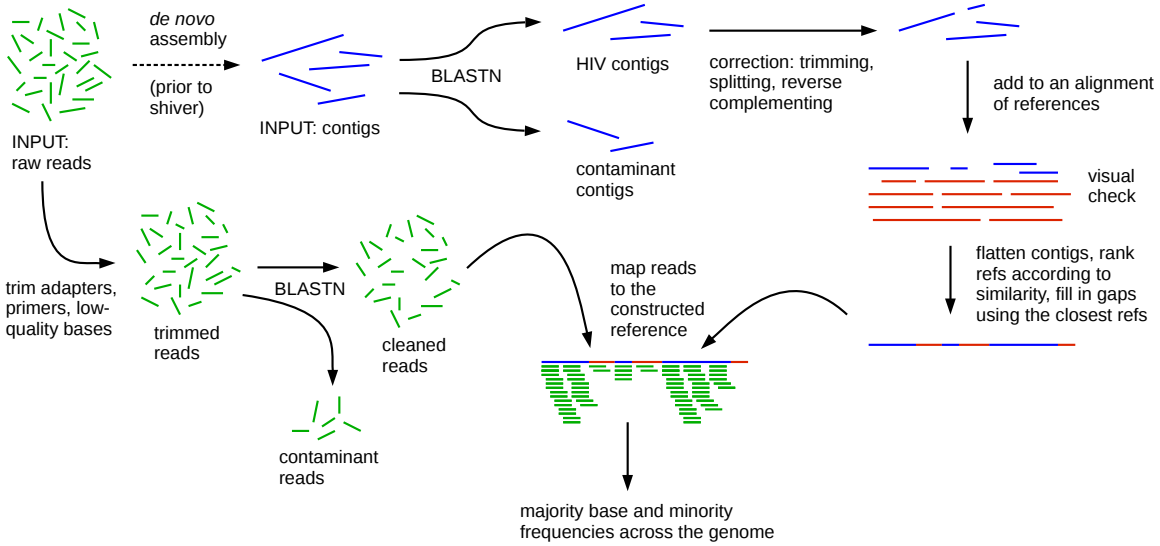
shiver

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

By Chris Wymant

This manual last updated February 8, 2024



`shiver` is a tool for mapping reads (fragments of genetic sequence) to a custom reference sequence constructed using *de novo* assembled contigs, in order to minimise the biased loss of information that occurs from mapping to a 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 applicability; 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 here; please cite this if you find `shiver` helpful. If you use `shiver`, please also cite the publications of its dependencies: `SAMtools` [1], `MAFFT` [2], `Trimmomatic`[3], `blast` [4], and `BWA` [5] or `bowtie` [6] 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.

`shiver` is run 'through the command line' (or 'in a terminal'), not by clicking on things in a graphical user interface. If you don't know how to run programs from the command line, you need to learn that first[1].

`shiver` runs natively on Linux and Mac OS, but not Windows. To install the dependencies of `shiver` follow these instructions. Once the dependencies are installed, to run `shiver` you only need to download its code from the GitHub repository; after download no installation is required. If you have VirtualBox installed, you can run this Linux Ubuntu 16.04.03 virtual machine, which contains `shiver` and all its dependencies. (A virtual machine is basically a self-contained copy an operating system and some extra software, which you can run on another operating system (including Windows). This virtual machine

[1] These resources were recommended by the organisers of the Wellcome Genome Campus' *Genomics and Clinical Virology* course: http://rik.smith-unna.com/command_line_bootcamp, http://www.ee.surrey.ac.uk/Teaching/Unix, http://swcarpentry.github.io/shell-novice/.

also contains our separate tool phyloscanner, which allows you to infer and analyse phylogenies using mapped reads, for example those produced by shiver).

To update your shiver code, in a terminal you need to change directory to where your shiver code lives, and run the command git pull.

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 occurrence of ~/shiver/ in this manual by my/other/directory/shiver/; alternatively if you're feeling lazy, create a symbolic link to my/other/directory/shiver, call it shiver, and put it in your home directory.

By default, to run a shiver command, you have to tell the command line where the binary/executable file for that command is found, i.e. including its path, e.g. ~/shiver/shiver_align_contigs.sh. If you don't want to type the path every time – you want to be able to just type shiver_align_contigs.sh– you need to add the directory containing the shiver code (~/shiver/) to your PATH environmental variable. Search on Google if you don't know how to do that. **Do not** add 'sh ' before a shiver command – this forces your operating system to interpret the code as shell language (it's actually bash) and will result in a strange error.

