# SEAStAR tools User Guide, version 0.3.0

Vaughn Iverson, Chris Berthiaume

Armbrust Lab*
School of Oceanography
University of Washington

March 2, 2012

## Contents

## 1 Introduction

The SEAStAR tools are a collection of programs which implement many data and processing intensive steps in an analysis pipeline producing quantitative analysis of alignments to reference databases, and/or binned assembled genomic scaffolds, directly from raw metagenomic next-generation sequence reads. (Figure 1).

*http://armbrustlab.ocean.washington.edu

## 1.1 Philosophy and limitations

SEAStAR used to be an acronym for Select and Estimate Abundance from Short Aligned Reads. However, the set of tools and their capabilities have expanded well beyond that description, and so now SEAStAR is just the name of our whole analysis pipeline.

Our philosophy has been to create stable, modular, high-performance tools that fill important gaps we encountered between other established pre-existing tools in our pipeline. As such, you will see references to other packages in Figure 1 and throughout this document. We recognize that considerable innovation is ongoing in analysis software for next generation sequence, and so we have maintained a modular, pipelined approach to permit experimentation with, and substitution of, new tools as they become available. Due to the wide variety of different computational environments that undoubtedly exist in different labs and institutes, our focus will be on releasing stable, documented, portable and high performance *components* from our analysis pipeline, but not the pipeline scripts themselves. This is because these scripts will need to be highly customized for both the goals and computational environment of any specific project.

Initially our goal is to release tools to enable others to work directly with datasets generated by the ABI SOLiD$^{\text{TM}}$sequencing platform. We recognize that some of the components we have developed may be valuable for use with Illumina® sequence data, or with a mixture of sequence data types, but we will be placing the highest priority on completing our support for SOLiD data before evaluating which components in our pipeline are good candidates for generalizing to support both colorspace and nucleotide reads (see section 4 for more discussion of SOLiD data).

## 1.2 Planned release phases

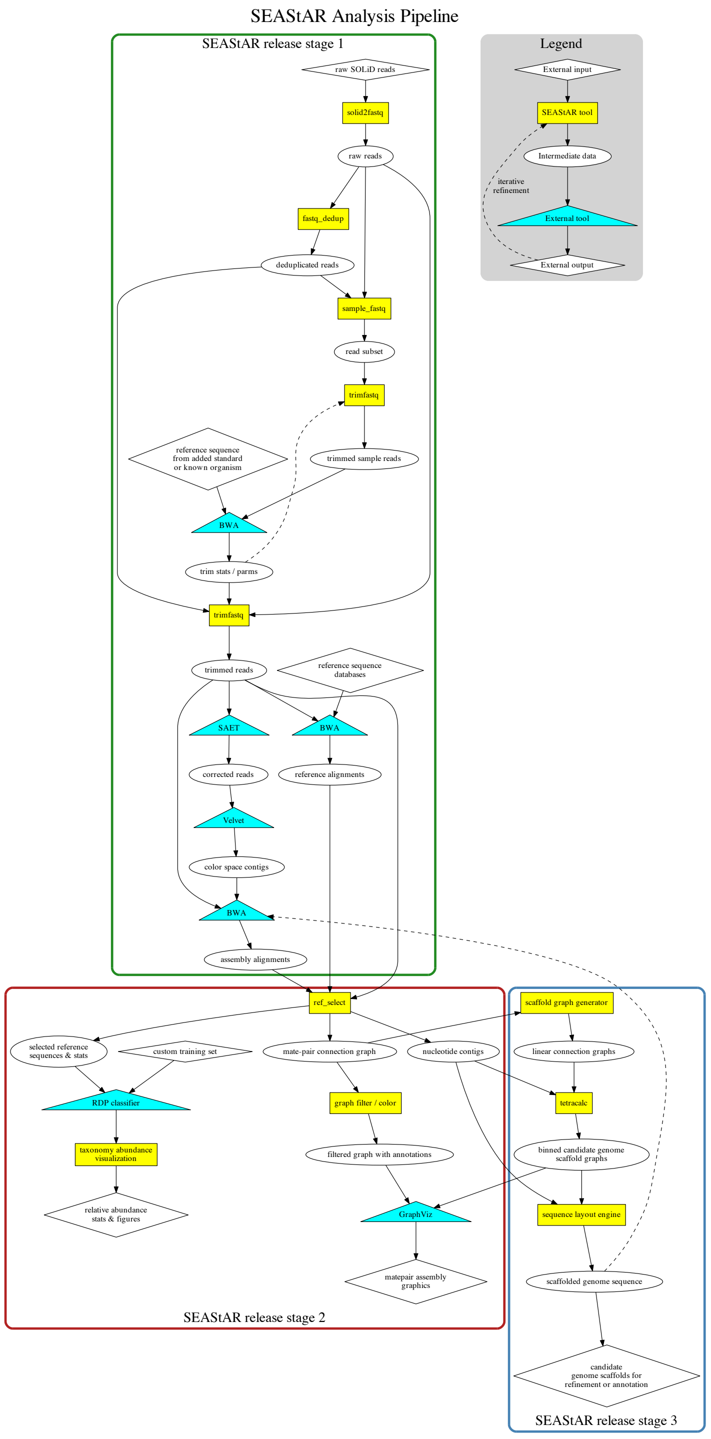
The SEAStAR analysis pipeline is being released in three stages (Figure 1). For up-to-date status information about the development and release process, please visit the SEAStAR website[1].

