Demultiplex Report

Trevor Enright

August 2, 2018

Part 1

Table 1: File associations

File	Reads	Meaning
1294_S1_L008_R1_001.fastq.gz	read1	Forward sequence read
1294_S1_L008_R2_001.fastq.gz	index1	Forwrad index
1294_S1_L008_R3_001.fastq.gz	index2	Reverse index
1294_S1_L008_R4_001.fastq.gz	read2	Reverse sequence read

Lets specify a cut off mean score of whatever score is 99.9% likely to be correct or 0.1% likely to have an error.

That is $-10 \times log_{10}(1 - 0.999) = 30$ Phred Score.

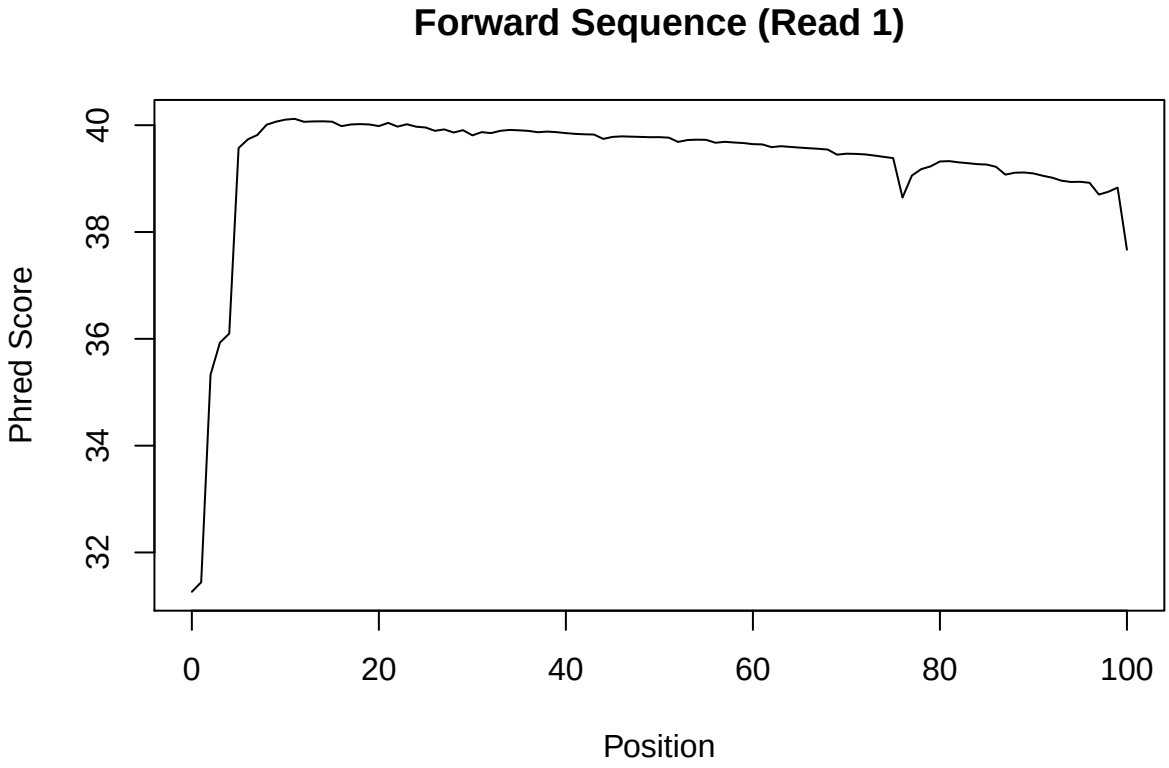
To get at the number of index reads that have one or more "N"

