ngsShoRT 2.0 manual

Sari Khaleel (Sari.S.Khaleel.DM AT dartmouth.edu)
Dartmouth Medical School
Last updated 05-06-2013

Table of Contents:

I. Problem Definition:

General Problem:
Take a pair of paired-end (PE) fastq files (SRR*_1.fastq and SRR*_2.fastq) OR a single read from a single reads file (SRR.fastq) and trim it using several available methods applied in a certain sequence.

Complications:

1. De Bruijn Graph assemblers like Velvet, SOAPdenovo and ABySS break reads into small K-long words (K-mers). Reads shorter than K are not used for assembly. So, ngsShoRT discards any read that has been rendered too short by trimming by having a min_read_length cutoff that is specified by the user (default is 21).

2. If you intend to assemble trimmed reads at different K-values, set min_read_length to one base longer than the biggest K-value used for assembly. So, if you're assembling at K = (21, 31, 41), set min_rl to 42.

3. **A read is "good"** if its length is >= min_rl **AND** it was not filtered out by the following read-trimming methods: lqr, nperc, ncutoff, 5adpt (kr), qseqB (kr), and qseq0. Some non-filtering methods (see 5) will also occassionally trim out reads.

4. TERA, 3end, 5end, Mott are base-by-base trimming methods that we have designed to stop trimming a read if its length = min_rl. This ensures that reads trimmed by these methods won't become too short for assembly.

5. However, some methods can still trim reads shorter than min_rl : adpt (ka), qseqB (ka), and nsplit. This is intentional because a read where the majority of bases are adapter, B-scored, or N-bases (respectively to the methods) are highly erroneous.

6. For PE-read pairs, trimming can render only one read in the pair bad (by trimming or filtering). The pair is fully removed from the final trimmed output files, but the good (surviving/widowed) read in the pair is saved separately in a surviving_SE_mates.fastq file. This file can be co-assembled with the shuffled PE files by Velvet:

```
./velveth output_directory hash_length -fastq -shortPaired shuffled_trimmed_PE_file
 -fastq -short surviving_SE_mates.fastq
```

II. Trimming Methods algorithms:

Whether used for paired-end or single reads, the actual trimming methods are designed to work on ONE read at a time. PE reads trimming methods will first apply single read trimming (using the methods below) to each read separately, then handle the trimmed reads as a pair using PE_ trimming modules.

Quality trimming methods include *lqr* (removal of low quality reads)*, TERA* (trimming low quality 3'-end bases based on their average running quality score), and *mott* (Modified R. Mott trimming of reads)*,* and use quality score data (i.e., they convert the ASCII quality score to integer values) to trim reads. qseqB is a special case of quality-trimming where B-scored bases are removed. Unlike the previously mentioned methods, it doesn't really calculate quality scores and can only work if the quality score mapping was Illumina's.

Non-quality trimming modules include 3end, 5end, nsplit, nperc, 5adpt, and qseq0. 3end and 5end simply remove a specific number of bases from the 3' and 5' end of reads, respectively. In contrast, nsplit, nperc and 5adpt examine the alien bases (Ns, adapter sequences) in the sequence line to trim reads. Qseq0 is a special case that works only for qseq files. It removes reads whose filtering flag was 0 (i.e., they did not pass filtering during Illumina sequencing analysis).

Non-trimming methods include i2s and s2i, which allow switching the quality scoring of reads from Illumina to Sanger or from Sanger to Illumina, respectively.

Below is a discussion of five of the more complicated methods and their algorithms. These methods and the other less complex methods are also discussed in terms of usage and implementation in section **IV.B.3 (Under the Hood: Trimming Modules)**

1. TERA : (trim by the) Three End Running Average quality score

Reasoning: Illumina reads generally have lower base-call quality towards the 3'-end of reads. Instead of trimming a specific number of reads from the 3'-end (which our trimmer allows through max_rl INT), or trimming 3'-end bases under a specific cutoff, we prefer the running average of quality scores because it adapts better to quality scores, and is not affected by outliers, as seen in the example below where trimming is unaffected by the appearance of relatively high-quality bases in the middle of very low quality bases.

Concept: Trim bases from the 3' end of a read, based on the running average quality score, RAQS, of its bases. Starting at the last (the most 3') base of the read, begin counting RAQs of all bases until reaching a base X where RAQS exceeds a cutoff value specified by -tera_avg. If X's 5' index is < min_rl, it's set to min_rl (see above section to understand why we do this). All bases 3' to X-index are trimmed out.

Example:

```
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNGCTACTNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
@@@@@@@@@@@@BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBHHHHHH@@@@@@@@B@BBB@BBBBB@@BBBBB@@@BB@@@BB

(where B=2, @ = 0, H=8 in Illumina scoring).
```

at 3'-end avg =2 and min_read_length = 31 is trimmed down to min_read_length because even though there was a substring of called bases with H (=8) quality score, the running average quality score for this string (`HHHHHH@@@@@@@@B@BBB@BBBBB@@BBBBB@@@BB@@@BB`) was 1.9, which is <=2. So, we end up with the following string:

```
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
@@@@@@@@@@@@BBBBBBBBBBBBBBBBBBB
```

The above example also explains why TERA is not enough as a quality trimming procedure, since it's limited to min_read_length, and will therefore result in this bad read instead of completely trimming it out. There are two ways to handle this case: (1) set the min_read_length to zero, thus allowing TERA to trim entire reads. This is not a good idea because min_read_length ensures that read lengths are >= K-mer length (of DBG assembly), and are thus all usable by the DBG assembler. Our experiments show that setting min_read_length to zero resulted in losing too many bases AND rendered many read pairs too short to be used by the assembler (because their lengths are < K-mer length).
The better approach is (2), which is to precede 3'-end trimming with a step that removes entire bad reads (nperc or lqperc). We present our recommended order of applying trimming methods in IV.1.