Throughout this manual, a $ character at the start of a line just indicates that what follows should be run from command line; you should be able to just copy and paste this into your terminal, then press enter to execute the command. Don't include that initial $ character in what you copy and paste. Be sure to highlight the whole command for copying and pasting. If a command is split over multiple lines, you can copy all of them in one go then press enter.

Contents

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:

- A `shiver` configuration file; one of these with default values is `~/shiver/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. The config file is discussed further in section 4.
- An alignment of your choice of existing reference genomes (lots of which are available to download from the LANL HIV database) called `MyRefAlignment.fasta`, say.
- Fasta files containing the adapters and PCR primers used for sequencing, called `MyAdapters.fasta` and `MyPrimers.fasta`, say. Adapters will be removed using Trimmomatic, and "The naming of the various sequences within this file determines how they are used" – the Trimmomatic manual. Adapter sequence is trimmed out of reads, wherever it is found; afterwards PCR primer sequences are trimmed when perfect matches to them are found at the end of a read.

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

```
$ ~/shiver/shiver_init.sh  MyInitDir  config.sh  MyRefAlignment.fasta  \
  MyAdapters.fasta  MyPrimers.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. Sample processing writes and reads in again (many) files in the working directory, so **it's very important work in a separate, empty directory for each sample you process**. Otherwise you might overwrite existing files, or the files from processing one sample could interfere with the files from processing another sample. There's an example of creating a new directory for each of a series of samples in section 7.

2 How to process a single sample

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

1. As of version 1.6.0, shiver can handle both *paired* reads (each fragment is sequenced forwards from one end and backwards from the other) and unpaired reads (one read is generated per fragment).
   - If your data is paired reads, 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.
   - If your data is unpaired reads, there are no restrictions on read names.
2. Our favourite 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.

Processing a sample requires two commands. If your contigs are in a file `contigs.fasta`, the first command 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 9 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. In this alignment, nothing has been done to the contigs (hence the name *raw*).

`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 it) 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;
- overhangs the edge of the alignment of existing references (i.e. in the alignment the beginning of the contig is to the left of the beginning of all references, or the end of the contig is to the right of the end of all references).

Appropriate action is taken in each case – trimming, cutting into pieces, 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. Therefore both of these files should be inspected, the better one chosen, and edited if necessary; we go into a lot more detail about this in section 11.

With an alignment of contigs you're happy with, say `SID_cut_wRefs.fasta`, and if your paired reads are in files `reads_1.fastq.gz` and `reads_2.fastq.gz`, run

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

or if you have unpaired reads, replacing `reads_1.fastq.gz reads_2.fastq.gz` by your single file of reads.

3 What output do I get?

Output files include:

- `SID.bam`, `SID_ref.fasta`: the bam file of mapped reads, and the reference sequence to which they were mapped.
- `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 this coordinate is ambiguous. Specifically, the extra coordinate is position in the *global alignment* (see section 8), and the rows that are skipped are positions where the reads had an insertion with respect to the mapping reference. Positions where the mapping reference has an insertion with respect to the global alignment (i.e. where the contigs had an insertion with respect to the existing references you gave `shiver` as input) are included at the appropriate position in this file, just with an undefined global coordinate. `SID_BaseFreqs_WithHXB2.csv` is the same thing as `SID_BaseFreqs_ForGlobalAln.csv` but using the coordinates of the HXB2 sequence instead of the coordinates of the global alignment (this is only produced if the config file variable `GiveHXB2coords` is left at its default value of `true`; you'll certainly want to change this to `false` if you're using `shiver` on non-HIV data).

- `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 when multiple samples are processed, so we postpone their explanation to section 8.

- `SID_InsertSizeCounts.csv`: the insert-size distribution.

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 are the result of remapping to the consensus called from the first round of mapping. The files `SID[something]` and `SID_remap[something]` will likely be very similar, but the latter is expected to be slightly more accurate. Files related to the global alignment do not have a remapped counterpart (see the final paragraph of section 8 for the reason why).

Some files are not produced by default, but their production can be specified in `config.sh`:

- `SID_PreMapping_1.fastq`, `SID_PreMapping_2.fastq`. These are produced if you change the value of the config file variable `KeepPreMappingReads` to `true`. They are the reads after removal of (i) adapters, (ii) PCR primers, (iii) low-quality bases, and (iv) those reads suspected of being contamination based on blasting to the contaminant contigs. This corresponds to the state of the reads right before the mapping step. Note that not all of these reads will map, for example because the removal of suspected contaminant reads based on the contaminant contigs is not perfectly sensitive (some contaminant reads may not have been assembled into contigs). So if what you're after is the fully processed reads with as many contaminant reads removed as possible (e.g. to upload to some public database), you don't want these files, you want the reads from the bam file. (By default these files are produced, but with `temp_` prepended to their names to indicate that you probably don't want to keep them.)

- `SID_MinCov_X_Y_wContigs.fasta`. This is produced if you change the value of the config file variable `AlignContigsToConsensus` to `true`. It is 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). This is just for interest, to see how the contigs compare with the consensus.

- `SID_ContaminantReads.bam`. This is produced if you change the value of the config file variable `MapContaminantReads` to `true`. It is a bam file of only the contaminant reads mapped to the HIV reference; it shows you the contaminant reads that would have been in your bam file of genuine reads if you had turned off the contaminant-contig-based cleaning of reads.

Lots of intermediate/temporary files are produced during processing; these you probably don't want to keep. By default their names all begin `temp_` (you can change this in the config file if you really want). If you

- chose a sample ID (for labelling output) that was not `temp` and did not begin with `temp_`, and

- ran `shiver` from inside an empty directory as strongly advised, and

- are still inside the same directory where you ran `shiver`, and

- have not added any other files with names beginning `temp_` since running `shiver`,

then the command `rm temp_*` will delete all and only the temporary files. (Warning: the `rm` command irrecoverably deletes files, and the wildcard character `*` matches *anything*. If you accidentally put a space between `temp_` and `*`, the previous command would delete *all files* in the current directory. If the `rm` command makes you nervous, you can delete files by selecting them in a graphical file explorer window and clicking delete (probably, depending on your operating system).)

4 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 (some new features required 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: rename `config.sh` to `config_original.sh`, make 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 (by running `git pull` inside `~/shiver/`), by running