**Stage 1 (current):** Contains the tools we have developed for working with SOLiD reads, converting them to colorspace FASTQ files, removing PCR duplicates, and trimming the reads based on quality and information content for various uses. Optional read error correction is performed by the SAET tool (a component of the denovo2 package[2]). This code will enable the user to produce files that can be directly used with the BWA and Velvet packages for colorspace alignment and assembly, respectively.

**Stage 2 (future):** Contains tools for processing SAM[3] alignment files and assembled colorspace contigs into mate-paired assembly graphs with associated nucleotide contigs that can then be visualized and explored using GraphViz[4]. This stage's tools will also enable selection and visualization the best sequences from a reference database (e.g. the RDP[5] 16S database) alignment with associated statistics such as coverage and relative abundance. In the case of alignments with RDP, these sequences can then be classified (using the RDP classifier[6]) and visualized taxonomically.

[1]http://armbrustlab.ocean.washington.edu/seastar
[2]http://solidsoftwaretools.com/gf/project/denovo/frs/
[3]http://samtools.sourceforge.net/
[4]http://www.graphviz.org/
[5]http://rdp.cme.msu.edu/
[6]http://sourceforge.net/projects/rdp-classifier/

Figure 1: Software pipeline diagram showing the planned release phases for the SEAStAR software.



**Stage 3 (future):** Contains tools for splitting assembly graphs into parsimonious linear scaffold graphs, binning these scaffold graphs using tetra-nucleotide statistics, and constructing scaffold sequences with the order and orientation of contigs properly determined. Use of stage 3 tools enables production of candidate genome assemblies ready for refinement, gap-filling (if desired) and annotation.

## 1.3 Technical advice

The next-generation sequence datasets that the SEAStAR tools are designed to work with are very large (> one billion reads). All compiled components of the SEAStAR tools are threaded and optimized for modern multi-core processors. Many SEAStAR tools will take advantage of 8 or more cores, particularly when compressed (gzip format) input and output files are being processed. When running on such a system, you will find that these components may run considerably more than 10× faster than equivalent tools written in a scripting language such as Perl or Python. For this reason, the operation of such tools will often be "I/O bound"; that is: the performance bottleneck is the speed of the disk(s) and/or network connections connected to the workstation, and not the availability of CPU cycles. The use of compressed files and local, independent disks for input and output files can often relieve some of this bottleneck and further improve run times.

Several of the SEAStAR tools—as well as many third-party tools used in our pipeline—have very high memory requirements when working with typical next-generation sequence datasets. We note these cases in the discussion of each individual SEAStAR tool, and provide guidance about the impact different parameters will have on memory use. However, in general, doing this kind of work will require a workstation with at least 64 gigabytes of DRAM, and depending on the dataset and the types of analysis your project requires, you may require some multiple of that amount (128, 256, or 512 gigabytes) to successfully implement an analysis pipeline that meets your needs. In any case, when running these tools, it is highly advisable to set a session limit on the memory a process can attempt to allocate, to prevent machines from crashing and impacting other users who may be logged in. For example, adding the Unix shell command:

```
ulimit -v 64000000
```

to any script that invokes high memory tools will effectively prevent a 64GB machine from being unintentionally brought down (i.e. thrashing the swap file) by an application that attempts to allocate more memory than is physically available.

## 1.4 License and citation

The SEAStAR tools are "open source" and are currently publicly distributed under the GPL version 3 license[7]. Please contact the authors if you are interested in negotiating an alternative licensing arrangement.