Notes:

A special use for this method is the removal of B-quality bases in Illumina reads. For Illumina 1.3+ data a quality score of B id not equal to phred score of 2; it just means that the base call was unreliable/erroneous [2]. 3end with 3end_avg of 2 will trim off only sequences of bases with B scores (and lower), as seen in the example below:

```
GATACGGCGACCACCGAGATCTACACTCTTTCCCTACACGACGCTCTTCCGATCT
'OOOO@[Y[ZY_```\_Y___`_``_BBBBBBBBBBBBBBBBBBBBBBBBBBBBB
```

where at min_read_length = 20 and 3end_avg =2, 3'-end trimming results in:

```
GATACGGCGACCACCGAGATCTACAC
'OOOO@[Y[ZY_```\_Y___`_``_
```

2. 5'-adapter trimming (5adpt):

Concept: Trim adapter sequences from reads. To be more specific, trim out the matched adapter sequences (along with some user-specified number of bases before and after it), then do one of three actions to the read itself (see below).

- -5a_f: Adapter sequences are listed in a txt file, such that each sequence is on a separate line (so the lines are separated by carriage returns that can be removed by PERL's chomp). There are adapter sequences for read_1, and different (complementary) adapter sequences for read_2.
  Adapter_sequences are loaded from the adapter_sequences file, which can be either the default (Illumina) sequences file, or a user-created list that uses our format (or create a modified copy of our five_prime_adapter_seq_TEMPLATE.txt -- see VII). This format allows for deletion of bases before (5' to) or after (3' to) the matched substring.

- -5a_mp: Match percentage is specified by the user.
- -fmi : The furthest matching index (i.e., how far into the read should the script be searching for adapter sequences). We recommend setting it to raw_read_length – 10. Read_length obviously depends on your PE file and their read lengths.

- Approximate (Fuzzy) matching is used to match the adapter sequence to the read in a 5'-3'direction.
- If an adapter sequence is matched, the method will:
  1. Remove the detected adapter sequence and then remove a specific number of bases before and after it (the number of before and after bases is specified in the adapter_sequences file –see VII),
  2. Do one of two actions depending on the value of -5a_axn:
     - kr: **K**ill the entire **r**ead
     - ka: **k**ill the detected adapter sequence and all bases **a**fter it

For details on the used adapter sequences and how we apply adapter trimming, see VIII.

3. nsplit

Reasoning: we "split" reads instead of simply deleting the N-block then **merging** the read's pieces is that this would violate the actual sequence order and spacing between bases. Splitting the read into two daughter reads maintains structural information.
But, Why not delete/split around all N-bases? Initially, we wanted *nsplit* to remove all Ns and split reads around them recursively until no N bases are left. The result was splitting the reads into too many, very small daughter reads that were not used for assembly because they were shorter than K-mer length used for DBG assembly. So, although assemblers like velvet may convert (N) bases to (A)s, it's a problem that we have to accept.

Concept:

1. Search the read for substrings of consecutive uncalled bases (N, n, .), which we call Nblocks, whose length >= min_Nblock_l (min_Nblock_l is a user-specified cutoff).
2. Split the read into around the longest of the blocks. If all blocks have the same length, split at the leftmost block.
3. Delete the parent read, and add daughter reads (as long as their length is >= min_read_length) to the trimmed file, paired with the mate (or its own split daughter reads) of their parent read. If both parent reads were split, shuffle their daughter reads (see below an example of Nsplitting and all its possible cases in ngsShoRT).

Note : nsplitting is a good, but insufficient method for deleting ambiguous (high N-content) reads:

In the following read, there are two N-blocks. The leftmost one is longer.
```
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNGCTACTNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
@@@@@@@@@@@@BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBHHHHHH@@@@@@@@B@BBB@BBBBB@@BBBBB@@@BB@@@BB
```
So, the read is split to an empty string (the left of the N-block), and this:

```
GCTACTNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
HHHHHH@@@@@@@@B@BBB@BBBBB@@BBBBB@@@BB@@@BB
```

Nsplitting in ngsShoRT

Quick recap: The purpose of N-splitting is to remove sequences of uncalled bases in a read. The sequence of uncalled bases (usually represented by N, n, .) must be longer than a specified cutoff. We call this an Nblock

Let's call the right read in a read pair Read_1 and the left read Read_2.

```
Read_1   ==========NNNNNN<<<<<<<<<<NN<<<<<
Read_2   ----------------------NNNNNNN~~~~
```

We cannot simply remove the Nblocks in the example above and merge the read's left and right sides because this will change the actual sequence structure. So, we split the read into two pieces, the piece to the right and the piece to the left of the Nblock. Like reads, a piece is "good" if it's longer than min_read_length. Each 'good' piece is used to form a new daughter read, and is paired with its parent reads' mate (or its daughter reads).

