible output, allow the default of 10% errors and write the resulting FASTQ file to output.fastq:

```
atropos --bwa -a CGCCTTGGCCGTACAGCAG solid.csfasta solid.qual > output.fastq
```

Instead of redirecting standard output with >, the `-o` option can be used. This also shows that you can give the adapter in colorspace and how to use a different error rate:

```
atropos --bwa -e 0.15 -a 330201030313112312 -o output.fastq solid.csfasta solid.
↪qual
```

This does the same as above, but produces BFAST-compatible output, strips the _F3 suffix from read names and adds the prefix "abc:" to them:

```
atropos -c -e 0.15 -a 330201030313112312 -x abc: --strip-f3 solid.csfasta solid.
↪qual > output.fastq
```

Bowtie

Quality values of colorspace reads are sometimes negative. Bowtie gets confused and prints this message:

```
Encountered a space parsing the quality string for read xyz
```

BWA also has a problem with such data. Atropos therefore converts negative quality values to zero in colorspace data. Use the option `--no-zero-cap` to turn this off.

Sequence Read Archive

The Sequence Read Archive provides files in a special "SRA" file format. When the `fastq-dump` program from the sra-toolkit package is used to convert these `.sra` files to FASTQ format, colorspace reads will get an extra quality value in the beginning of each read. You may get an error like this:

```
atropos: error: In read named 'xyz': length of colorspace quality
sequence (36) and length of read (35) do not match (primer is: 'T')
```

To make atropos ignore the extra quality base, add `--format=sra-fastq` to your command-line, as in this example:

```
atropos -c --format=sra-fastq -a CGCCTTGGCCG sra.fastq > trimmed.fastq
```

When you use `--format=sra-fastq`, the spurious quality value will be removed from all reads in the file.

CHAPTER 4

Recipes

For some trimming applications, the pre-defined adapter types behave differently from what you would like to have. In this section, we show some ways in which atropos can be made to behave in the desired way.

Note: This section is still being written.

Forcing matches to be at the end of the read

Use `-a TACGGCATXXX`. The `X` is always counted as a mismatch and will force the adapter match to be at the end. This is not the same as an anchored 3' adapter since partial matches are still allowed.

Removing more than one adapter

If you want to remove more than one adapter, let's say a 5' adapter and a 3' adapter, you have two options.

First, you can specify both adapters and also `--times=2` (or the short version `-n 2`). For example:

```
atropos -g ^TTAAGGCC -a TACGGACT -n 2 -o output.fastq input.fastq
```

This instructs atropos to run two rounds of adapter finding and removal. That means that, after the first round and only when an adapter was actually found, another round is performed. In both rounds, all given adapters (two in this case) are searched and removed. The problem is that it could happen that one adapter is found twice (so the 3' adapter, for example, could be removed twice).

The second option is to not use the `-n` option, but to run atropos twice, first removing one adapter and then the other. It is easiest if you use a pipe as in this example:

```
atropos -g ^TTAAGGCC input.fastq | atropos -a TACGGACT - > output.fastq
```

Trimming poly-A tails

If you want to trim a poly-A tail from the 3' end of your reads, use the 3' adapter type (`-a`) with an adapter sequence of many repeated `A` nucleotides. Starting with version 1.8 of atropos, you can use the following notation to specify a sequence that consists of 100 `A`:

```
atropos -a "A{100}" -o output.fastq input.fastq
```

This also works when there are sequencing errors in the poly-A tail. So this read

```
TACGTACGTACGTACGAAATAAAAAAAAAAA
```

will be trimmed to:

```
TACGTACGTACGTACG
```

If for some reason you would like to use a shorter sequence of `A`, you can do so: The matching algorithm always picks the leftmost match that it can find, so atropos will do the right thing even when the tail has more `A` than you used in the adapter sequence. However, sequencing errors may result in shorter matches than desired. For example, using `-a "A{10}"`, the read above (where the `AAAT` is followed by eleven `A`) would be trimmed to:

```
TACGTACGTACGTACGAAAT
```

Depending on your application, perhaps a variant of `-a A{10}N{90}` is an alternative, forcing the match to be located as much to the left as possible, while still allowing for non-`A` bases towards the end of the read.

