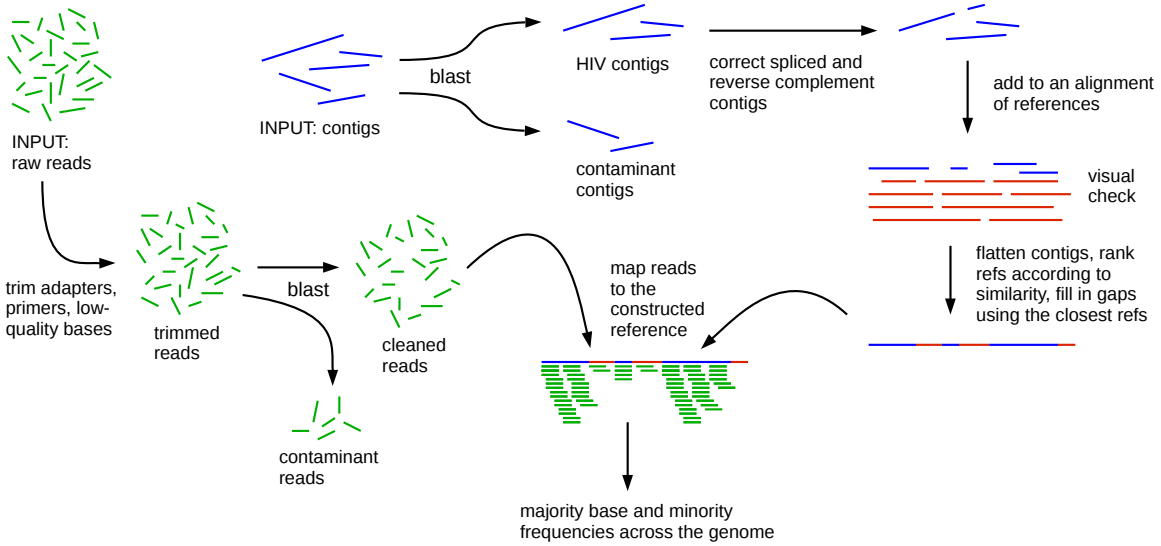
`shiver`

available from https://github.com/ChrisHIV/shiver

By Chris Wymant

This manual last updated December 4, 2017



`shiver` is a tool for mapping paired-end short reads to a custom reference sequence constructed using do novo assembled contigs, in order to minimise the biased loss of information that occurs from mapping to an reference that differs from the sample. From the mapped reads, base frequencies are quantified, and a consensus sequence is called. `shiver` was designed for HIV but has broader applicabilty; if you are using it for sequence data from organism X instead of HIV, mentally replace *HIV* by *X* when reading this manual.

The method and its performance are discussed in [1]; please cite this if you find `shiver` helpful. If you use `shiver`, please also cite the publications of its dependencies: `SAMtools` [2], `MAFFT` [3], `trimmomatic`[4], `BLAST` [5], and `BWA` [6] or `bowtie` [7] if you switched the default mapper from `smalt` (for which there is no publication to cite). Details of these are found in the file `CitationDetails.bib` in `shiver`'s `info` subdirectory.

If you have a problem getting the code to run or think you've found a bug, please create a *New issue* on the GitHub repository.

Throughout this manual, I'll assume your `shiver` code lives in `~/shiver/`. If you downloaded it somewhere else, e.g. `my/other/directory/shiver/`, simply replace every occurence of `~/shiver/` in by `my/other/directory/shiver/` (or if you're feeling lazy, create a link to `my/other/directory/shiver`, call it `shiver`, and put it in your home directory).

By default, to run `shiver` commands, you have to tell the command line where those commands can be found, i.e. include their path: `~/shiver/shiver_align_contigs.sh`. If you don't want to type the path every time, so that you can just type `shiver_align_contigs.sh`, you need to add the directory containing the `shiver` code (`~/shiver/`) to your `PATH` environmental variable. Google this if you don't know how to do that.

Throughout this manual, a `$` character at the start of a line just indicates that what follows should be run from command line – it should not be entered as part of your command.