```
$ diff config_original.sh config.sh
```

you can see whether the update changed the config file. Specifically, if that `diff` command produced no output, nothing has changed, and you can simply delete the new `config.sh` file that was created by the code update. If that `diff` command did produce some output, then the config file has changed as part of the update; in this case, running

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

will remind you of the changes you previously made to default parameter values. You can then do the same thing as before: rename the (updated) default config file, and make a copy in which you which you implement your desired changes to default parameter values.

If something I said above was confusing, don't struggle – let me know and I'll clarify.

5 The Fully Automatic Version Of `shiver`

As we discussed in section 2.5 of the supplementary material of the publication, it is possible to run an alternative version of `shiver` that proceeds from the start to the end without the break in the middle. This is done using the command `~/shiver/shiver_full_auto.sh`, but it is not recommended. Using `~/shiver/shiver_align_contigs.sh` followed by `~/shiver/shiver_map_reads.sh` is better.

6 Example Input Data Included With `shiver`

Inside your `shiver` directory is the `ExampleInput` subdirectory, containing what its name suggests:

- `MysteryHIV_1.fastq` and `MysteryHIV_2.fastq`: forwards and reverse reads simulated started from a mystery HIV sequence. There are unrealistically few reads compared to real Illumina data; the idea was to keep file sizes small.
- `MysteryHIV_contigs.fasta`: contigs that I have created manually, that illustrate `shiver`'s removal of contaminant contigs and correction of genuine contigs. `~/shiver/shiver_full_auto.sh` will refuse to process these, because the contigs require correction and it only proceeds to completion when the contigs require no correction. `MysteryHIV_contigs_SuitableForFullAuto.fasta` contains contigs that do not require correction, and can be processed by `~/shiver/shiver_full_auto.sh`.
- `adapters_Illumina.fasta`: default Illumina adapters. **Do not** assume these are what's needed for your sequence data – instead ask your sequencing team to provide the adapter sequences they used.
- `primers_GallEtAl2012.fasta`: the PCR primers of reference [7] (which were used for sequencing our data). **Do not** assume these are what's needed for your sequence data – instead ask your sequencing team to provide the PCR primer sequences they used.

With these input files and an alignment of existing references (e.g. downloaded from the LANL HIV database) you can run the commands of section 2. Visually inspecting the files `SID_raw_wRefs.fasta` and `SID_cut_wRefs.fasta`, can you see and understand what `shiver` has done to the contigs? (If not, section 11 talks you through lots of examples.)

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 file names 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 names suggest. 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 for this sample from the filename by removing the extension and 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 simpler
# to find the file we want for each sample.
$ 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;
  # here I'm trying to use minimal code for illustration.
  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

For each sample, `shiver` constructs a mapping reference from the alignment of that sample's contigs to the set of existing references. `shiver` knows how the sample's consensus aligns to this mapping reference (recall that `SID_MinCov_X_Y.fasta` is an alignment of these two sequences), and it knows how the mapping reference aligns to the set of existing references. Because the same set of existing references is used for each sample, `shiver` can figure out by translating alignment coordinates and how the mapping references of any two different samples align to each other, and thence how their consensuses align to each other. That's what the aforementioned `SID_MinCov_X_Y_ForGlobalAln.fasta` file is: the consensus with gaps inserted into it judiciously such that it's in the same set of coordinates for every sample. These coordinates are the same as those of the alignment of existing references you gave `shiver` as input, unless that alignment contained any positions that were gaps for every reference, in which case such positions were removed.

Constructing a global alignment is then achieved just by putting all those files into one file:

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

and you can include the existing references too, if you like, with