We realize that there can be more than one block of Ns in the read. Splitting a read at more than one block will almost certainly produce a set of pieces that are all smaller than min_read_length. So, it's better to split at just one block if more than one block is detected. Currently, the method detects and splits at only the longest (or leftmost if max lengths are equal) Nblock.

<u>For PE reads, the algorithm works as follows:</u>

1. *Search for target Nblock in read_1. If found, split read_1 and get the left and right piece. Else, keep the read as it is.*
2. *Search for target Nblock in read_2. If found, split read_2 and get the left and right piece. Else, keep the read as it is.*
3. *For each good piece in read_1, pair it with a good piece in read_2*

So, for our example,

1. Read_1 is broken to two pieces,

```
R1-L   ==========                    R1-R   <<<<<<<<<<NN<<<<<
```

2. Read_2 is broken to two pieces

```
R2-L   ----------------------        R2-R   ~~~~
```

(Note that R2-R is not a good piece)

3. For each good piece in read_1, pair it with a good piece in read_2. This produces a new set of paired reads:

```
R1-L   ==========
R2-L   ----------------------

R1-R   <<<<<<<<<<NN<<<<<
R2-L   ----------------------
```

<u>Note:</u>
In the above example, if Read_1 was the same, but the Nblock in Read_2 was at the end of the read;

```
Read_1   ==========NNNNNN<<<<<<<<<<NN<<<<<
Read_2   --------------------------NNNNNNN
```

Then Nsplitting of Read_1 would give us a left piece and a right piece that's an empty string, whose length (0) is < min_read_length, and is thus a "bad" piece.

Our new paired read set of shuffled daughter reads would be similar to the set from the above example

```
R1-L   ==========
R2-L   --------------------------

R1-R   <<<<<<<<<<NN<<<<<
R2-L   --------------------------
```

4. lqr

<u>Reasoning:</u> remove low-quality reads from the PE files.

<u>Concept:</u>
Given a user-specified Low quality score cutoff (--lqs) and a percentage cutoff for bases whose quality score is <= lqs, which we call lq_p.
- Count the number of bases whose qual score is <=lq. Let's call these LQ bases.
- Label the read "bad" if the percentage of LQ bases. If it's >= LQ_perc_cutoff, "good" otherwise.

5. mott

Reasoning: extract the highest-quality string of bases from the read. In other words, trim out low-quality 5' and 3' bases from the read.

Use the Richard-Mott algorithm (from CLC's BioGenomics Workbench) to trim reads.
This method is useful for trimming reads from both the 5' and 3' end to get a sequence with maximal running sum quality score.
The Richard-Mott trimming algorithm is described as follows in CLC's manual:

<u>The algorithm:</u>

```
1. For every base, convert its quality score, Q, to its corresponding Pe (Perror). Pe = 10 ^(-Q/10)
    So [Q= 0 --> Pe = 1], [Q=2 --> Pe = 0.6], [Q= 10 --> Pe = 0.1], [Q=20 --> Pe = 0.01], [Q=30 --> Pe =
0.001]
2. For every base, calculate its LmP value, which equals (Limit - Pe).
3. For every base (starting from the 3' end for short reads), add its LmP value to a running sum. If the sum
drops below zero, set it to zero.
4. When done with the entire sequence, retain the part of the sequence between 1st positive running sum and
the highest value of the running sum.
```

<u>How to choose limit value:</u>

# LmP = mott_limit - Perror[base], where Perror[base] = 10 ^(-Q/10), where Q = quality score
# So, when LmP = 0, mott_limit = 10^(-Q/10)

# Q_cuotff = 0 => mott_limit = 1
# Q_cuotff = 2 => mott_limit = 0.631 (Default value)
# Q_cutoff = 4 => mott_limit = 0.398

Sample results:
Note : the follow sequences are Illumina-socred, so B = qual score of 2, @ = qual score of zero

At Mott_limit = 1

The string
```
'@@@@@@@@@@@@BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB@@@@@@@@B@BBB@BBBBB@@BBBBB@@@BB@@@BB'
```
Is trimmed to
```
            'BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB@@@@@@@@B@BBB@BBBBB@@BBBBB@@@BB@@@BB'
```

And the string
```
'OOOO@[Y[ZY_```\_Y___`_``_B@@BB@BB@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@'
```
Is trimmed to
```
'OOOO@[Y[ZY_```\_Y___`_``_B@@BB@BB'
```

So for the Mott trimming algorithm, the algorithm will skip all bases for which the running_LmP sum adds to zero, and starts at the first point where running_sum_LmP is > 0. **This is good, except it can trim a read to a size < min_read_length**, which means (if the read length is < K-mer for DBG assembly) that it won't be used.

