1 Running the error correction code

1.1 Java I

Main Sequence

Some code, (e.g. adaptor trimming) needs to be run for each data set individually, while the majority is coded to run all data sets through sequentially.

```
STEP 1a:
Trims adaptors from sequences in FastQ format
Input: ..._sequence_fastq.txt
Run: -> Trim/Trim.java
Outputs: trimmed_sequences_...txt, (trimming_stats...txt)
```

An alternative to the step above is to use the programs FastTrim or FastCutTrim detailed below. This executes must faster as it does not collect detail matching statistics and it only searches for adaptor lengths which result in trimmed sequences of length 19 to 25, in which we are interested. FastCutTrim is designed for longer reads, for example it trims reads to 35 nucleotides long before matching adaptors. This saves time and allows for reuse of code.

```
STEP 1b:
Trims adaptors from sequences in FastQ format
Input: ..._sequence_fastq.txt
Run: -> Trim/FastCutTrim.java OR FastTrim.java
Outputs: trimmed_sequences_...txt, (trimming_stats...txt)

STEP 2:
Removes sequences outside a certain length range:
Input: Trimmed_Sequences_...txt,
Run: --> PostTrim/LengthCull.java
Outputs: Culled_...txt,

STEP 3:
Sorts the sequences alphabetically
Input: Culled_...txt
---> PostTrim/Sorting.java
Output: Sorted_.C_...txt

STEP 3a (Optional):
Counts the tracts in a set of sequences.
Input: trimmed_sequences_.txt
---> CountTracts.java
Outputs Tracts_.txt
Also see the folowing files for work concerning homopolymer tracts
unrelated to the paper Sleep et. al. 2013:
Aprobs.java (expected tracts)
Ahist.pl (observed tracts)
```

```
STEP 4:
This program is called merge for historical reasons, no merging is actually
done.  Sequences are post-fixed with the source cultivar.  Input files are
already sorted, output file remains sorted.
Input: Sorted_...txt
---> Merge/Merge.java
Outputs: Merged_...txt

STEP 5:
Counts abundance of each sequence and writes only the unique sequence.
Also calculated average quality scores per each base, affixes the name
of the cultivar (or equivalent) which contributed most in abundance for each read
Input: Merged_...txt
---> Merge/Unique.java
Outputs: Unique_...txt

STEP 6:
Sorts sequences in order of abundance.
Affixes a name constructed of the cultivar and its ranking
Input: Unique_...txt
---> Merge/Rank.java
Outputs: Ranked.txt
```

1.2 Unix

```
STEP 7:
To reverse the Ranked file use (*nix commands)
cat yourfile | perl -e '@x = <STDIN>; while ($y = pop @x) {print $y;}' > newfile
cat Ranked.txt | perl -e '@x = <STDIN>; while ($y = pop @x) {print $y;}' > Reversed.txt
```

1.3 Java II

Sequences of a set length

This sequence of code is at this stage run for a number of set lengths. In the example files, lengths 20 to 24 are coded. References to length 21 below are just as an example.

```
STEP 8:
Make list of sorted 1-SNP variants of every sequence.
This is to aid in the contruction of graphs of related sequences.
Input: Reversed.txt
---> GraphsLinks/MakeSortedVariants.java
Output: SortedList21.txt, (VariantList21.txt)

STEP 9:
Makes linked list of parent and child objects
Input: SortedList21.txt
---> GraphLinks/MakeLinkList.java
```

```
Outputs: Graph21.dat (for matlab e.g.: 1 2 1)
Outputs: LinkList21.txt (e.g. RC3322311 to RC6687822)
Outputs: Key21.txt (e.g. 1 RC3322311)

STEP 10:
This creates an automatic file of matlab code that can
be run to decompose the graphs (see next step).
Also adds reversed link which includes the highest named
  node in the opposite position to which it occurs
  (i.e. from or to)
  this is to ensure a sparse matrix with the same
  number of rows and columns (square - required by matlab)
  (and ensures carriage return on final line)
Input:
---> Graphlinks/FormatForMatlab.java
Output: Graphlinks/formatlab/AutoGraphDecomp.m, XX_YY.dat (graph files for matlab)
```