1 Initialisation

Before you begin processing a collection of samples there's an initialisation step: it should be run once only (i.e. not once for each sample). It requires

- The `shiver` configuration file, named by default `config.sh`. (Here it is needed only to tell `shiver` how to run the program blast; you can change your mind about all the pipeline parameters contained in this file later on. See also the section *The config file.*);
- An alignment of your choice of existing reference genomes (lots of which are available to download from the Los Alamos National Lab) called `RefAlignment.fasta`, say.
- Fasta files containing the adapters and primers used for sequencing, called `adapters.fasta` and `primers.fasta`, say. (Adapters will be removed using trimmomatic, and "The naming of the various sequences within this file determines how they are used" – trimmomatic docs. In the `ExampleInput` directory of this repository is a default Illumina adapters file, and a file containing the primers relevant to our sequence data – those of Gall *et al.* (J. Clin. Microbio. 2012). **Do not** assume these are what's needed for your sequence data – instead ask your sequencing team to provide the adapter and primer sequences they used.)

Initialisation files will be put into a directory called `MyInitDir` if you run

```
$ ~/shiver/shiver_init.sh  MyInitDir  config.sh  RefAlignment.fasta  adapters.fasta \
  primers.fasta
```

(remembering to skip that first `$` character.) You shouldn't touch `MyInitDir` or the files therein after running this command, or unknown bad things might happen.

After running that initialisation command (once only!) you can process any number of samples. Since sample processing generates (many) files in the working directory, to avoid overwriting files you should always work in an empty directory for each sample, as illustrated in the *Scripted usage to process batches of samples* section below.

2 `shiver` input

The input for each sample is (1) paired-end short reads, and (2) contigs generated by running *de novo* assembly on those short reads.

1. The forward read names must all end in "/1", the reverse read names must all end in "/2", and mates in a pair must match in what comes before this. e.g. Forward read names might be *FirstReadName/1*, *SecondReadName/1*, ... and reverse read names should then be *FirstReadName/2*, *SecondReadName/2*, ... This is so `shiver` knows how to pair the reads together.
2. Our chosen assembler is IVA; it can be installed from here or run on a virtual machine here, and the publication is here. Say your forward reads are in `reads_1.fastq.gz` and your reverse reads are in `reads_2.fastq.gz`; assembling them into contigs is as simple as running

   ```
   $ iva  -f  reads_1.fastq.gz  -r  reads_2.fastq.gz  MyOutputDirectory
   ```

   though you should run `iva --help` to see the options, including specifying adapter and primer sequences to trim from the reads. `shiver` wants *all* contigs produced by assembly; if by curiosity you check what organisms the different contigs come from, do not remove non-HIV contigs. These contaminant contigs (if present) are used to help remove contaminant reads.

3 How to process a single sample

Processing requires two commands. With forward/reverse read files named as above and contigs in `contigs.fasta`, the first is

```
$ ~/shiver/shiver_align_contigs.sh  MyInitDir  config.sh  contigs.fasta  SID
```

replacing `SID` ('sample ID') by a name used for labelling all output files for this sample. This command will produce a file named `SID.blast` (detailing blast hits of your contigs to those existing references supplied for the initialisation). Assuming at least one contig looks like HIV, `SID.blast` will not be empty (see section *Including samples without contigs* otherwise), and there will be another file called `SID_raw_wRefs.fasta` – an alignment of the HIV contigs (i.e. those that have blast hits) to your input existing reference genomes. As discussed in the `shiver` article, this alignment should be inspected, and in the minority of cases where this is required, edited; see the following section.

`shiver` decides that a contig needs correcting, if it

- is not wholly spanned by a blast hit, suggesting that the ends are not genuine HIV and should be trimmed off;
- has two or more blast hits (ignoring hits wholly inside other hits), suggesting that it is chimeric/spliced – erroneously connecting disconnected parts of the genome;
- has a blast hit in the opposite orientation to the reference, suggesting that that contig (or part of the contig) should be reverse complemented;
- contains an unrealistically large gap after alignment, to the existing references, suggesting that it should be split into two separate contigs.

Appropriate action is taken in each case, cutting and reverse-complementing as needed. If any of these actions were needed for any of the contigs, an additional file `SID_cut_wRefs.fasta` will be produced, which contains all contigs after correction aligned to the existing references. We expect that `SID_cut_wRefs.fasta` (if it exists) is a better alignment than `SID_raw_wRefs.fasta`, but it's safer not to take this for granted, for example unusual indels can result in multiple blast hits. It's therefore recommended to inspect both alignments, choose which set of contigs you think are more likely to approximate the true consensus, edit the alignment if you spot what you believe to be errors, and discard the other alignment.