Other things (unfinished)

- How to detect adapters
- Use atropos for quality-trimming only
- Use it for minimum/maximum length filtering
- Use it for conversion to FASTQ

CHAPTER 5

Changes

v1.1.5 (dev)

- Major update to the documentation.

v1.1.4 (2017.05.02)

- Exposed option to set PRNG seed when subsampling reads.
- Fixed issue #14: 'detect' and 'error' commands were broken. This involved rewriting those commands to use the same pipeline and reporting frameworks as the 'trim' and 'qc' commands. >>>>>>> issue14

v1.1.3 (2017.05.01)

- Updated Dockerfile to use smaller, Alpine-based image.
- Added Docker image for v1.1.2 to Docker Hub.
- Updated Travis config to automatically build Docker images for each release.
- Ported over improvements to adapter parsing (635eea9) from Cutadapt.
- Fixed #12: tqdm progress bar not working.
- Fixed #13: unnecessary differences in summary output between Cutadapt and Atropos.

v1.1.2 (2017.04.12)

- New 'qc' command computes read-level statistics.
- The 'trim' command can also compute read-level statistic pre- and/or post-trimming using the new '–stats' option.
- Major refactoring and improvement of reporting:
  - Text report now has data lined up in columns.

  - Reports can also be generated in JSON, YAML, and pickle formats.
  - Added optional dependency on jinja2, which enables generating reports using templates (including user-defined).
- Major internal code reorganization.
- Static code analysis (pylint).
- Switched to pytest for testing.
- Command-specific help will now show with 'atropos ' or 'atropos -h'
- Fixed adapter masking in InsertAligner (issue #7; thanks @lllaaa).
- Added developer/contributor documentation and guidelines.
- Implemented Atropos module for MultiQC, which reads reports in JSON format. This is currently available here and will hopefully soon be merged into MultiQC.
- Ported some recent enhancments over from Cutadapt.

v1.0.23 (2016.12.07)

- Identified a subtle bug having to do with insufficient memory in multi-threaded mode. The main thread appears to hang waiting for the next read from the input file. This appears to occur only under a strictly-regulated memory cap such as on cluster environment. This bug is not fixed, but I added the following:
  - Set the default batch size based on the queue sizes
  - Warn the user if their combination of batch and queue sizes might lead to excessive memory usage.
- Bug fixes

v1.0.22 (2016.12.02)

- Abstracted the ErrorEstimator class to enable alternate implementations.
- Added a new ShadowRegressionErrorEstimator that uses the ShadowRegression R package (Wang et al.) to more accurately estimate sequencing error rate. This requires that R and the ShadowRegression package and its dependencies be installed – MASS and ReadCount, which in turn depend on a bunch of Bioconductor packages. At some point, this dependency will go away when I reimplement the method in pure python.
- The error command now reports the longest matching read fragment, which is usually a closer match for the actual adapter sequence than the longest matching k-mer.

v1.0.21 (2016.11.23)

- Bugfixes

v1.0.20 (2016.11.22)

- Changed the order of trimming operations - OverwriteReadModifier is now after read and quality trimming.
- Refactored the main Atropos interface to improve testability.
- Added more unit tests.

v1.0.19 (2016.11.21)

- Fixed a major bug in OverwriteReadModifier, and in the unit tests for paired-end trimmers.

v1.0.18 (2016.11.20)

- Added OverwriteReadModifier, a paired-end modifier that overwrites one read end with the other if the mean quality over the first N bases (where N is user-specified) of one is below a threshold value and the mean quality of the other is above a second threshold. This dramatically improves the number of high-quality read mappings in data sets where there are systematic problems with one read-end.

v1.0.17 (2016.11.18)

- Perform error correction when insert match fails but adapter matches are complementary
- Improvements to handling of cached adapter lists
- Merged reads are no longer written to –too-short-output by default
- Many bugfixes and improvements in deployment (including a Makefile)
- Many

v1.0.16 (2016.09.20)

- Migrate to Versioneer for version management.
- Enable stderr as a valid output using the '_' shortcut.
- Add ability to specify SAM/BAM as input format.
- Add option to select which read to use when treating a paired-end interleaved or SAM/BAM file as single-end.
- Remove restrictions on the use of –merge-overlapping, and enable error correction during merging.
- We are beginning to move towards the use of commands for all operations, and read-trimming now falls under the 'trim' command. Currently, calling atropos without any command will default to the 'trim' command.
- When InsertAdapterCutter.symmetric is True and mismatch_action is not None, insert match fails, at least one adapter match succeeds, and the adapter matches (if there are two) are complementary, then the reads are treated as overlapping and error correction is performed. This leads to substantial improvements when one read is of good quality while the other is other is of poor quality.