Since the length of the retained string by the above algorithm will not necessarily be >= min_read_length. So, we did a simple modification to the algortithm: if the running sum of LmP values is still zero at the base whose index is min_read_length , stop right there and let the retained string be the substring from base #0 to base #min_read_length (indexing from the 5' end) So,

```
The MODIFIED Mott algorithm to keep the length of the trimmed piece at >= min_read_length

1. For every base, convert its quality score, Q, to its corresponding Pe (Perror). Pe = 10 ^(-Q/10)
   So [Q= 0 --> Pe = 1], [Q=2 --> Pe = 0.6], [Q= 10 --> Pe = 0.1], [Q=20 --> Pe = 0.01], [Q=30 --> Pe =
0.001]
2. For every base, calculate its LmP value, which equals (Limit - Pe).
3. For every base (starting from the 3' end for short reads), add its LmP value to a running sum. If the sum
drops below zero, set it to zero, AND:
        → IF THE 5' INDEX OF THIS BASE = MIN_READ_LENGTH, stop here and return the substring from base #0 to
         base #min_read_length
4. When done with the entire sequence, retain the part of the sequence between 1st positive running sum and
the highest value of the running sum.
```

<u>How to choose limit value:</u>
At limit = 1 , LmP will be -ve for only bases with Q < 0

At limit = 0.6, LmP will be -ve for only bases with Q < 2

So, if @= 0 and B= 2 qual scores (Illumina scoring), and min_read_length = 40, Using Mott limit = 1 will give us :

```
1. Using the original Mott algorithm
Before:
'@@@@@@@@@@@@BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB@@@@@@@@B@BBB@BBBBB@@BBBBB@@@BB@@@BB'
After                                                                                                 :
'BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB@@@@@@@@B@BBB@BBBBB@@BBBBB@@@BB@@@BB'
Before:
'OOOO@[Y[ZY_```\_Y___`_``_B@@BB@BB@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@'
After : 'OOOO@[Y[ZY_```\_Y___`_``_B@@BB@BB'

2. Using the MODIFIED Mott algorithm
Before:
'@@@@@@@@@@@@BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB@@@@@@@@B@BBB@BBBBB@@BBBBB@@@BB@@@BB'
After                                                                                                 :
'BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB@@@@@@@@B@BBB@BBBBB@@BBBBB@@@BB@@@BB'
Before:
'OOOO@[Y[ZY_```\_Y___`_``_B@@BB@BB@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@'
After : 'OOOO@[Y[ZY_```\_Y___`_``_B@@BB@BB@@@@@@@' <-- note how Mott stopped at base #40 (min_read_length).
```

Other methods

5end, 3end, qseq0, qseqB, nperc, ncutoff, rmHP: see their descriptions in section IV.B.3

III. Recommended Sequence and parameters of Trimming Methods

The user can specify the methods to be used in any sequence they want.
If all methods were to be applied, our recommended sequences are:

For QSEQ Reads: qseq0_qseqB_nperc_lqr_5adpt_nsplit_tera
For ALL Reads: nperc_lqr_5adpt_nsplit_tera

- Note that Mott can be used instead of TERA. Mott removes low quality 5' and 3' bases, TERA removes only the 3'
- If you know that the first N 5' bases for all reads have low quality, add 5end before nperc, and set -n5 to N
- If you know that the last N 3' bases for all reads have low quality, add 3end before nperc, and set -n3 to N

1. Reasoning

We found that Illumina datasets generally start and end with reads that have very low quality: almost all bases are uncalled (N) and/or have a quality score of B, which denotes an error and/or a very low quality scores (down to 0, which means 100% error rate). Such low quality reads need to be completely removed instead of applying computationally-expensive trimming methods to them.
3'-end and Mott trim cannot completely remove such reads as they are limited to min_read_length (see II.1 and VI), and so it's better (and much faster) to start with *nperc*, followed by *lqr*. *nperc* will remove ambiguous reads that have too many uncalled bases (over NPerc%). Next, *lqr* will remove low quality reads, where a read is labeled "bad" if it has over lq_p% of lqs bases (lq_p and lqs are specified by the user). Now, the reads are ready for trimming out adapter sequences.

The next step is to reduce the number of N-bases inside the reads by using *nsplit* to split the reads around the largest N-block (whose length is >= a user-specified cutoff). Finally, *TERA* or *mott* can be applied to the trimmed reads to trim them even further. While *TERA* removes only low quality bases at the end of a read, *mott* will try to trim out low quality bases from both ends.

2. Recommended/Default parameter values

The best way to determine optimal parameter values for dataset is trial and error: start with some low parameters (low quality score cutoffs, high *nperc* and *lqr* cutoffs), and then read the final_PE_report.txt/ final_SR_report.txt file to see how many bases, reads, and read pairs were trimmed out by the used set of methods and parameter values.

Another good way to determine the amount of trimming needed is to produce a FASTQC report (using software from http://www.bioinformatics.bbsrc.ac.uk/projects/fastqc/) for your dataset. The report will show a lot of useful data (quality score distribution over read length, Kmer content, etc), and can be also used to assess the improvement of quality scores in trimmed data.

Here's a set of methods and their parameter values that have generally resulted in improving our dataset quality scores and their assembly N50 and coverage measures (using K <= half read length with velvet 1.1.04, SOAPdenovo 1.05, and CLC 3.2):
nperc: --nperc_p 70
lqperc: --lq_p 70, --lqs 4
5adpt: --5a_mp 100, --fmi (raw read length – 12), --5a_axn ka
nsplit: --nsplit_len 5
TERA: --tera_avg 2 ← In Illumina 1.4+ scoring, B denotes a quality score of 2 and/or an error in sequencing.
Mott: --mott_lim 0.6 (see section VI).