With an alignment of contigs you're happy with, say `SID_raw_wRefs.fasta`, run

```
$ ~/shiver/shiver_map_reads.sh  MyInitDir  config.sh  contigs.fasta  SID  \
  SID.blast  SID_raw_wRefs.fasta  reads_1.fastq.gz  reads_2.fastq.gz
```

and you're done. You'll probably want to delete the intermediate files produced during processing (beginning with `temp` by default), i.e. `rm temp*`.

4 A manual step between the two automatic steps? Blasphemy!

The reason for this is discussed in more detail in the `shiver` paper. Correct alignment has already been attempted automatically in the first step, but unfortunately aligning whole HIV genomes automatically, perfectly, 100% of the time is a dream: HIV has very high mutation and substitution rates, including frequent insertions and deletions. Compounding this problem is the fact that de novo assembly output is sometimes an imperfect representation of the sample, and aligning problematic sequences may give problematic alignments. For most samples the contigs and their alignment are fine, and seeing this from the alignment takes one or two seconds of human time. When editing is required, it amounts to one of two things: deleting a whole contig if it appears to be wholly junk (i.e. short stretches of poorly aligned sequence separated by large gaps), or deleting the end of a contig if just the end looks wrong (i.e. a stretch of sequence in very poor agreement with the existing references, or a stretch of sequence separated from the rest of the contig by a gap implausibly large for a real deletion). Visually checking and editing when needed ensures an alignment of contigs you can trust, allowing the second `shiver` command to reliably construct a) a tailored reference that minimises mapping bias, and b) a global alignment of all samples, instantly, without further need for an alignment algorithm: see the end of section *Scripted usage to process batches of samples*. This R package by Oliver Ratmann uses machine learning to correct such alignments; however for each sample you must manually check the corrections to ensure they are correct, instead of checking the alignment itself.

5 What output do I get?

- `SID.bam`, `SID_ref.fasta`: the bam file of mapped reads, and the reference to which they were mapped. (It's a shame the bam file format does not include the reference, c'est la vie.)
- `SID_BaseFreqs.csv`: the frequencies of A, C, G, T, '-' for a deletion, and N for unknown, at each position in the genome. `SID_BaseFreqs_ForGlobalAln.csv` is almost the same thing; it contains an extra column with a coordinate that can be used to compare different samples, at the cost of having to skip some rows for which the coordinate is not defined (see section 8). `SID_BaseFreqs_WithHXB2.csv` contains the base frequencies and an extra column for the coordinate in HXB2; again this has the cost of needing to exclude some rows (where the reads had an insertion with respect to the reference).
- `SID_MinCov_X_Y.fasta`: the pairwise alignment of the consensus genome and the reference used for mapping. In place of `X` and `Y` here you'll see the two coverage thresholds specified in `config.sh`: `X` is the minimum number of reads to call a base (if there are fewer than `X` reads we call `?` instead of a base), `Y` the minimum number to use upper case for the base (to signal increased confidence). The mapping reference is included with the consensus here to give context to any `?` characters.
- `SID_MinCov_X_Y_ForGlobalAln.fasta`, `SID_coords.csv`: these files are useful for when multiple samples are processed, so we postpone their explanation to section 8.
- `SID_clean_1.fastq`, `SID_clean_2.fastq`: the unmapped reads after removal of adapters, primers, low-quality bases, and those read pairs suspected of being contamination.
- `SID_InsertSizeCounts.csv`: the insert-size distribution.

Files that are not produced by default, but whose production can be specified in `config.sh`:

- `SID_MinCov_X_Y_wContigs.fasta`: an alignment of the consensus, the reference used for mapping, and all HIV contigs for this sample (as they were after any automatic cutting and any manual editing).
- `SID_ContaminantReads.bam`: a bam file of only the contaminant reads mapped to the HIV reference (to illustrate the importance of removing these reads from the HIV reads prior to mapping).

If the config file variable `remap` is left at its default value of `true`, some of the files named above will have a second copy with `SID` replaced by `SID_remap` in the file name: these will likely be very similar, but expected to be slightly more accurate – if they exist, use them instead.

6 The config file

In `config.sh` you can change pipeline parameters from their default values. During development of `shiver`, `config.sh` has often been updated (e.g. new features might require new parameters). As this may happen again in the future, to make sure you keep any changes you've made to default values, I recommend the following: make one copy of `config.sh` called `config_original.sh`, a second copy called `config_MyChanges.sh`, leave the former untouched and change whatever you like in the latter. Then when you update your `shiver` code, running

```
$ diff config_original.sh ~/shiver/config.sh
```

will show you whether the update has changed the config file; if so, running

```
$ diff config_original.sh config_MyChanges.sh
```

will remind you of the changes you made to default parameter values.