We are in the process of preparing several publications specific to the SEAStAR analysis pipeline and the custom tools it contains. In the interim, we ask that you cite the following paper (which was the first to use and describe these tools):

[7]http://gplv3.fsf.org/

Iverson, V., Morris, R. M., Frazar, C. D., Berthiaume, C. T., Morales, R. L .and Armbrust, E. V., Untangling Genomes from Metagenomes: Revealing an Uncultured Class of Marine Euryarchaeota, *Science* **335** pp.587-590 (2012)

## 2 Quick Start

This section is a quick introduction to using SEAStAR tools for common use cases. For installation instructions see the `README` file in the SEAStAR source tree. Data used in these examples can be found in the `test_data` directory of the SEAStAR source tree. Copy the following files from that directory to a new working directory.

```
test_data/lambda_reads_F3.csfasta.gz
test_data/lambda_reads_F3_QV.qual.gz
test_data/lambda_reads_R3.csfasta.gz
test_data/lambda_reads_R3_QV.qual.gz
test_data/lambda.fasta.gz
```

### 2.1 Read preparation

Convert SOLiD `.csfasta` and `.qual` files to gzipped `.fastq` files (see Section 4 for FASTQ format details).

```
solid2fastq -z lambda_reads lambda_conv
```

Remove presumed PCR duplicate reads, identified by mate-pairs seen more than once.

```
fastq_nodup -z -l 13 -d 2 -e 3 lambda_conv lambda_dedup
```

Trim and filter the de-duplicated FASTQ reads based on read quality, information content, and length.

```
trimfastq -z -p 0.5 -l 34 -m 34 -e 3.0 --add_len lambda_dedup lambda_trim
```

Randomly sample reads from `.fastq` files, retaining approximately 10% of the original reads. Randomly subsampled read sets are useful for quickly tuning the parameters of `trimfastq` for a given set of reads against some reference.

```
samplefastq -z -f 0.1 lambda_trim lambda_trim_samp
```

### 2.2 Alignment to a reference

If the short read aligner BWA[8] (version $\leq$ 0.5.10) is installed it can be used to align de-duplicated and trimmed lambda FASTQ reads against the lambda reference genome. The resulting alignment file is in SAM format.

[8] Available from: http://bio-bwa.sourceforge.net/

```
bwa index -a is -c lambda.fasta.gz

bwa aln -c -n 0.001 -l 18 lambda.fasta.gz lambda_trim.read1.fastq.gz
     >lambda_trim.read1.sai

bwa samse -n 1000000 lambda.fasta.gz lambda_trim.read1.sai
     lambda_trim.read1.fastq.gz 2>lambda_trim.read1.samse.log
     >lambda_trim.read1.sam

bwa aln -c -n 0.001 -l 18 lambda.fasta.gz lambda_trim.read2.fastq.gz
     >lambda_trim.read2.sai

bwa samse -n 1000000 lambda.fasta.gz lambda_trim.read2.sai
     lambda_trim.read2.fastq.gz 2>lambda_trim.read2.samse.log
     >lambda_trim.read2.sam

bwa aln -c -n 0.001 -l 18 lambda.fasta.gz lambda_trim.single.fastq.gz
     >lambda_trim.single.sai

bwa samse -n 1000000 lambda.fasta.gz lambda_trim.single.sai
     lambda_trim.single.fastq.gz 2>lambda_trim.single.samse.log
     >lambda_trim.single.sam
```

### 2.3 De novo assembly

If the colorspace aware build of the de novo assembly tool Velvet[9] is installed it can be used to assemble lambda colorspace contigs. First re-trim reads at a stricter quality cutoff and output a Velvet style `.fastq` file of interleaved mate-pairs, then assemble with Velvet.

```
trimfastq -z --mates_file -p 0.9 -l 34 -m 34 --add_len -e 3.0 lambda_dedup
     lambda_trimvelvet

velveth_de lambda_asm/ 15 -fastq.gz -shortPaired lambda_trimvelvet.mates.fastq.gz
     -short lambda_trimvelvet.single.fastq.gz >lambda_asm.velveth_de.log 2>&1

velvetg_de lambda_asm/ -scaffolding no -read_trkg no -ins_length auto
     -ins_length_sd auto -exp_cov 50 -cov_cutoff 5 -min_contig_lgth 50
     >lambda_asm.velvetg_de.log 2>&1
```

## 3 Command Reference

### 3.1 solid2fastq

This tool converts SOLiD `.csfasta` and `.qual` input files (raw reads transferred off the instrument) to output files that are an interoperable variant of the FASTQ file format (see SEAStAR FASTQ

[9]Available from: http://www.ebi.ac.uk/∼zerbino/velvet/

format details in section 4).

Usage: `solid2fastq [options] <in_prefix> <out_prefix>`

Where `<in_prefix>` denotes the input prefix which is the part of the input filename shared in common between all `.csfasta` and `.qual` files generated by the same sequence library. It is typically shown in the `Title:` comment line near the top of `.csfasta` format input files. For example:

```
# Title: GG050409_20090604_matepair_50
```

is found near the top of input files:

```
GG050409_20090604_matepair_50_R3.csfasta
GG050409_20090604_matepair_50_R3_QV.qual
GG050409_20090604_matepair_50_F3.csfasta
GG050409_20090604_matepair_50_F3_QV.qual
```

And so use of `GG050409_20090604_matepair_50` for `<in_prefix>` will specify these four files for input to the `solid2fastq` tool.