```
$ cat MyInitDir/ExistingRefAlignment.fasta >> GlobalAln.fasta
```

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

In order to translate a consensus into the global alignment coordinates, two kinds of insertion must be excised: insertions of the consensus with respect to the mapping reference (which should be rare, because the mapping reference was tailored to be a best guess for what the consensus would turn out to be), and unique insertions in the contigs not seen in any of the existing references (which should be rare if a large and diverse set of references were given as input). It's not possible to figure out how such positions align

to each other just by translating coordinates. They're absent in the global alignment, but are still present in the individual consensus files (`SID_MinCov_X_Y.fasta`), which you can align to each other using e.g. `mafft` if desired.

Note that the consensus whose positions are translated for the global alignment is the consensus from `shiver`'s first round of mapping (to a reference constructed from contigs with gaps filled in by existing references), not from `shiver`'s second round of mapping (to the consensus called in the first round; done only if `remap=true` in the config file). The reason for this, if you're interested, is that the alignment of the first-round mapping reference to the existing references is known (`shiver` can figure it out from the alignment of contigs to existing references) but the alignment of the second-round mapping reference is not.

9 Handling missing data

`shiver` will exit prematurely in the event of a problem, as you would hope. An exit status of 3 is reserved for situations where there is an absence of data to work with, rather than an error as such; in a big heterogeneous data set some samples may contain no HIV. `shiver_align_contigs.sh` will return an exit status of 3 if and only if there are no contigs after extraction of those thought to be HIV and contig correction (which may decrease the length of contigs below a threshold for inclusion). `shiver_map_reads.sh` will exit with status 3 if and only if:

- the reads file (or files, for paired reads) is empty at the start, or
- the reads file (or files, for paired reads) is empty after trimming, or
- the reads file (or files, for paired reads) is empty after removal of reads that blast best to contaminant contigs, or
- the bam file of mapped reads is empty.

Any other problems that prevent `shiver` from running to satisfactory completion should result in an exit status of 1.

For samples devoid of HIV contigs, `shiver_align_contigs.sh` will exit with status 3 as discussed and will not produce a `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 that a partial consensus could be called, but 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 one constructed from the 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, that file will not be produced, but you'll still get the `SID_MinCov_X_Y.fasta` file.