7 Scripted usage to process batches of samples

Before we start, note that scripted use of `shiver` (like any other command-line program) to process multiple files is easier if you know the basics of playing with filenames from the command line. Consider this toy example:

```
# We have two directories: one with text files, one with csv files.
$ ls CsvDir/
bar.csv  foo.csv
$ ls TxtDir/
bar.txt  foo.txt
# Assign the text file we want to a variable:
$ DesiredTextFile=TxtDir/foo.txt
# This shows how to find the associated csv file:
$ DesiredTextFileNoPath=$(basename "$DesiredTextFile")
$ echo "$DesiredTextFileNoPath"
foo.txt
$ DesiredTextFileNoPathNoExtention="${DesiredTextFileNoPath%.txt}"
$ echo "$DesiredTextFileNoPathNoExtention"
foo
$ AssociatedCsvFile=CsvDir/"$DesiredTextFileNoPathNoExtention".csv
$ ls "$AssociatedCsvFile"
CsvDir/foo.csv
# Or, the above steps all in one go:
$ DesiredTextFile=TxtDir/foo.txt
$ ls CsvDir/$(basename "${DesiredTextFile%.txt}").csv
CsvDir/foo.csv
```

Now back to `shiver`. Say your current directory contains three subdirectories: `contigs`, `MyInitDir` and `reads`, containing what the name suggests. Here are your contigs:

```
$ ls contigs/
sampleA.fasta sampleB.fasta sampleC.fasta
```

and here are your reads:

```
$ ls reads/
sampleA_1_fastq.gz sampleA_2_fastq.gz
sampleB_1_fastq.gz sampleB_2_fastq.gz
sampleC_1_fastq.gz sampleC_2_fastq.gz
```

Start by aligning the contigs for all samples:

```
$ for ContigFile in contigs/*.fasta; do
  # Extract the SID from the filename: remove the extension and the path
  SID=$(basename "${ConfigFile%.fasta}")
  # Make, and change into, an empty directory for processing this contig file
  mkdir AlignmentOutput_"$SID"
  cd AlignmentOutput_"$SID"
  ~/shiver/shiver_align_contigs.sh ../MyInitDir ../config.sh "$ContigFile" "$SID"
  cd ..
done
```

As explained above, for those samples for which contig correction is necessary, `SID_cut_wRefs.fasta` will be produced as well as `SID_raw_wRefs.fasta`, and only the better looking of these two should be kept (and edited if needed). Let's say you put one of these two files for each sample all together in a directory called `CheckedContigAlignments`.

```
# For samples that had an SID_cut_wRefs.fasta file, we kept either that file or
# SID_raw_wRefs.fasta. Let's rename all these files to have the same suffix -
# removing '_raw' or '_cut' to leave just SID_wRefs.fasta - to make it easier
# to find the one we want.
$ cd CheckedContigAlignments
$ for alignment in *_cut_wRefs.fasta; do
  mv -i "$alignment" "${alignment%_cut_wRefs.fasta}"_wRefs.fasta
done
$ for alignment in *_raw_wRefs.fasta; do
  mv -i "$alignment" "${alignment%_raw_wRefs.fasta}"_wRefs.fasta
done
# Now let's map! Make and change into an empty directory for processing each sample.
$ cd ..
$ for ContigFile in contigs/*.fasta; do
  # Find the other files for this sample.
  # NB careful scripting would check that files exist before using them.
  SID=$(basename "${ConfigFile%.fasta}")
  mkdir MappingOutput_"$SID"
  cd MappingOutput_"$SID"
  BlastFile=../AlignmentOutput_"$SID"/"$SID".blast
  alignment=../CheckedContigAlignments/"$SID"_wRefs.fasta
  reads1=../reads/"$SID"_1.fastq.gz
  reads2=../reads/"$SID"_2.fastq.gz
  ~/shiver/shiver_map_reads.sh ../MyInitDir ../config.sh "$ContigFile" \
  "$SID" "$BlastFile" "$alignment" "$reads1" "$reads2"
  cd
done
```

and you're done mapping.