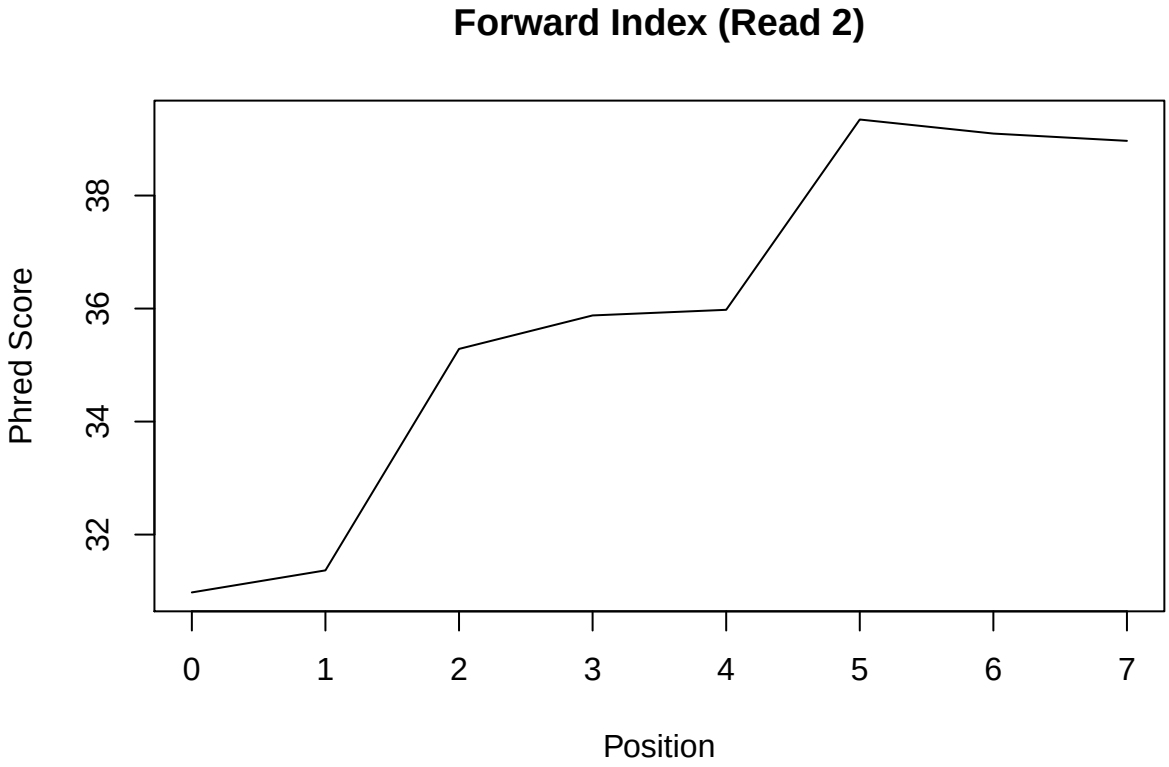
3. `$zcat 1294_S1_L008_R2_001.fastq.gz | sed -n 2~4p | grep -c N   3976613`

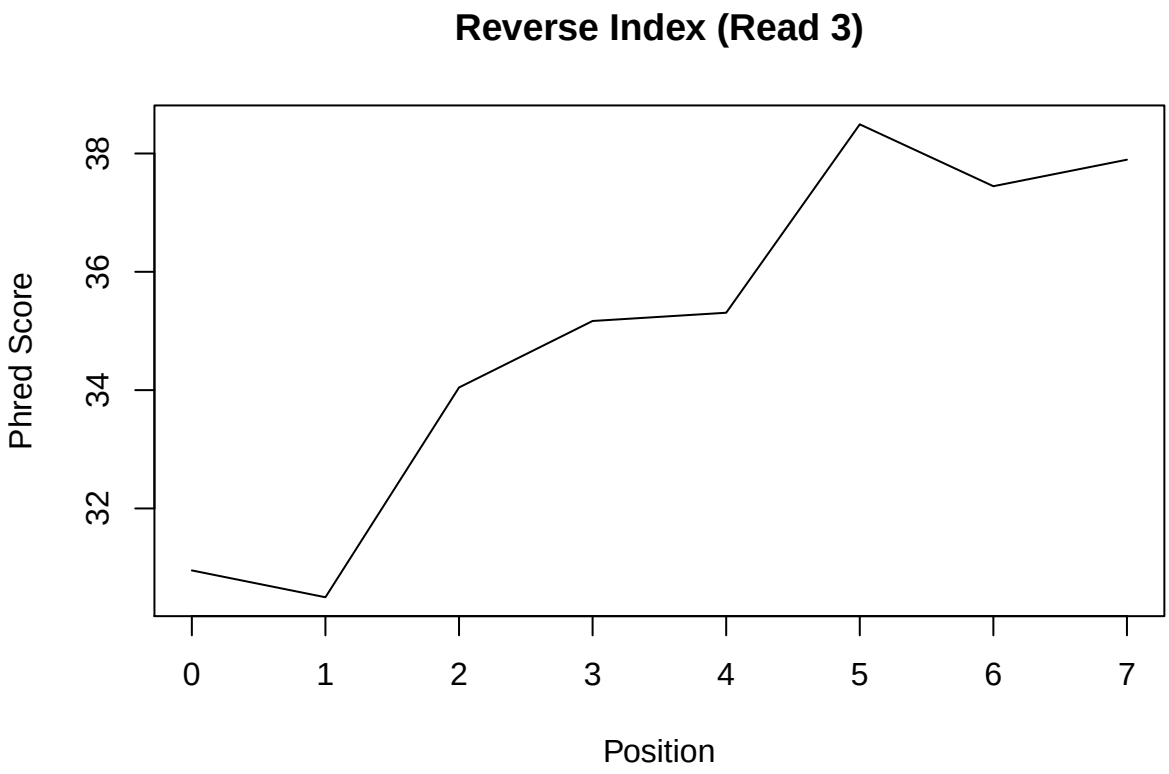
   `$zcat 1294_S1_L008_R3_001.fastq.gz | sed -n 2~4p | grep -c N   3328051`