IV. Under the hood

IV.A. The single_fQ_read and PE_fQ_pair objects:

Although this trimmer is used for paired-end reads or single_reads file, its basic trimming methods are designed to work on ONE read at a time.
Reads are read from a FastQ or a QseQ file and are used to create a single_fQ_read object. You can see the code of this class and its perldoc documentation at (`nsgShoRT_1.0/modules_and_classes/read_classes/single_fQ_read.pm`)

The `single_fQ_read` class has the following components

`_header`	corresponds to the header line of a fastQ read, or parts [0]:[2]:[3]:[4]:[5]#[6]/[7] of the qseq line
`_seq`	corresponds to the sequence line of a fastQ read, or part [8] of the qseq line
`_comment`	corresponds to the comments line of a fastQ read
`_qual`	corresponds to the sequence line of a fastQ read, or part [9] of the qseq line
`_qseq_filter`	corresponds to the filtering flag of a qseq read (part [10] of the qseq line)
`_skip`	the "skip this read" flag. Each read is initialized with a value of 0 (do not skip), but read-filtering methods (lqr, nperc, qseq0, qseqB, 5adpt, ncutoff) will set the flag to 1 if a read fails their test
`_ascii_zero`	the ascii character number that's equivalent to phred score of 0 for this read's sequence line
`_min_rl`	the minimum allowed length for this read

The most important methods of `single_fQ_read` are

`skip()`	set the aforementioned `_skip` flag to 1. Used by read-filtering methods (see above)
`get_length()`	get the length of the read's `_seq` line
`is_good()`	returns the value of `((get_length()>= min_rl) && (_skip != 1))`
`switch_scores(x2y)`	switch the ascii-phred mapping from x to y. x2y is either s2i (Sanger to Illumina) or i2s (Illumina to Sanger).

By now, I hope it's clear how easier trimming becomes when the trimming subject is always a single_fQ_read object that can hold the information of both qseq and fastq reads, as well as having a large set of methods.

The `PE_fQ_pair` class is created from two `single_fQ_read` objects, and allows access to its read objects (and their functions) using the methods `read1()` and `read2()`. For example: `PE_fQ_pair->read1->is_good`
Aside from `read1()` and `read2()`, the most important method of `PE_fQ_pair` is `get_status()`. This method returns 4 possible values depending on the pair's reads:
3 : both reads are good
2 : only read2 is good
1 : only read1 is good
0 : neither of the reads are good.

Trimming subroutines are divided into single_read trimming subroutines, which do the actual work (applying the algorithm to the sequence/qual lines, etc) on a single `single_fQ_read` object; and PE_ trimming subroutines, which take an `PE_fQ_pair` object and send each of its reads through the single_read trimming subroutine.

IV.B. Basic program structure and modules:

IV.B.1. ngsShoRT.pl

Interface program. Takes parameters from a special input_file.txt or using command-line parameters. See Quick_Manual.txt for more details on using this interface script, its command-line parameters, etc.

IV.B.2. Processing modules:

a. Process_single_read_file

- If 5adpt is used in the method sequence, it extracts adapter sequences using extract_5adpt_sequences.pm
- Create the output directory and prepare input and output filehandles
- Repeat until done with input file:
- Read 4 lines at a time (4 lines = one read) or 1 line from the qseq file (1 line = 1 read). The info from each read is used to create a single_fQ_read object.
- Pass the read object to the process_single_read
- When done, use print_final_SR_report.pm to print trimming stats

b. Process_PE_file.pm

- If 5adpt is used in the method sequence, it extracts adapter sequences using extract_5adpt_sequences.pm
- Create the output directory and prepare input and output filehandles
- Repeat until done with input files:
- Read 4 lines at a time from each of the two PE fastq files (4 lines = one read, 2 reads = a read pair) or 1 line from each of the qseq files (1 line = 1 read). The info from each read is used to create a single_fQ_read object, and the two objects (one for each read in the pair) are used to create a PE_fQ_pair.
- Pass the read pair to the process_PE_read_pair
- When done, use print_final_PE_report.pm to print trimming stats

c. Process_single_read.pm

- Takes the single_fQ_read object and applies trimming methods (listed below and in section II) in the order specified by the user, keeping track of trimmed bases and updating their counters.
- After applying each method to the PE read pair, process_single_read.pm checks the read's is_good flag:

`read->is_good = 1`: *the read is good*, the next trimming method is applied. If trimming is over, the read is printed out to the trimmed file.

`read->is_good = 0`: *the read is bad*, and is not printed out.

d. Process_PE_reads.pm

- Takes the PE_pair object and applies trimming methods (listed below and in section II) in the order specified by the user, keeping track of trimmed bases and updating their counters.
- After applying each method to the PE read pair, process_PE_reads.pm checks the pair's status:

  `PE_pair->get_status = 3`: *the pair is good*, the next trimming method is applied. If trimming is over, the pair is printed out to the trimmed _1 and _2 fastq read files.

  `PE_pair->get_status = 2 or 1`: *the pair is bad* (because at least one read is bad) *but one of its read is good*, the good read (now known as a single, unpaired, surviving or widowed read) is printed to surviving_SE_mates.fastq.

  `PE_pair->get_status = 0` :*the pair is bad and both reads are bad*, the pair is simply not printed out.

e. extract_5adpt_sequences.pm

extract_5adpt_sequences: extracts adapter sequence data from the adapter sequences file.

IV.B.3. Trimming Modules

Each of the following trimming modules takes a read pair or single read, applies a trimming method to it, then returns a new read (pair) or set of reads/pairs if nsplit was successful.