Note that `solid2fastq` relies on the SOLiD naming conventions for the suffixes of these files (e.g. `_R3.csfasta`). You are free to rename these files with different prefixes, but the suffixes must remain the same. You may, however, compress these files using gzip and add the customary `.gz` suffix at the very end of the filename, which tells `solid2fastq` to decompress the files as it reads them.

The `<out_prefix>` parameter is similar to the `<in_prefix>` as it is used to name the output `.fastq` file(s) generated by this tool. The output prefix may be any text, except you should generally avoid whitespace and punctuation characters other than "_". This is because, by default, the names of all output reads are also prefixed with this string and the variant of the FASTQ format used by the SEAStAR tools depends on certain punctuation characters to quickly parse these read names (see SEAStAR FASTQ format in section 4 for details). This file naming constraint may be relaxed through the use of optional parameters described below.

Note that `<in_prefix>` and/or `<out_prefix>` may contain directory path information, but in the case of `<out_prefix>` this will necessitate the use of the `--prefix` or `--no_prefix` options (described below).

`solid2fastq` writes a single output `.fastq` file for each matching pair of `.csfasta` / `.qual` input files. The names of these output files are constructed as:

`<in_prefix>_F3[.csfasta|_QV.qual]` $\rightarrow$ `<out_prefix>.read1.fastq`
`<in_prefix>_R3[.csfasta|_QV.qual]` $\rightarrow$ `<out_prefix>.read2.fastq`

singlets from `F3/R3` $\rightarrow$ `<out_prefix>.single.fastq`

Note that for paired-end libraries, alternate naming suffix conventions for the `R3` input are also recognized (substituting `F5-BC` or `F5-P2` for `R3`).

Additionally, for paired libraries (mate-paired or paired-end) should `solid2fastq` encounter any singlet reads (those with a missing mate), they are by default given the appropriate suffix and then written to a file named:

```
        <out_prefix>.single.fastq
```

`solid2fastq` optional parameters `[options]`:

```
        [-h|--help] [--version] [--prefix=<string>] [-n|--no_prefix]
        [-z|--gzip] [-b|--bc=<BC>] [-s|--singles]
```

`-h`
`--help`

Print a guide to valid command line parameters and their correct usage to the terminal, and then exit.

`--version`

Print the version number of the executing program and then exit.

`--prefix=<string>`

Specify the prefix string to add to read identifiers. For example, `.csfasta` read identifier:

```
        >1_68_381_F3
```

will become:

```
        @xx+<string>:1_68_381/1
```

in the `.fastq` output file. By default this is `<out_prefix>` described above.

The use of a terse, but unique, prefix string is advisable because it helps identify the source, and processing steps, performed on a set of reads as they flow through an analysis pipeline. Keep in mind that this text will be added to the output file for every read, so short strings are preferable. The output prefix may be any text you would like, except you may not use whitespace or punctuation characters other than "`_`".

`-n`
`--no_prefix`

Overrides the default read prefixing behavior and instead preserves the read identifiers as they are in the input `.csfasta` file(s). For example:

```
        >1_68_381_F3
```

simply becomes:

```
        @xx+:1_68_381/1
```

`-z`
`--gzip`

Write output `.fastq` files compressed in the gzip format, and with filenames suffixed `.gz`.

`-b <BC>`
`--bc=<BC>`

Used only for barcoded SOLiD libraries, in addition to `<in_prefix>`, to specify input files for a specific barcode (`<BC>`).

`-s`
`--singles`

Write two separate singlet files that segregate the singlets by input files. This is useful for doing strand-specific analyses using paired-end libraries, where the mated reads are sequenced from opposite strands. For example:

> `<in_prefix>_F3[.csfasta|_QV.qual]` → `<out_prefix>.read1.fastq`
> singlets from `F3` → `<out_prefix>.single1.fastq`
>
> `<in_prefix>_R3[.csfasta|_QV.qual]` → `<out_prefix>.read2.fastq`
> singlets from `R3` → `<out_prefix>.single2.fastq`

## 3.2 fastq_nodup

This tool reads paired `.fastq` files (with associated singlets), and removes the lowest quality pair(s) of presumed PCR duplicate reads. This operation assumes that multiple read-pairs sharing substantially the same sequences for both mates are statistically unlikely to occur frequently at random (due to the distribution of insert sizes), and therefore represent likely artifactual duplications resulting from PCR over-amplification during library construction. If this assumption does not hold for a given library, then use of this tool is inappropriate.