To script this kind of thing, you can simply 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. (Be careful to follow up any errors from `shiver_align_contigs.sh` that could result in an absence of this file for another reason!) 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
  # Insert code here to re-assign 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 `blast` hits to some set of references, and corrects the contigs by trimming, cutting into pieces 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 in your consensus: the number of positions where at least `X` reads were mapped increases as `X` decreases, trivially. However the point of having such a threshold is that contamination is generally present in the input read data, and may still be present in the reads after processing and mapping. Decreasing the coverage thresholds increases sensitivity to HIV reads at the cost of specificity, and you ought to balance these, not just maximise the former. The gold-standard way of figuring out how low you can safely set your coverage threshold without including many contaminant reads would be to sequence a set negative control samples and see what coverage they typically have, purely from contamination. (As always, negative controls must be obtained and processed in exactly the same way as the genuine samples to be meaningful, i.e. taking samples from people uninfected by HIV then processing as usual. Simply sequencing water, for example, would not capture the fact that blood contains other nucleic acids that can be mistaken for HIV.) Absent such information, another way to get a handle on this problem is provided by `LinkIdentityToCoverage.py`, described below. Yet another way, for dated sequences, might be to maximise the 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 alignment (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 sub-sequences 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. Let us make two assumptions: (i) when contaminant reads survive the steps of `shiver` to be present in the resulting BAM file, the coverage of these reads is not linked to the coverage of genuine reads (i.e. having twice as many genuine reads does not double the number of contaminant reads), and (ii) contaminant reads

are more different from the mapping reference than geunine reads[2]. These assumptions imply that at positions of lower coverage the background of contaminant reads makes up a larger fraction of what's there, and so the mean read identity will tend to be lower. In the data I have worked with, `LinkIdentityToCoverage.py` has shown that below a particular coverage value, mean read identity decreases, whereas above that value it is stable. This is what I have used for the coverage threshold, namely 30 for the Gall *et al.* protocol [7] with MiSeq and 300 with HiSeq, and 5 for the Bonsall, Golubchik *et al.* protocol [8].