v1.0.15 (2016.09.14)

- Fixed missing import bug in 'detect' command.
- Added estimate of fraction of contaminated reads to output of 'detect' command.
- Optionally cache list of known contaminants rather than re-download it every time.

v1.0.14 (2016.09.13)

- Implemented _align.MultiAligner, which returns all matches that satisfy the overlap and error thresholds. align.InsertAligner now uses MultiAligner for insert matching, and tests all matches in decreasing size order until it finds one with adapter matches (if any).
- Major improvements to the accuracy of the 'detect' command.
- Added options for how to correct mismatched bases for which qualities are equal.
- Added option to select a single pair of adapters from multiple sequences in a fasta file.
- Fixed report when insert-match is used.
- Fixed several bugs when using the "message" progress bar (thanks to Thomas Cokelaer!).
- Fixed a segmentation fault that occurs when trying to trim zero-length reads with the insert aligner.
- Sevaral other bugfixes.

v1.0.13 (2016.08.31)

- Add options to specify max error rates for insert and adapter matching within insert aligner.
- Add new command to estimate empirical error rate in data set from base qualities.

v1.0.12 (2016.08.30)

- Add ability to correct errors during insert-match adapter trimming.
- Implement additional adapter-detection algorithms.
- Fix bug where default output file is force-created in parallel-write mode

v1.0.11 (2016.08.24)

- Clarify and fix issues with bisulfite trimming. Notably, rrbs and non-directional are now allowed independently or in combination.

v1.0.10 (2016.08.23)

- Introduced new 'detect' command for automatically detecting adapter sequences.
- Options are now required to specify input files.
- Major updates to documentation.

v1.0.9 (2016.08.22)

- Bugfix release

v1.0.8 (2016.08.19)

- Reverted previously introduced (and no longer necessary) dependency on bitarray).
- Switched the insert aligner back to the default implementation, as the one that ignores indels is not any faster.

v1.0.7 (2016.08.18)

- Re-engineered modifiers.py (and all dependent code) to enable use of modifiers that simultaneously edit both reads in a pair.
- Add –op-order option to enable use to specify order of first four trimming operations.
- Implemented insert-based alignment for paired-end adapter trimming. This is currently experimental. Benchmarking against SeqPurge and Skewer using simulated reads showed that the method Cutadapt uses to align adapters, while optimal for single-end reads, is much less sensitive and specific than the insert match algorithms used by SeqPurge and Skewer. Our algorithm is similar to the one used by SeqPurge but leverages the dynamic programming model of Cutadapt.

v1.0.6 (2016.08.08)

- Based on tests, worker compression is faster than writer compression when more than 8 threads are available, so set this to be the default.

v1.0.5 (2016.08.06)

- Interanal code reorganization - compression code moved to separate module
- Eliminated the –worker-compression option in favor of –compression (whose value is either ‘worker’ or ‘writer’)
- More documentation improvements

v1.0.3 (2016.08.05)

- Significant performance improvements:
  - Start an extra worker once the main process is finished loading reads
  - Use system-level gzip for writer compression
  - Use writer compression by default
- More documentation fixes
- Disable quality trimming if all cutoffs are set to 0
- Eliminated the –parallel-environment option

v1.0.1 (2016.08.04)

- Fix documentation bugs associated with migration from optparse to argparse

v1.0 (2016.07.29)

- Initial release (forked from cutadapt 1.10)
- Re-wrote much of filters.py and modifiers.py to separate modifying/filtering from file writing.
  - File writing is now managed by a separate class (seqio.Writers)
  - There are container classes for managing filters (filters.Filters) and modifiers (modifiers.Modifiers)
- Re-wrote all of the output-oriented code in seqio.py
  - Formatting Sequence objects is now separate from writing data
  - There is a container class (seqio.Formatters) that manages the formatters for output files
  - Added support for interleaved output
- Implemented multiprocessing
  - Added several new options in scripts.atropos to control parallelization
  - Wrote all of the parallel processing code in atropos.multicore
  - Renamed scripts.atropos.process_single_reads() to scripts.atropos.run_serial() and rewrote to work similarly to atropos.multicore.run_parallel()
  - Added ability to merge report statistics from multiple worker threads
- Added miRNA and bisulfite sequencing options to scripts.atropos
- Added progress bar support
- Switched argument parsing to argparse
- Reorganized the monolithic scripts.atropos.main() into multiple functions
- Dropped all support for python 2.x