Each module contains three trimming subroutines: the actual trimming subroutine (which works for a single read), for trimming single reads, which is called by the subroutine for trimming PE reads. So, the subroutine for trimming PE reads uses the single_read subroutine to trim the read pair, then does some PE read-specific operations, like creating new daughter read pairs (nsplit and 5adpt-sp) or deleting the read pair altogether if it's flagged "bad" by lqperc, nperc, or 5adpt-kr.

The algorithms of the single_read trimming subroutines are explained in section II.

Note 1: All modules, except Three_end and Mott, can trim out entire reads resulting in the removal of the read pair from the trimmed output (see section V.1). Three_end and Mott can NOT trim out entire reads because they can trim a read only down to min_read_length, which is usually > 0. However, if min_read_length is set to zero, then three_end and mott CAN delete entire reads. See section I for recommended min_read_length values.

Note 2: Quality trimming modules require, as input, sequence and quality score lines from fastq and use quality score data to trim reads. These modules include *LQPerc, ThreeEnd*, and *Mott*. Non-quality trimming modules require only the sequence line and input and examine the alien bases (Ns, adapter sequences) to trim the reads. These modules include *NPerc, Nsplit*, and *five_prime_PE_adapter_trim*.

five_prime _adpt.pm

Note: you need to install CPAN String::Approx 'aslice' for this module to work.

USAGE:
To trim 5' (Illumina) adapter sequences from _1 and _2. It uses fuzzy matching (String::Approx 'aslice' to match (Illumina) adapter sequences (listed in /path/PE_fastq_trimmer_v.1.0/Illumina_PE_adapter_seqs.txt). The sequences listed in this file are explained in detail in Section VII.

Approximate matching is case-insensitive, and is done according to a user-specified match_percentage. A match_percentage of 90% means that for every 10 bases, only one mismatch is allowed, and so on.

The measure of approximateness for String::Approx is *Levenshtein edit distance*. More detail on how this String:: Approx works can be found at : http://search.cpan.org/~jhi/String-Approx-3.26/Approx.pm

Parameters:

- -5a_f is the adapter seq.s file, which can be one of our built-in libraries for Illumina and 454 primers (please see quick_manual.txt for the list) or a user specified filepath (the file MUST follow the five_prime_adapter_seq_TEMPLATE.txt format).
  Default is "i-g" (Illumina Genomic library)
  - Note that the library files are list at <ngsShoRT path>/illumina_and_454_primers

  - Available Illumina libraries are:
    i-g (Illumina genomic, Default), i-p (Illumina PE), i-m (Illumina multiplex),
    i-n (Illumina NlaIII), i-d (Illumina DpnII), i-r (Illumina sRNA)

  - Available 454 (pyrosequencing) libraries are:
    p-b (pyroseq basic), p-r (pyroseq sRNA), p-p (pyroseq PE), p-a (pyroseq amplicon)

  Alternatively, you can list your own adapter sequences in a modified copy of the template file, five_prime_adapter_seq_TEMPLATE.txt.

- -5a_axn what axn to take when one of the adapter sequences matches to a read (in addition to removing the adapter sequence, of course).
  kr: Kill the whole Read
  ka: Kill bases AFTER (to the 3' end side of) the adapter sequence

- -5a_mp INT is the matching percentage. 100 is Default. 90 means that one mismatch is allowed every 10 bases.

- [-5a_ins INT -5a_del INT -5a_sub INT]
  - These are optional modifiers for fuzzy matching (default is undefined). They refer to the maximum allowed number of insertions, deletions, and substitutions respectively.
  - So, for example (-5a_mp 90 -5a_ins 0 -5a_del 0) means that one mismatched character is allowed in every 10 chars, but it can NOT be a deletion or an insertion. Thus, it can only be a substitution.

- -5a_mx_len_dif is the maximum allowed difference in length between the adapter sequence and the substring (of the read) that it matches to. This is used to control unusual fuzzy-matching cases. Default is 3, which means the difference in length cannot be more than 3 bases.

- -5a_fmi is the Furthest allowed Matching Index, i.e., how far can the search go in the read. This depends on how you designed your reads. For example, if you know that there are no adapter sequences deeper than 50 bases into a read, set fmi to 50. Default is 'full' read length

SUBROUTINES
five_prime_adpt trims adapter sequences from single reads, and
- (5a_axn kr) kill the entire read
- (5a_axn ka) kill the detected adapter sequence and all bases after it

SR_five_prime_adpt
PE_five_prime_adpt

NOTES
(1) String::Approx 'aslice' does NOT extract the matched substring from the target string (the read). Instead, it returns an index for where the match begins and a size for the matched region. The five_prime_adpt subroutine uses these values to locate the approximately-matched string by extracting the substring that starts at index and ends at index+size.

If you do not have String::Approx installed in your library and/or you're having problems installing it from CPAN and/or you don't care much for approximate matching, see Startup_Tutorial.pdf to learn how to get around needing this module.

Some problems have been reported with aslice's approximation (see above URL) that generally are overestimating the match length, or returning a match that is too short (see below). If you want to avoid this altogether, simply use --5adpt_mp 100 to set the match percentage to 100, which makes aslice simply do what standard perl regex match does.