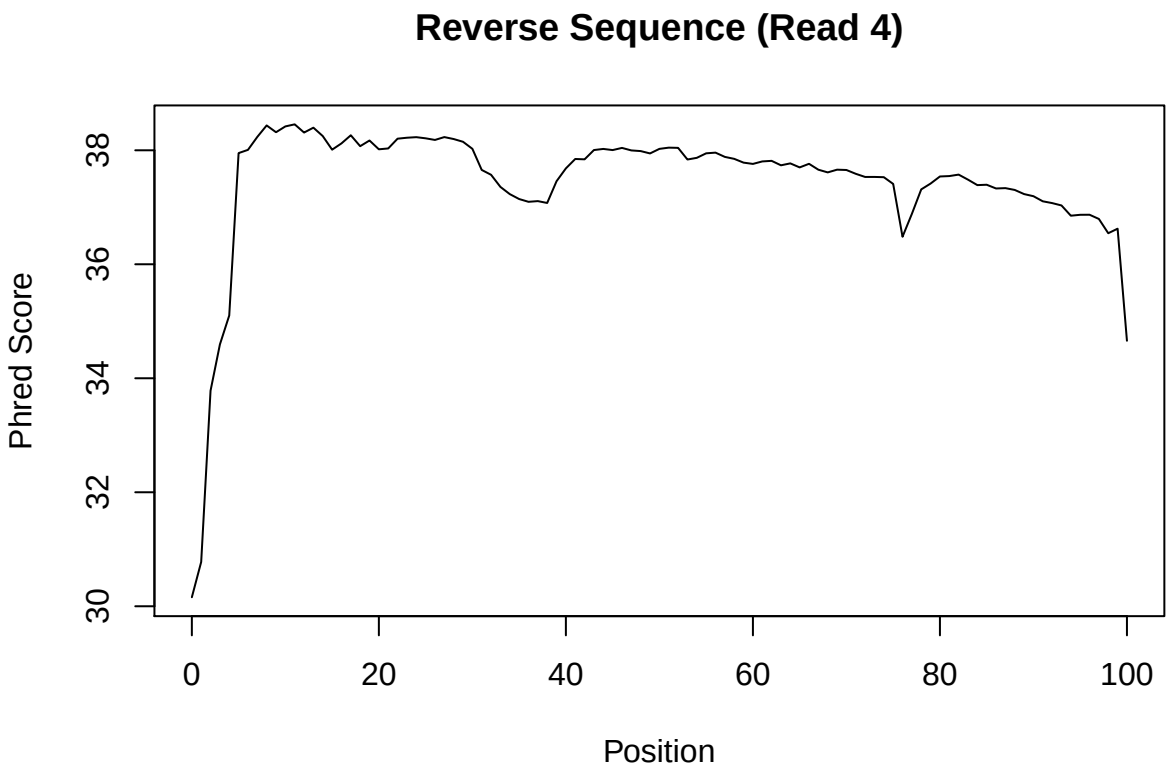
   `Thus, 7,304,664 index reads have one or more "N" in them.`

Average Quality by Position Plots:






Reverse Index (Read 3)
Phred Score
32
34
36
38
0
1
2
3
4
5
6
7
Position



Part 2

Problem: Indexes can be swapped during sequencing causing reads to be misassociated with their index. For ease of catching these errors for paired-end reads there is the same index (reverse compliment) placed on each side of the insert for both forward and reverse reads.