8 The Global Alignment

Now for those aforementioned `*_MinCov_X_Y_ForGlobalAln.fasta` files. These are generated by excising unique insertions seen in a sample (i.e. not seen in any of the existing references used for initialisation), and then inserting gaps using coordinate translation in such a way that these files are all aligned with each other and with the alignment of existing references. Constructing a global alignment is then achieved just by putting all those files into one file:

```
$ cat MappingOutput_*/*_ForGlobalAln.fasta > GlobalAln.fasta
$ cat MyInitDir/ExistingRefAlignment.fasta >> GlobalAln.fasta
```

Now it's over to you for phylogenetics, GWAS etc.

9 Including samples without contigs

You might have some samples for which none of the contigs have blast hits to the existing references you supplied, i.e. it looks like all contigs are contaminants. In this case the contig alignment step will produce an empty blast file and no `SID_raw_wRefs.fasta` file. If you have total confidence in your *de novo* assembler, you could choose to dismiss such samples as total sequencing failure – that would be reasonable. On the other hand perhaps there are enough HIV reads to call a partial consensus, and the assembler just failed to build any contigs out of them. Because of this possibility, in place of the alignment-of-contigs-to-existing-references argument that's given to `shiver_map_reads.sh`, you can supply a fasta file containing a single sequence: that sequence will be used as the reference for mapping, instead of a tailored one constructed from contigs. If that sequence is one of the existing references you provided at the initialisation step, `shiver` knows how to do the coordinate translation necessary to produce a `SID_MinCov_X_Y_ForGlobalAln.fasta` file; if not, this file will not be produced (you'll still get your consensus sequence though).

To script this kind of thing, you can just check whether the alignment of contigs to existing references exists for this sample: if not, choose a sequence to use as your reference to mapping. This would probably be most easily achieved by making a big look-up table before you start. (e.g. running the program kraken or kallisto on the reads for each sample, one can see which of the existing references has the largest number of reads attributed to it; one might obtain a non-null result even when no HIV contigs were assembled.) For example in the scripted usage above, after
`alignment=../CheckedContigAlignments/"$SID"_wRefs.fasta`, you could have

```
if [[ ! -f "$alignment" ]]; then
  # Reassign the $alignment variable to be a file containing
  # the reference previously chosen for this sample.
fi
```

10 Sample Reprocessing and Analysis

Individual steps from `shiver` can be run with stand-alone command line tools, for ease of reapplication elsewhere; these are contained in the `tools` subdirectory of the `shiver` code directory. For example `CorrectContigs.py` is run with a file of contigs and a file of their `BLASTN` hits to some set of references, and corrects the contigs by cutting and reverse-complementing where needed. Also included in `shiver` are command-line tools for easy analysis and modification of sample output without rerunning the whole pipeline:

- Recall `X` and `Y` above – the coverage thresholds for calling bases. You can generate another consensus sequence using different values, `X2` and `Y2` say, without rerunning the whole pipeline, thus:

  ```
  $ tools/CallConsensus.py SID_BaseFreqs.csv X2 Y2 > MyNewConsensus.fasta
  ```

  If you decrease the value of `X` you will see more bases and fewer `?` characters: the less strict you are with your requirement on 'vertical' coverage (the number of mapped bases at a point; the height at a given position in the bam file, if you like), the more 'horizontal' coverage (i.e. genomic coverage, the total number of bases called in the consensus) will increase. That's good right? Yes and no. You could also increase your horizontal coverage by replacing all ? characters by a random selection of As, Cs, Gs and Ts, but it's not recommended. Decreasing these thresholds increases sensitivity to HIV reads at the cost of specificity (since some reads are bound to be contaminants), and you ought to balance these, not just maximimise the former. For example with dated sequences, you could try to get the strongest correlation between real time and the evolutionary distance inferred from a phylogeny – the R^2 of the molecular clock – since too little real sequence and too much contaminant sequence will both screw up your phylogeny. To regenerate a coordinate-translated version of this consensus for the global alignnent (of all consensuses produced by shiver), `tools/TranslateSeqForGlobalAln.py` can be run, taking as its two arguments the consensus, and `SID_coords.csv` generated by the full run of shiver.

- Another parameter in the configuration file is the minimum read *identity* – the fraction of bases in the read which are mapped and agree with the reference – required for a read to be considered mapped, and so retained in the BAM file. If you wish to increase this after completion of `shiver`, reads with an identity below your new higher threshold can be discarded by running `RemoveDivergentReads.py` on a BAM file. Running `shiver_reprocess_bam.sh` on the resulting BAM file (or indeed any BAM file) implements just the last steps in `shiver`, namely generating pileup, calculating the base frequencies, and calling the consensus.