Notably, this tool is most appropriate for metagenomes—or any other sample type—where no reference genome is available for performing duplicate removal through reference alignments. This is because `fastq_nodup` works entirely through error-tolerant internal comparisons of read-pairs to each other within an entire sequenced library.

To accomplish this, `fastq_nodup` builds a large table of sequence prefixes, requiring a substantial amount of memory. The amount of memory required can be controlled through tuning the various parameters, but a minimum of 32Gb will probably be necessary for most read datasets.

Note that singlet reads are also randomly removed in the same proportion as matching mated reads. This is done assuming that PCR duplicated pairs are as likely to produce singlets as non-duplicated pairs. For example:

```
S1, S1, M1 -- M2, M1 -- M2, M1 -- M3
```

In the above example `S`=singlet reads, `M`=mated reads, and the numeric suffixes indicate matching sequences. So the lowest quality pair of `M1--M2` mates will be removed from the output, and each of the `S1` reads will have a $\frac{1}{3}$ probability of being removed (disregarding quality). The $\frac{1}{3}$ probability is calculated from the fact that $\frac{1}{3}$ of the pairs containing sequence `M1` were presumed PCR duplicates and therefore removed.

Usage: `fastq_nodup [options] <in_prefix> <out_prefix>`

where `<in_prefix>` and `<out_prefix>` specify the input and output filename prefixes to use for input files and naming output files. For example:

```
<in_prefix>.read1.fastq → <out_prefix>.read1.fastq
<in_prefix>.read2.fastq → <out_prefix>.read2.fastq
<in_prefix>.single.fastq → <out_prefix>.single.fastq
```

The input files may be gzip format compressed files with the customary `.gz` suffix at the very end of the filename, which tells `solid2fastq` to decompress the files as it reads them. Multiple single files (i.e. `.single1.fastq` and `.single2.fastq` suffixes) are also recognized.
fastq_nodup optional parameters `[options]`:

```
[-h|--help] [--version] [-v|--verbose] [-z|--gzip]
[--prefix=<string>] [--no_prefix] [-l|--index_len=<u>]
[-m|--match_len=<u>] [-d|--index_err=<u>] [-e|--match_err=<u>]
[--seed=<n>]
```

`-h`
`--help`

Print a guide to valid command line parameters and their correct usage to the terminal, and then exit.

`--version`

Print the version number of the executing program and then exit.

`-v`
`--verbose`

Print additional statistics and diagnostic information about the run.

`-z`
`--gzip`

Write output `.fastq` files compressed in the gzip format, and with filenames suffixed `.gz`.

`--prefix=<string>`

Specify the prefix string to add to read identifiers. For example:

```
@xx+<string>:1_68_381/1
```

Note, this will replace any existing prefix. By default this is `<out_prefix>` described above.

`-n`
`--no_prefix`

Overrides the default read prefixing behavior and instead preserves the read identifiers exactly as they are in the input fastq file(s).

`-l <u>`
`--index_len=<u>`

Controls the read sequence prefix length of the search index. This should be set long enough so that random prefix collisions are infrequent, but short enough so that a large number of sequence errors are unlikely to accumulate. This parameter also impacts the memory use of this tool, with each increment increasing the index size by a factor of 4. This parameter defaults to `<u>`=14 nucleotides and may not exceed 32 nucleotides.

Both mates of each read-pair are indexed, and this indexed look-up is used to find a list of mate sequences corresponding to each sequence prefix. Given that the mates also must match, it is acceptable for a low-level of prefix collisions to occur in this index. That is, assuming the level of prefix collision is low (only a small number of non-duplicated reads share any given prefix), it will be highly improbable that reads matching a given prefix will share mates with matching sequence at random.

`-m <u>`
`--match_len=<u>`

Controls the required match length of the unindexed mate in detecting a duplicate pair. Like the previous parameter (`--index_len`), this setting has an impact on memory use, although memory requirements only increase linearly with match length. This parameter should be set as high as memory permits to improve specificity.

This parameter defaults to the full read length, although lower settings are acceptable assuming that the `--index_len` is appropriately set, and the sum of `index_len` and `match_len` exceeds about 35 nucleotides (that is, random 35 nucleotide joint index-match collisions are extremely unlikely assuming each match is sufficiently long).

`-d <u>`
`--index_err=<u>`

Controls the number of mismatches tolerated in the indexed sequence prefix while still permitting a sequence match. This value defaults to `<u>`=1 and may be set in the range 0–2. This setting has a major impact on run time, with each increment increasing it by a factor of about 4. Duplicated reads with more than `index_err` mismatches in the `index_len` prefix may still be found by the mate's indexed prefix, assuming that it does not also have more than `index_err` errors.