- `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_ByConsensuses.py` and `AlignBaseFreqFiles_ByReference.py` align not two sequences, but two of the csv-format base frequency files output by `shiver`. This allows comparison not just of consensus sequences between two samples but also of minority variants; it also allows a comparison of sequences (e.g. counting differences) to be linked to coverage information.
- `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.
- `EstimateAmbiguousBases.py` estimates IUPAC ambiguity codes (e.g. M means A or C) in a set of aligned sequences by taking the most common of the possible bases at that position.
- `MergeAlignmentsToCsv.py` creates a csv-format multiple sequence alignment from a set of pairwise alignments that all share one sequence. Each position (i.e. of a single base) in the shared sequence defines one position in the merged alignment; what each of the other sequences has here could be a single base, a gap, or a kmer.
- `QuantifyPairwiseIndels.py` calculates, for each possible pair of sequences in a multiple sequence alignment, the size and position of the relative indels (i.e. taking their relative alignment from the overall alignment, ignoring positions at which both have a gap).
- Finally some simple tools for convenient command-line processing: `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; `AddAllPossibleSNPsToSeqs.py` outputs all possible sequences differing from input sequences by one base; `CheckFastaFileEquality.py` checks whether two fasta files contain the same sequences, ignoring formatting; `ExactBLAST.py` searches for exact sequence matches allowing for gaps; `FindNamedReadsInSortedFastq.py` efficiently searches for (sorted) reads in a (sorted) fastq file; `UngapFasta.py` removes gaps from sequences.

11 Examples of inspection & modification of contigs aligned to existing references

To quote the supplementary material of the `shiver` publication:

> For HIV sequences, reference [9] states that "Algorithmic alignment does not necessarily retrieve the best alignment. It is important to always verify whether the sequence data are aligned unambiguously and, if necessary, manually correct the alignment." Reference [10] echoes this for any evolving pathogen: "the 'best' alignment chosen by an alignment program is not necessarily the 'true' alignment... Alignment quality should also be inspected manually in a visualisation program". The commonness of indels in HIV makes alignment more difficult,