- `FindNumMappedBases.py` calculates the total number of mapped bases in a BAM file (where read length is constant this equals the number of mapped reads multiplied by read length, minus the total length of sequence clipped from reads), optionally binned by read identity. In the absence of mapped contaminant reads, and all else being equal, mapping to a reference which is closer to the true consensus should map more bases and mapped reads should have higher identities.

- `FindClippingHotSpots.py` counts, at each position in the genome, the number and percentage of reads that are clipped from that position to their left or right end. Having many such reads is a warning sign of the kind of biased loss of information discussed in the `shiver` paper.

- `FindSubSeqsInAlignment.py` finds the location of specified subsequences in an alignment.
- `LinkIdentityToCoverage.py` finds, for each different coverage encountered when considering all positions in a BAM file, the mean read identity at such positions. The mean read identity tends to be lower at positions of low coverage due to a background of contaminant reads, which differ from the reference by virtue of being contamination, but which are nevertheless similar enough to be mapped. Quantifying the decline in identity at low coverage helps inform what coverage threshold is appropriate for a given data set.
- `AlignMoreSeqsToPairWithMissingCoverage.py` allows more sequences to be added to a pairwise alignment in which one sequence contains missing coverage (such as a consensus and its reference), correctly maintaining the distinction between gaps (indicating a deletion) and missing coverage.
- `AlignBaseFreqFiles.py` aligns not two sequences, but two of the csv-format base frequency files output by `shiver`. Optionally a similarity metric is calculated at each position in the alignment, between 0 (no agreement on which bases/gaps are present) and 1 (perfect agreement on which bases/gaps are present and on their proportions). This allows comparison not just of consensus sequences between two samples but also of minority variants.
- `ConvertAlnToColourCodes.py` converts each base in a sequence alignment into a colour code indicating agreement with the consensus and indels; `AlignmentPlotting.R` takes such colour codes and visualises the alignment. These scripts were used to produce the alignment + coverage + gene diagrams in the `shiver` paper.
- Finally some simple tools for convenience: `FindSeqsInFasta.py` extracts named sequences from a fasta file, with options including gap stripping, returning only windows of the sequences, and inverting the search; `PrintSeqLengths.py` prints sequence lengths with or without gaps; `SplitFasta.py` splits a fasta file into one file per sequence therein.

References

[1] C. Wymant, F. Blanquart, A. Gall, M. Bakker, D. Bezemer, N. J. Croucher, T. Golubchik, M. Hall, M. Hillebregt, S. H. Ong, J. Albert, N. Bannert, J. Fellay, K. Fransen, A. Gourlay, M. K. Grabowski, B. Gunsenheimer-Bartmeyer, H. Günthard, P. Kivelä, R. Kouyos, O. Laeyendecker, K. Liitsola, L. Meyer, K. Porter, M. Ristola, A. van Sighem, G. Vanham, B. Berkhout, M. Cornelissen, P. Kellam, P. Reiss, and C. Fraser, "Easy and accurate reconstruction of whole hiv genomes from short-read sequence data," *bioRxiv*, 2016.

[2] H. Li, B. Handsaker, A. Wysoker, T. Fennell, J. Ruan, N. Homer, G. Marth, G. Abecasis, R. Durbin, and . G. P. D. P. Subgroup, "The sequence alignment/map (sam) format and samtools," *Bioinformatics*, 2009.

[3] K. Katoh, K. Misawa, K. Kuma, and T. Miyata, "Mafft: a novel method for rapid multiple sequence alignment based on fast fourier transform," *Nucleic Acids Research*, vol. 30, no. 14, pp. 3059–3066, 2002.

[4] A. M. Bolger, M. Lohse, and B. Usadel, "Trimmomatic: a flexible trimmer for illumina sequence data," *Bioinformatics*, vol. 30, no. 15, pp. 2114–2120, 2014.

[5] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403 – 410, 1990.

[6] H. Li and R. Durbin, "Fast and accurate long-read alignment with burrowswheeler transform," *Bioinformatics*, vol. 26, no. 5, pp. 589–595, 2010.

[7] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg, "Ultrafast and memory-efficient alignment of short dna sequences to the human genome," *Genome Biology*, vol. 10, p. R25, Mar 2009.