`-e <u>`
`--match_err=<u>`

Controls the number of mismatches tolerated in a matched mate sequence (in the `match_len` prefix). This value defaults to 3 and must be $\geq 0$. This setting has little impact on run time. Setting `match_err` to higher values will reduce the specificity of the mate-matching and should therefore be accompanied by correspondingly larger values of `match_len`.

`--seed=<n>`

Integer seed used by the internal pseudo-random number generator. This only affects the random selection of matching singlets (as described above). Defaults to `<n>`=12345.

## 3.3 samplefastq

This tool randomly samples reads from a given set of fastq files, producing a corresponding set of output fastq files. This is useful for more quickly tuning parameters for the `trimfastq` tool (described below) and for producing simulated read datasets for various other purposes.

Usage: `samplefastq -f <float>|<int> [options] <in_prefix> <out_prefix>`

where the mandatory `-f` parameter indicates how the input files should be sampled and `<in_prefix>` and `<out_prefix>` specify the input and output filename prefixes to use for input files and naming output files. For example:

`<in_prefix>.read1.fastq` → `<out_prefix>.read1.fastq`
`<in_prefix>.read2.fastq` → `<out_prefix>.read2.fastq`
`<in_prefix>.single.fastq` → `<out_prefix>.single.fastq`

The input files may be gzip format compressed files with the customary `.gz` suffix at the very end of the filename, which tells `solid2fastq` to decompress the files as it reads them. Multiple single files (i.e. `.single1.fastq` and `.single2.fastq` suffixes) are also recognized.

Note that for paired sequences (as shown above) read pairs are always sampled together; that is, each pair of reads counts at a single read unit for sampling purposes, as does each singlet read.

`samplefastq` manditory parameter `-f`:

`-f <float>` or `<int>`
`--frac_sample=<float>` or `<int>`

The `-f` parameter tells `samplefastq`, either directly or indirectly, how many output reads to randomly sample from the input `.fastq` files. Values $< 1.0$ are interpreted as a fraction of the input reads to retain, whereas values $\geq 1.0$ are rounded to an integer count of reads to retain.

Note that when a read count is provided, the input files must be read twice by `samplefastq`; once to count the number of input reads, and then again to randomly sample those reads. Once the number of input reads is known, the requested read count is converted into a fraction and operation proceeds as though that fraction had been initially provided to the `-f` parameter. Therefore, if a given set of inputs is to be sampled repeated, it will be considerably more efficient to obtain the count of reads once and calculate the fraction to retain for each subsequent sampling. The requested (or calculated) fraction of reads to sample is used internally as the probability of retaining each read (or pair) and each is considered an independent trial. Because of this, the precise fraction (or count) of reads requested is unlikely to be generated, although the sampled output will typically be very close to that requested.

`samplefastq` optional parameters `[options]`:

`[-h|--help] [--version] [--seed=<int>] [-z|--gzip]`

`-h`
`--help`

Print a guide to valid command line parameters and their correct usage to the terminal, and then exit.

`--version`

Print the version number of the executing program and then exit.

`-z`
`--gzip`

Write output fastq files compressed in the gzip format, and with filenames suffixed `.gz`.

`--seed=<n>`

Integer seed used by the internal pseudo-random number generator. To obtain different samples from the same input file(s), set this parameter to different values. Repeated runs with all parameters the same will produce identical outputs. Defaults to `<n>`=12345.

## 3.4 trimfastq

This tool is used to trim or reject individual reads on the basis of quality scores. Reads may also be rejected based on low information content (entropy). When run on paired reads, this operation is performed on mates independently, and when one mate of a pair is rejected, the other becomes a singlet.

Trimming based on quality scores is performed with the goal of producing trimmed reads that have a calculated probability of error below some threshold. That is, bases are trimmed off the 3' end of the read until the remaining bases have a joint probability of error below a threshold. For example, if a threshold of 0.5 is selected, that means the output reads will be expected to have, on average, 0.5 erroneous nucleotides per read.

For colorspace (SOLiD differentially encoded) reads, the probability of error is calculated as the joint probability of error in each of two consecutive color positions. That is, because single color errors will typically be correctable, the probability of a nucleotide error at a given position is the probability that both corresponding color positions are erroneous (i.e. the pairwise product of color error probabilities yields the probability of uncorrectable nucleotide errors).

Minimum read lengths may also specified such that reads trimmed shorter than the specified minimum to meet an error probability threshold will instead be rejected.

Reads may also be independently rejected for failing to have a minimum mean information content. `trimfastq` optionally calculates the mean entropy of each read (as average bits per dinucleotide) and reject reads that fail to meet a given threshold. A common error for next-generation sequencers is to produce junk reads where all or part of the read contains highly repetitive (low information) sequence. Random DNA sequence contains 4 bits per dinucleotide of entropy, and we have observed that (non-repeat) natural sequences typically contain more than 3 bits per dinucleotide.