Solution: If swapping occurs then we should know by matching the forward index read with the reverse compliment index read. For the easiest solution, we'll only accept 100% matches that are one of the indexes provided. Ideally, we'd accept for mismatches that could only be one of the indexes provided using error correction methods including probability with set theory. After selecting for index hopping, or lack thereof, we'll select for average quality score of a read that is above a threshold.

The output will be 48 FASTQ files of the forward reads for each of the indexes forward and reverse. There will be an additional 2 text files for the reads that were unaccepted for index hopping and reads that were unaccepted due to not meeting the average quality score requirement.

General Algorithm:

1) Make bins for *keep* and *throw out* for testing index hopping
2) Make bins for *keep* and *throw out* for testing average quality score threshold
3) Consolidate bins and create FASTQ files for each index / bin *keep* and write FASTQ files
4) Create text files of the reads that do not meet quality score cutoff and those that are putatively index swapped or have "Ns"

Functions:

Reverse Compliment - given a DNA sequence (string), return the reverse compliment (string)

sample input: “GTAGCGTA” returns: “TACGCTAC”

IndexSwapping - given a R2 and R3 FASTQ index files and a set of indexes, compare R2 and R3 (reverse compliment) sequence lines and either pass or fail for that read. return a success dictionary with the value being the index and the key being the read number, also return lists of reads (the number of the read in the file) that *failed* either by having “N” or by putative swapping.

sample input: R3sample, R2sample, {“GTAGCGTA”, “CGATCGAT” , “GATCAAGG” , “AACAGCGA” , “TAGCCATG” , “CGGTAATC” , “CTCTGGAT” , “TACCGGAT”} sample output: {1,“GTAGCGTA”}, [2,3]

QualityCutoff - given either R1 or the R4 FASTQ read file and a cutoff phred score, calculate the mean quality score, x, of that read. If x is above the cutoff phred score: bin into a list of *keep*; else bin into list of *throw out*. return *keep* list and *throw out* list.

sample input: R1sample, 20 sample output: [1,2,3] , []

WriteFASTQfiles - given output dictionary and list from IndexSwapping and QualityCutoff keep list: Create 48 strings for each of the 24 indexes, 1 forward and 1 reverse; go through the R1 and R4 files and if the key in the dictionary then append the read from R1 and R4 to the string of the dictionary value index. Write to the output files. return None.

WriteDismissed - given output *failed* lists from IndexSwapping and QualityCutoff *throw out* list: loop through lists and append to either a string of IndexSwapping or string of QualityCutoff. Write each string to a separate file. return None.

Advanced Method:

Instead of dismissing indexes with “N”s we can make a guess at what it was supposed to be. An algorithm to accomplish this would be to:

1) When an N is encountered in an index, make a set of putative indexes by removine the N and replacing with either A,G,T,C. Thus if an index has one N there would be four different possibilities in the set. If there was two N’s there would be 4^2 = 16 possibilities in the set.
2) Find the intersection of this set with the set of known indexes. If the intersection is just one index then we keep the intersection as the index.

Useful Functions:

BuildSet - given an index (string) with at least one “N”, return a set with the possibilities of what that index could be.

sample input: “NCATGACG” returns: {“ACATGACG”,“TCATGACG”,“CCATGACG”,“GCATGACG”}

Intersect - given two sets, If the intersection of the sets is one element then return that element, else return False

sample inputs: {“ACATGACG”,“GGACTGCT”} {“ACATGACG”,“TCATGACG”,“CCATGACG”,“GCATGACG”} returns: {“ACATGACG”}

Files:

Sample FASTQ read file read (R1sample):

```
@K00337:83:HJKJNBBXX:8:1101:1265:1191 1:N:0:1
GNCTGGCATTCCCAGAGACATCAGTACCCAGTTGGTTCAGACAGTTCCTCTATTGGTTGACAAGGTCTTCATTTCTAGTGATATCAACACGGTGTCTACAA
+
A#A-<FJJJ<JJJJJJJJJJJJJJJJFJJJJFFJJFJJJAJJJJ-AJJJJJJJFFJJJJJJFFA-7<AJJJFFAJJJJJF<F--JJJJJJF-A-F7JJJJ
@K00337:83:HJKJNBBXX:8:1101:1289:1191 1:N:0:1
ACGGTGCTAGACGGTGCTAGACGGTGCTAGACGGTGCTAGACGGTGCTAGACGGTGCTAGACGGTGCTAGACGGTGCTAGACGGTGCTAG
+
A#A-<FJJJ<JJJJJJJJJJJJJJJJFJJJJFFJJFJJJAJJJJ-AJJJJJJJFFJJJJJJFFA-7<AJJJFFAJJJJJF<F--JJJJJJF-A-F7JJJJ
@K00337:83:HJKJNBBXX:8:1101:1299:1191 1:N:0:1
GNCTGGCATTCCCAGAGACATCAGTGNCTGGCATTCCCAGAGACATCAGTGNCTGGCATTCCCAGAGACATCAGTGNCTGGCATTCCCAGAGACATCAGT
+
A#A-<FJJJ<JJJJJJJJJJJJJJJJFJJJJFFJJFJJJAJJJJ-AJJJJJJJFFJJJJJJFFA-7<AJJJFFAJJJJJF<F--JJJJJJF-A-F7JJJJ
```

Sample FASTQ read file read (R4sample):

```
@K00337:83:HJKJNBBXX:8:1101:1265:1191 1:N:0:1
GNCTGGCATTCCCAGAGACATCAGTACCCAGTTGGTTCAGACAGTTCCTCTATTGGTTGACAAGGTCTTCATTTCTAGTGATATCAACACGGTGTCTACAA
+
A#A-<FJJJ<JJJJJJJJJJJJJJJJFJJJJFFJJFJJJAJJJJ-AJJJJJJJFFJJJJJJFFA-7<AJJJFFAJJJJJF<F--JJJJJJF-A-F7JJJJ
@K00337:83:HJKJNBBXX:8:1101:1289:1191 1:N:0:1
ACGGTGCTAGACGGTGCTAGACGGTGCTAGACGGTGCTAGACGGTGCTAGACGGTGCTAGACGGTGCTAGACGGTGCTAGACGGTGCTAG
+
A#A-<FJJJ<JJJJJJJJJJJJJJJJFJJJJFFJJFJJJAJJJJ-AJJJJJJJFFJJJJJJFFA-7<AJJJFFAJJJJJF<F--JJJJJJF-A-F7JJJJ
@K00337:83:HJKJNBBXX:8:1101:1299:1191 1:N:0:1
GNCTGGCATTCCCAGAGACATCAGTGNCTGGCATTCCCAGAGACATCAGTGNCTGGCATTCCCAGAGACATCAGTGNCTGGCATTCCCAGAGACATCAGT
+
A#A-<FJJJ<JJJJJJJJJJJJJJJJFJJJJFFJJFJJJAJJJJ-AJJJJJJJFFJJJJJJFFA-7<AJJJFFAJJJJJF<F--JJJJJJF-A-F7JJJJ
```

Sample FASTQ index file read (R3sample):

```
@K00337:83:HJKJNBBXX:8:1101:1265:1191 2:N:0:1
TACGCTAC
+
#AA<FJJJ
@K00337:83:HJKJNBBXX:8:1101:1289:1191 2:N:0:1
NACAGCGA
+
#AAAFJJJ
@K00337:83:HJKJNBBXX:8:1101:1299:1191 2:N:0:1
NTCCTAAG
+
#AAFFJAJ
```

Sample FASTQ index file read (R2sample):

```
@K00337:83:HJKJNBBXX:8:1101:1265:1191 2:N:0:1
GTAGCGTA
+
#AA<FJJJ
@K00337:83:HJKJNBBXX:8:1101:1289:1191 2:N:0:1
TCGCTGTA
+
#AAAFJJJ
@K00337:83:HJKJNBBXX:8:1101:1299:1191 2:N:0:1
CTTAGGAT
```

```
+
#AAFFJAJ
```