The main problem in our experience was that at 5adpt_mp 90-99, 'aslice' will occasionally find matches, then specify (index, size) values that give a substring that is too short:

```
Adapter                   matched substring in the read
CTCGGCATTCCTG        --> CTG
CTCGGCATTCCTG        --> CCTG
GATCGGAAGAGCGGTTCAG --> GAG
```

Our method for limiting such cases is using this optional modifier:
-5a_max_match_len_diff INT

Which specifies the maximum allowed difference in length between the adapter_sequence and the substring that it matches to (in the read). So, the length of the matched substring CANNOT be different from the length of the adapter sequence by more than 5adpt_max_match_len_diff. Our default value is 3.

(2) Our script uses a main variable for matching: match percentage (which we use to specify amatch percentage for 'aslice'). It also accepts optional modifiers: 5adpt_ins INT and 5adpt_del INT 5adpt_sub INT, which correspond to THE MAXIMUM ALLOWED number of insertions, deletions, and substitutions respectively (see above URL). We made these ins/del/sub modifiers optional for simplicity.

For example, (--5a_mp 90 --5a_ins 0 –5a_del 0) allows 1 mismatch for every 10 bases, but it can ONLY BE a substitution (implicit from setting both 5adpt_ins and 5adpt_del to 0).

(3) For debugging purposes, adapter trimming prints out the matched adapter sequences, their reads, match index and size as well as other info in the following file:

/path/output_directory/extracted_adapter_sequences_at_MP_percent_match.txt
where 5a_mp = the match_percentage used for this run (e.g. 100).

LQR.pm

USAGE:
To trim low quality reads. More specifically, trim reads with >= lq_p% of lqs bases, where lqs is a user-specified low quality score cutoff and lq_p is a user-specified percentage cutoff of lqs bases.

SUBROUTINES

The *LQR* subroutine checks each read for the percentage of LQ bases. If it's >= lq_p%, the sets the skip flag of a read to 1

The *SR_LQR* takes a read, runs it through &*LQR*, then updates the deleted read counters if (!read->is_good)

The *PE_LQR* takes a read pair, runs each read through &*LQR*, then updates the counters if the pair's status isn't 3 (both reads are good)

NOTES

N/A

Mott.pm

USAGE:

Uses the modified Richard-Mott trimming algorithm to trim a read from (potentially) both sides to extract a substring with the highest possible running sum of LmP (see VI) based on the given mott_limit and min_read_length values.

SUBROUTINES

Mott_to_min_read_length: the singleRead trimming subroutine. It uses the modified Richard-Mott trimming algorithm (Section VI).

The *SR_Mott* takes a read, runs it through Mott_to_min_read_length, then updates the deleted read counters if (!read->is_good)

PE_Mott: takes a read pair and runs each read through *Mott_to_min_read_length*, updates trimming stats, and returns the trimmed read pair.

NOTES

Mott_to_min_read_length trims a read's low quality bases with worst-case scenario being the trimming of the read down to min_read_length which is usually > 0. So, Mott almost never completely trims a read (and thus causes the read pair to be skipped). However, if min_read_length = 0, Mott is allowed to completely trim a read if all its bases are under mott_limit.

NPerc.pm

USAGE:

To trim reads with over nperc_p% (nperc_p is a user-specific cutoff percentage) of uncalled (N or .) bases. We call these reads ambiguous reads.

SUBROUTINES

NPerc: sets the skip flag of a read to 1 if its sequence has a percentage of Ns higher than nperc

The *SR_NPerc* takes a read, runs it through &*NPerc*, then updates the deleted read counters if (!read->is_good)

PE_NPerc: takes a read pair and runs each read through *PE_NPerc*, updates trimming stats.

NOTES

N/A

Nsplit.pm

USAGE:

To remove the largest block of Ns (that is not shorter than nsplit_len) from a read by splitting the read into right and left pieces (to the right and left of the N-block) that are used to create daughter reads which replace the parent read.

SUBROUTINES

Nsplit: the simple single read subroutine which simply searches the read for N-blocks whose length is >= min_read_length, then split the read around the longest of these N-blocks.
PE_Nsplit: takes a read pair and runs each read through Nsplit, and then manage all the possible cases of nsplit.

NOTES

N/A

TERA.pm

USAGE:

Uses TERA trimming to trim a read from the 3'-end, removing bases with a running average quality score that is under the tera_avg cutoff specified by the user.

SUBROUTINES
TERA: the singleRead trimming subroutine. Trims the seq and qual of a read

SR_TERA : takes a read and runs it through TERA, updates trimming stats.

PE_TERA : takes a read pair and runs each read through TERA, updates trimming stats.

NOTES
TERA trims a read's low quality bases with worst-case scenario being the trimming of the read down to min_read_length, which is usually > 0. So, *TERA* almost never completely trims a read (and thus causes the read pair to be skipped). However, if min_read_length = 0, *TERA* is allowed to completely trim a read if all its bases are under tera_avg.

qseq0.pm

USAGE: filter out qseq reads whose filtering flag (the last part of the qseq line) equals 0

qseqB.pm

USAGE:
Search for a string of 'B'-scored bases in an illumina-generated qseq or fastQ file. When found, delete the string
This methods works ONLY IF the quality score line is Illumina-mapped (ascii_zero is 64. So, '@' = 0 and so on). In qseq files, 'B' does not mean phred =2; it instead stands for 'unknow quality score.'