1.4 Matlab

Graph Decomposition

The following alternative step is performed for all sequences, using the automatically generated code from step 10 above.

```
STEP 11b:
Decomposes the link list into connected subcomponents.
Input: XX_YY.dat (e.g. RT_3_20.dat)
---> Graphlinks/formatlab/AutoGraphDecomp.m
Output: NumSubGNodes.txt, Subgraphs_XT_L_ln.txt
```

1.5 R Alternative to Section 1.4

Decomposes each graph into its subgraphs using the R Package igraph. Output is in the format of subgraph each node belongs to with 5 items recorded on each line.

```
STEP 11c:
Graphs/XX_YY_2X
--->RDecomp/Decompose.r
RSubgraphs/RSubgraphs_XX_Y_2X.txt
```

1.6 Java III

Note that SubgraphsXX.txt appears to have records over multiple lines when opened in notepad (even with wordwrap turned off), this is not the case and data appears on a single line.

The initial filtering for this step is done with a size cutoff for number of nodes for a large graph of (recommended 40-200 depending on size of data set and frequency of errors - a too high threshold will give too little data). An extra condition of a certain proportion of the reads lying in the main read (most abundant node) has been developed. An initial proportion of 90% is used,

and progressively lower thresholds for filling in gaps. These files are used for input into R code (See step 14)

```
STEP 12:
Input: subgraphsXX.txt, KeyXX.txt, GraphXX.dat, NumNodesSubgraphs.txt
---> Model/SubgraphsD.java
Outputs: probabilitySummaryXX.java - probability of errors at different
positions and of different types
ChildFileXX_YY.txt - children of sequence length XX large graph num YY
graphDetailXX_YY.txt - links with node names, actual sequence and
 abundances
graphLinksXX_YY - just the links
```

When the graphs are decomposed the singletons are left out. This is unavoidable in the matlab code, but can be turned on or off in the R code. However, to keep things as similar as possible (and to reduce the number of graphs that are corrected), this is turned off in the R code and the singletons need to be found using the FindOrphans program detailed below.

```
STEP 13 (can be done at any time):
Finds all orphans which were lost in the graph
 decomposition. Does this by comparing the Key file
  (which it also sorts into alphabetical order by sequence
  to the Sequences files which it also sorts (in same manner
Input: Sequences..., Key...
---> FindOrphans.java
Output: Orphans..., (Key_Sorted...)
 (check that closing files is happening as some
 files appear to have a half line at the end.)
```

These Orphan sequences will need to merged back in later. The majority of these are usually of abundance 1.

1.7 R

The following R code is at the heart of the error fitting model. It reads in the probabilities of an error at each position for each graph. It calculates the weighted (on abundance) average of these probabilities. It then constructs 95% binomial confidence intervals (using this weighted average as the proportion) and removes 'outliers' deemed too high more likely to be biological variants. It then recalculates the weighted average and checks the confidence intervals again. These estimates are calculated for each of the main read proportion thresholds (90%,...). Higher threshold estimates are used first, if there are no examples, lower results are used in their place. After this a second check for outliers (lowering reads that are over 2 times higher than the average of both neighbours) is performed. An exponential curve in positions 2 to 24 is then fitted.

```
STEP 14:
Input: Single AllProbs
binomialWWWWW(date).r
```

1.8 Java IV

Correction

Now the code needs to classify the sequences in every graph as a sequencing error or biological variant. This is done using the model constructed over previous steps. First all the graphs (not just the large ones) are constructed. Then these are traversed and sequences written to the appropriate output files as described below.

```
STEP 15:
Input: NumNodesSubg.txt,Subgraphs/Subgraphs_...,
       Keys/Key..., FullGraphs/Graph...
--> Correcting/WriteGraphsD.java
Output: output.../graph_... (A file for each graph)

STEP 16:
Input: NumNodesSubg.txt, input/new_final_fits_...,
       output.../graph_...
--> Correcting/TraverseD.java (simple threshold)
--> Correcting/TraverseHyD.java (hypothesis testing)
Output: probs/Summary_..., Classified/errors_....,
        Classified/correct_...., Classified/abundantCorrect...,

The steps above also require the additional classes:
- Model.java
- Node.java
- NodeList.java
- SeqUtil.java
- SNP.java
```

Tract removal can be done as a final step if still necessary.

```
Removes Poly-A, C, G and T tails
Input:
--> PostTrim/TractRemoval.java
Outputs: TractFree_...txt,
```

Code for plotting is in the plotting folder.