[2] This requires that there aren't parts of the genome with more contaminant reads than genuine reads – because remapping to the consensus would then make the new reference look like the contaminant reads – except at positions below the coverage threshold, where no consensus is called and so the reads do not affect the reference for remapping.

as does the fact that the contigs may be an imperfect representation of the true sample even after correction. We used `Geneious` [11] for sequence visualisation and editing where needed.

As well as revealing alignment error, inspection of the aligned contigs allows the user to check for any remaining problems with the contigs. We suggest that in general the user inspects both the alignment of the existing references with the *raw* HIV contigs (before any correction by shiver), and the alignment of the existing references with the *corrected* HIV contigs, as a check that all `shiver`'s modifications of the contigs are desired. An example of when this might not be the case is when the sample contains an indel not observed in the existing reference set, that is large enough to cause the contig to be split in two at that point, but which the user thinks might be genuine rather than an a misassembly (through previous/expert knowledge, or perhaps simply observing the same indel in multiple samples in a dataset). With sufficiently accurate mapping, reads will map here correctly whether or not the reference constructed from the contigs contains the indel, making the question moot; however with mapping inaccuracies of the kind shown in Figure 2 [from the `shiver` publication, not this document – the `shiver` manual] possible, it's best to get the reference's structure as correct as possible before mapping.

Recap: the alignment of contigs to references is used to construct a reference tailored to this sample by using contig sequence at parts of the genome where it is available, and filling in gaps *between* contigs using the reference that most closely resembles the contigs. Gaps *within* contigs are not filled in – they are taken to represent genuine deletions; this is why it is important to split a single contig into multiple pieces if alignment of those pieces would result in unrealistically large gaps separating them. At positions spanned by two or more contigs, the base (or gap) of the longest contig is used. This means if a problematic contig is wholly inside a longer contig, it does not cause any problems. (Gaps do not count towards determining the length of a contig.)

In the following pages I show some examples aligned raw and cut contigs to references from the data set of the project I work on (BEEHIVE), for discussion of what to look for. I used 181 references but only show a handful of them to keep the images a reasonable size. I've cut the sequence identifiers off the image, to keep everything unidentifiable; I've drawn a red line across each screenshot so that the contigs can be distinguished from the existing references (the contigs are above the line, the references below).