Parameters:

-qB_num :	cutoff num of B-scored bases (BSBs) for qseqB. Default is 5
-qB_mode :	'global' means count all BSBs in the read. 'local' means look for a string of BSBs that whose length is >= qB_num
-qB_axn :	what to do if something was detected. kr kills the whole read, ka isavailable only with 'local' mode and it removes the B-string and all bases after it

3end.pm

USAGE: trims x bases from the 3' end of a read

5end.pm

USAGE: trims y bases from the 5' end of a read

i2s

USAGE:
switch read quality score mapping from Illumina (ASCII of zero score = 64) to Sanger (ASCII of zero score = 33)

s2i

USAGE:
switch read quality score mapping from Sanger (ASCII of zero score = 33) to Illumina (ASCII of zero score = 64)

rmHP

USAGE:
remove homopolymer sequences made of certain bases (default is "agct", so all of them) if the homopolymer's length is >= a certain cutoff (default is 8)

IV.B.4. Reporting Modules

- print_final_PE_report.pm and print_final_SR_report.pm for PE and SR read files, respectively.
- Called by process_PE_file.pm and process_single_read_file.pm, respectively.
- Produce final_PE_report.txt and final_SR_report.txt, respectively.

V. Built-in Illumina and 454 Adapter Sequences

We have supplied ngsShoRT with built-in libraries of known illumina and 454 primers. The libraries can be accessed by the user from the -5a_f option:

Available Illumina libraries are:
i-g (Illumina genomic, **Default**), i-p (Illumina PE), i-m (Illumina multiplex), i-n (Illumina NlaIII), i-d (Illumina DpnII), i-r (Illumina sRNA)

Available 454 (pyrosequencing) libraries are:
p-b (pyroseq basic), p-r (pyroseq sRNA), p-p (pyroseq PE), p-a (pyroseq amplicon)

The official citation for 5' Illumina adapters is from:
http://intron.ccam.uchc.edu/groups/tgcore/wiki/013c0/Solexa_Library_Primer_Sequences.html

Roche/454/Pyrosequencing adaptors were copied from multiple primer kit documentations available at https://www.roche-applied-science.com/

When trimming, 5adpt allows the user to locate and remove NOT ONLY the matched sequence, but bases before and after it. This allows for potential cases of known artifact fragmentation. This optional feature is implemented in the way we format (and expect users to format) our adapter list:

Our trimmer takes a text file containing a list of adapter sequences. Each input line in the file consists of 4 parts, separated by tabs:

```
Read sequence        +n       -m
```

Where read is either r1 or r2 (read 1 or read 2), sequence is the 5'-3' adapter sequence, +n is the number of bases to be deleted AFTER (in the 3' direction) the matched sequence, +m is the number of bases to be deleted BEFORE (in the 5' direction) the matched sequence.

Note that 5a_axn ka (kill after) is essentially specifying +n (number of bases to delete bases after match) to all the bases 3' of the match.

VI. Output files

For PE input (-pe1 SR_foo_1.fq -pe2 SR_foo_2.fq):

- trimmed_SR_foo_1.fastq	
- trimmed_SR_foo_2.fastq	
- surviving_SE_mates.fastq	Contains surviving (widowed) mates. See prev section
- log.txt	Contains used params and trimmer progress
- final_PE_report.txt	Contains total and by-method trimming stats

For SR input (-sr SR_foo.fq):

- trimmed_SR_foo.fastq	
- log.txt	Contains used params and trimmer progress
- final_SR_report.txt	Contains total and by-method trimming stats

VII. References and Suggested Readings

Cox, M. *et al.* (2010) SolexaQA: At-a-glance quality assessment of Illumina second-generation sequencing data. *BMC Bioinformatics*, **11**:485.

CLC bio. CLC Genomics Workbench, User Manual. http://www.clcbio.com/files/usermanuals/CLC_Genomics_Workbench_User_Manual.pdf

FASTX-Toolkit, http://hannonlab.cshl.edu/fastx_toolkit/

Miller,J.R. *et al*. (2010). Assembly algorithms for next-generation sequencing data. *Genomics*, **95**, 315-327.

Schendure,J. and Hanlee,J. (2008) Next-generation DNA sequencing. *Nature biotechnology,* **26**, 1135-1145.

Zerbino,D. and Birney,E. (2008). Velvet: Algorithms for de novo short read assembly using de Bruijn graphs. *Genome Research,* **18** (5):821-829.

Zerbino,D. (2008). Velvet Manual, version 1.1. *Available online at http://www.ebi.ac.uk/~zerbino/velvet/Man ual.pdf*

DiGuistini,S. *et al.* (2009). De novo genome sequence assembly of a filamentous fungus using Sanger, 454 and Illumina sequence data. *Genome* ***Biology*****,** 10**:**R94.

Garcia,T.I. *et al.* (2011). Effects of short read quality and quantity on a de novo vertebrate transcriptome assembly. *Comparative Biochemistry and Physiology, Part C. ScienceDirect*, In Press.

Shulaev,V. *et al.* (2010). The genome of woodland strawberry (*Fragari vesca*). *Nature Genetics,* **43**, 109-116.

Haridas,S. *et al.* (2011). A biologist's guide to de novo genome assembly using next-generation sequence data: A test with fungal genomes. *Journal of Microbiological Methods*, **86**:3, 368-375.

Atherton,R. *et al.* (2010). Whole genome sequencing of enriched chloroplast DNA using the Illumina GAII platform. *Plant Methods*, **6**:22.