Usage: `trimfastq [options] <in_prefix> <out_prefix>`

where `<in_prefix>` and `<out_prefix>` specify the input and output filename prefixes to use for input files and naming output files. For example:

```
<in_prefix>.read1.fastq → <out_prefix>.read1.fastq
<in_prefix>.read2.fastq → <out_prefix>.read2.fastq
<in_prefix>.single.fastq → <out_prefix>.single.fastq
```

The input files may be gzip format compressed files with the customary `.gz` suffix at the very end of the filename, which tells `solid2fastq` to decompress the files as it reads them. Multiple single files (i.e. `.single1.fastq` and `.single2.fastq` suffixes) are also recognized.

`trimfastq` optional parameters `[options]`:

```
[-h|--help] [--version] [-c|--color_space] [-z|--gzip]
[-v|--invert_singles] [-s|--singles]  [-p|--correct_prob=<d>]
[-l|--min_read_len=<u>] [-m|--min_mate_len=<u>]
[-f|--fixed_len=<u>] [--prefix=<string>] [--add_len]
[--no_prefix] [-e|--entropy_filter=<d>] [--entropy_strict]
[--mates_file]
```

`-h`
`--help`

Print a guide to valid command line parameters and their correct usage to the terminal, and then exit.

`--version`

Print the version number of the executing program and then exit.

`-z`
`--gzip`

Write output fastq files compressed in the gzip format, and with filenames suffixed `.gz`.

`--prefix=<string>`

Specify the prefix string to add to read identifiers. For example:

```
@xx+<string>:1_68_381/1
```

Note, this will replace any existing prefix. By default this is `<out_prefix>` described above.

`-n`
`--no_prefix`

Overrides the default read prefixing behavior and instead preserves the read identifiers exactly as they are in the input fastq file(s).

`--add_len`

Add the final trimmed length value to the read prefix. For example:

```
@TT+lambda:1_81_912/1
TGTTGTGCGCGCCATAGCGAGGGGCTCAGCACGCGTCCCTCCGCCCCAC
+
5/0)358)%57*)%881/(/4'.'53*&#91*-'&,&%$''%&'46-+#
```

Trims to (where `37` in the first line indicates the final trimmed length) :

```
@TT+37|lambda_trim:1_81_912/1
TGTTGTGCGCGCCATAGCGAGGGGCTCAGCACGCGTC
+
5/0)358)%57*)%881/(/4'.'53*&#91*-'&,&
```

`-c`
`--color_space`

Use color-space errors to trim and reject reads. This overrides the preferred default behavior of using uncorrectable nucleotide space errors for trimming colorspace reads. However, for trimming non-colorspace reads (e.g. Illumina reads) this parameter should be used, as error correction through differential encoding is not possible with such reads.

`-v`
`--invert_singles`

Causes singlet file(s) output to be the opposite configuration of the input. That is:

`<in_prefix>.single.fastq` → `<out_prefix>.single[12].fastq`

or

`<in_prefix>.single[12].fastq` → `<out_prefix>.single.fastq`

Note that this option is only valid when there are input singlet reads.

`-s`
`--singles`

Write two singlet files (`.single1.fastq` and `single2.fastq`), one for new singlets generated from each paired input file. Note that this option is only valid when there are no input singlet reads. The default behavior in this case is to write a single combined singlet file (`.single.fastq`).

`-p <d>`
`--correct_prob=<d>`

Minimum mean probability that trimmed output reads are free of uncorrectable errors (or all errors with `-c`). Default value is `<d>`=0.5.

`-l <u>`
`--min_read_len=<u>`

Minimum length of a singlet or longest-mate of a pair, in nucleotides. Default value is `<u>`=24 bases.

`-m <u>`
`--min_mate_len=<u>`

Minimum length of the shortest mate of a pair, in nucleotides. Default value is the value provided for `--min_read_len`. This value must be $\leq$ the value used for `--min_read_len`. Use of this parameter allows paired reads shorter than `min_read_len` to be retained as long as their mate satisfies `min_read_len`.

`-f <u>`
`--fixed_len=<u>`

Trim all reads to a fixed length, still rejecting reads that don't meet specified quality thresholds at this length. Default is no fixed length.

`-e <d>`
`--entropy_filter=<d>`

Reject reads with mean per position measured information (entropy) below the given value (in bits per dinucleotide). The range of valid values is 0.0–4.0 inclusive. By default this filter is off.

`--entropy_strict`

Reject reads for low entropy overall, not just the retained part after trimming. By default this setting is off. Use of this setting requires use of `--entropy_filter`.

`--mates_file`

In addition to other outputs, produce a Velvet compatible mate-paired output file (named `<out_prefix>_mates.fastq`) with read2 mates reversed as required.

## 4 Colorspace FASTQ Format Reference

The SEAStAR tools use certain conventions for organizing read data in `.fastq` files. This section describes these conventions in sufficient detail to understand the operation of the tools described in this document.

SOLiD reads and associated quality statistics from the instrument are packaged in FASTA-like format files called `.csfasta` and `.qual` files. Fragment (non-paired) libraries produce a single matched set of files named by convention as:

`<prefix>_F3.csfasta` and `<prefix>_F3_QV.qual`

Paired libraries (either mate-paired or paired-end) additionally produce a second matched set of files named:

`<prefix>_R3.csfasta` and `<prefix>_R3_QV.qual`

For some paired-end libraries, the `R3` in these (second mate) files may be replaced by `F5-BC` or `F5-P2`.

There is no guarantee that every read in an `F3` file has a corresponding mate in an `R3` file, or vice-versa. These unmated reads are called “singlets”.

Most open source tools do not support these off-instrument SOLiD FASTA-like files, and so they must be converted to FASTQ format files for compatibility. This is complicated by the fact that SOLiD reads do not represent nucleotides, but are instead differential encoded colorspace reads, where each numbered color represents a coded di-nucleotide transition[10] (e.g. `AA` = `0`, `AC` = `1`, etc.). For colorspace reads to be translated into nucleotide space, an initial known base, the primer base, is required. The following example shows the colorspace sequence for one read prefixed with primer base “`T`”:

```
>1_68_381_F3
T01200011211220011021332321220121211221111322011220
```

One simple approach to using these reads would be to simply convert them to nucleotide space `.fastq` files and from that point onwards ignore the colorspace source of the reads. Unfortunately, this does not work well because colorspace errors at any position in a read will effectively corrupt all of the resulting converted nucleotide sequence from the erroneous position(s) onwards. In addition, there are distinct potential advantages to processing (aligning and assembling) SOLiD reads in colorspace, delaying conversion to nucleotide space until later in the analysis pipeline when it can be done much more accurately (due to the error correction potential of the di-nucleotide encoding). For these reasons, the SEAStAR tools preserve the colorspace information in the FASTQ representation.

To accomplish this, we follow the convention used by MAQ, BWA and Velvet, mapping colorspace numbers to colorspace letters as `[0123.]` → `[ACGTN]`. Because the initial primer base and its following color value are not from the sampled DNA, and thus will not correspond to the actual sequence ~75% of the time, these two values need to be removed from the colorspace read data that is actually used for alignment or assembly. However, they need to be kept in some form so that the original nucleotide sequence of the read can be reconstructed at a later stage in the pipeline. To handle this, we have developed the following convention:

Original colorspace `.csfasta` read:

```
>1_68_381_F3
T0120001121122001102133232122012121122111113220..2.0
```

Convert to colorspace letters:

```
>1_68_381_F3
TACGAAACCGCCGGAACCAGCTTGTGCGGACGCGCCGGCCCCTGGANNGNA
```

Convert to FASTQ format (with quality information added), relocating the primer base and following color “`TA`” to the header line (discarding the initial quality value):

```
@TA+lambda:1_68_381/1
CGAAACCGCCGGAACCAGCTTGTGCGGACGCGCCGGCCCCTGGACCGGA
+
9’4,)9,*%4/’#,32)*.1-1(5&3,%,$:*&#’8,+2’40$)05(,&
```

[10]See SOLiD colorspace whitepaper for more details:
http://www3.appliedbiosystems.com/cms/groups/mcb_marketing/documents/generaldocuments/cms_058265.pdf

The read prefix `lambda` is added as an identifier for this set of reads, by default, from the `.csfasta` filename prefix. The read name suffix changes from the SOLiD specific `F3` to `/1` designating it as the first mate. Similarly for mated reads, `R3` (or `F5-BC` / `F5-P2`) suffixes are changed to `/2`, designating them as the second mates.

The resulting `.fastq` files may then be processed though a variety of quality control tools that modify (e.g. trim, error correct) and discard reads. Trimming can result in reads that vary in length, and since variable read length information is not always preserved through some downstream analyses (e.g. some alignment outputs), it is occasionally desirable to (optionally) preserve this information in the read header as well, where the "`49`" indicates the actual number of color encoded nucleotides:

```
@TA+49|lambda:1_68_381/1
CGAAACCGCCGGAACCAGCTTGTGCGGACGCGCCGGCCCCTGGACCGGA
+
9'4,)9,*%4/'#,32)*.1-1(5&3,%,$:*&#'8,+2'40$)05(,&
```

Reads conforming to these FASTQ conventions are used (and assumed) throughout the SEAStAR analysis pipeline, and any additional tools (custom scripts, etc.) that modify FASTQ read data will need to preserve these conventions in their outputs.