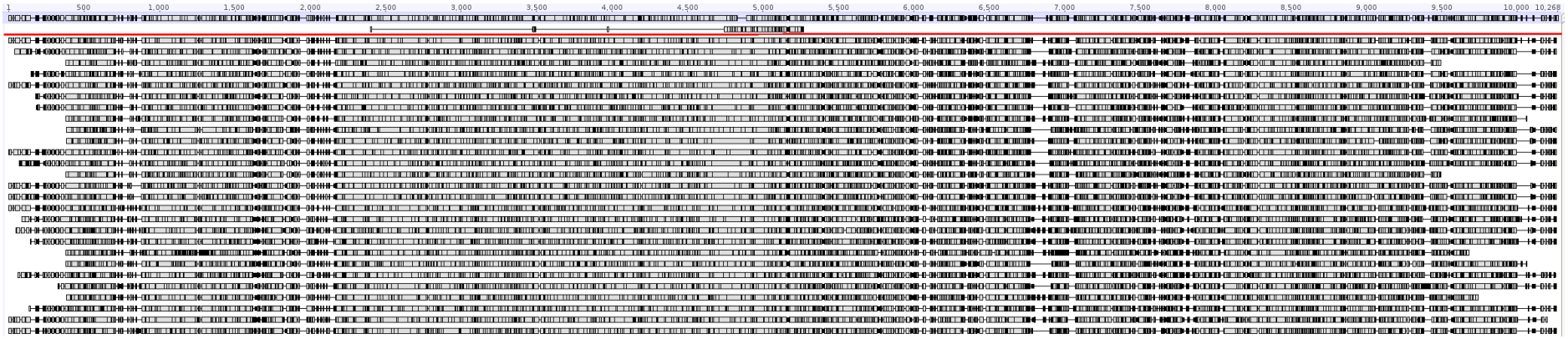
What do I mean when I describe contigs as 'looking good' or 'aligning nicely'? Roughly speaking, it means that if you were to look at the part of the alignment spanned by the contig (ignoring what comes before the contig starts and after it ends), you can't tell visually what's the contig and what are the references – they look the same, having comparable densities of SNPs (single nucleotide polymorphisms – base changes), and comparable indels (in terms of size and position). In these screenshots SNPs are indicated by black vertical lines within the sequence (c.f. grey parts of sequence that agree perfectly with whatever is most common here) and deletions are indicated by thin black horizontal lines inside the sequence.

11.1 Raw contigs bad, corrected contigs good

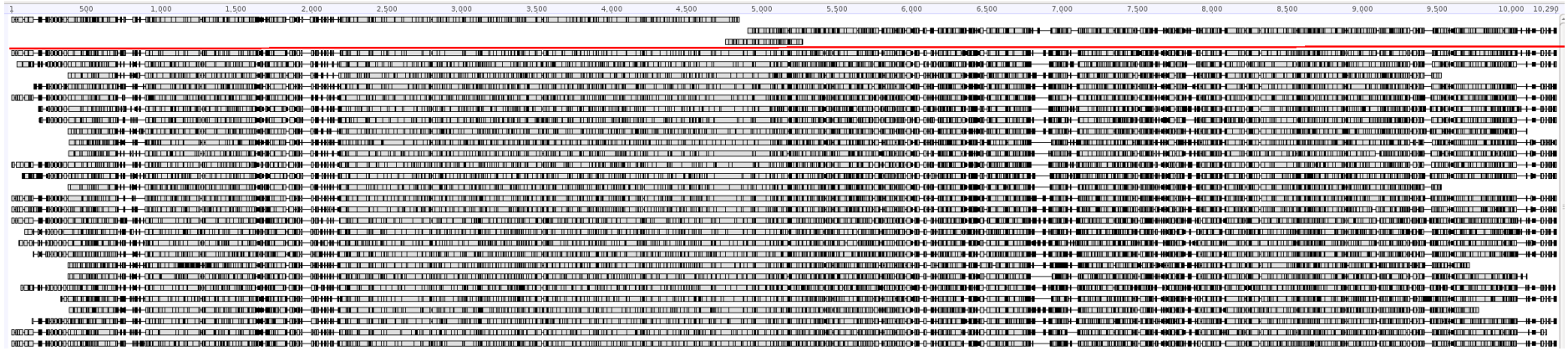
(a) Raw contigs and existing references

(b) Corrected contigs and existing references

Figure 1: The second raw contig aligns nicely from just before position 2000 to just before position 4000, then the rest of the contig after this looks like junk. Correction split the second raw contig into two; the part that looked like junk, when reverse complemented and aligned separately, now aligns nicely to positions 1 to 2500.

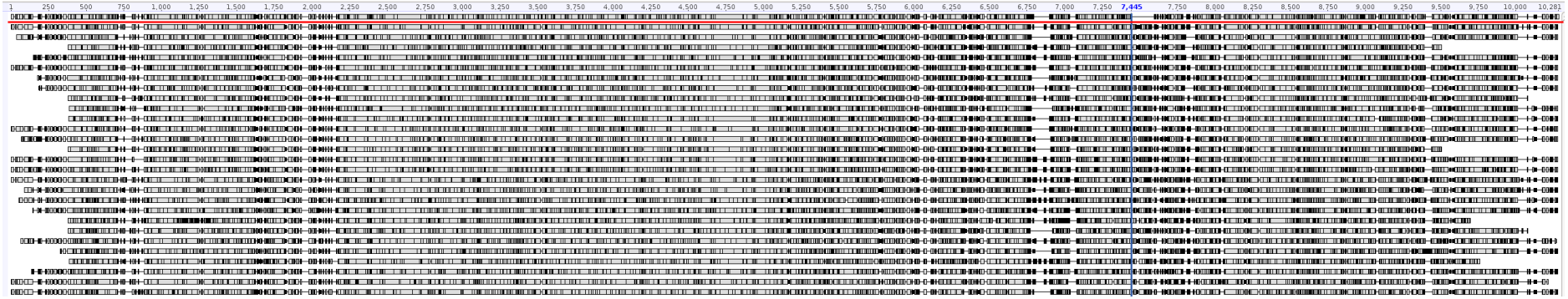


(a) Raw contigs and existing references

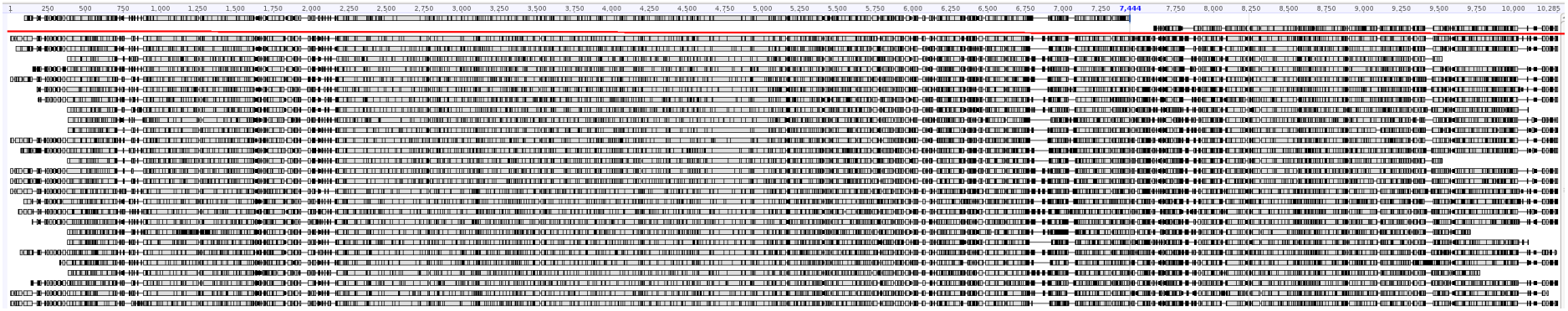


(b) Corrected contigs and existing references

Figure 2: The first raw contig has a deletion at a conserved part of the genome where the references don't have indels. The second raw contig has a few straggly bases on its left-hand side that have been badly aligned. Correction split the first raw contig into two at the unusual deletion, meaning that the gap between the resulting two contigs will be treated as missing and will be filled in by something else (in this case the other raw contig) instead of treated as a genuine deletion. Correction also deleted those straggly bases from the left-hand end of the second raw contig.

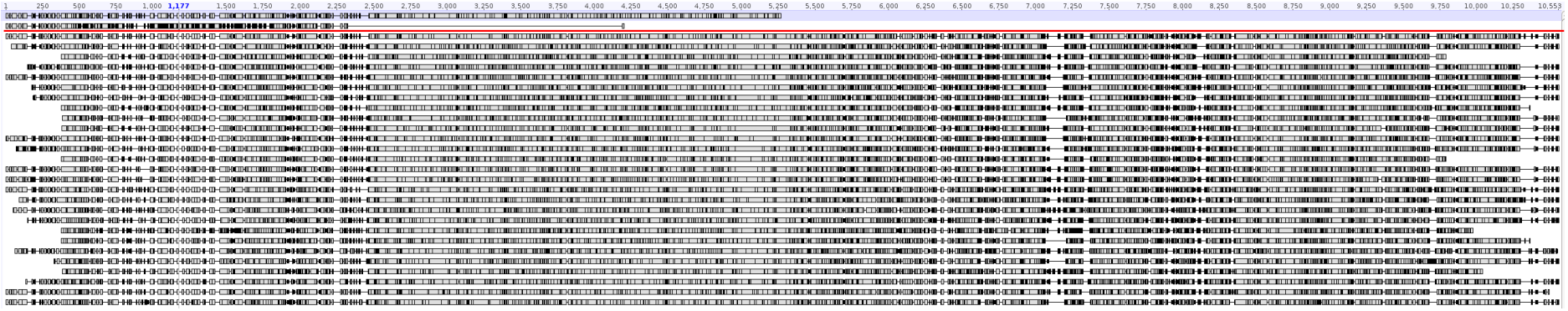


(a) Raw contigs and existing references

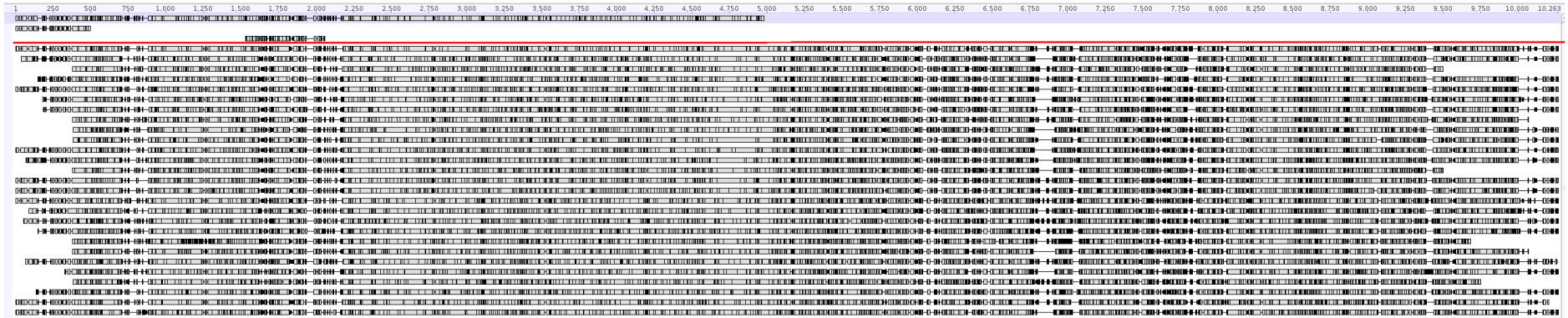


(b) Corrected contigs and existing references

Figure 3: The raw contig has a deletion at a part of the genome where indels are observed, but the deletion is considerably larger than any present in the reference: misassembly seems more plausible than a genuine deletion. Correction split the contig into two so that this gap will be filled with the sequence of the closest existing reference.



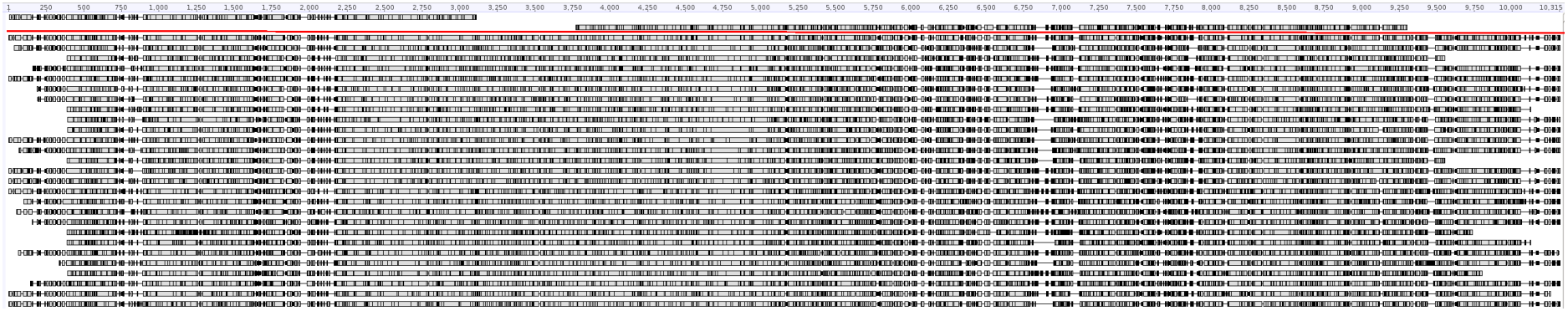
(a) Raw contigs and existing references



(b) Corrected contigs and existing references

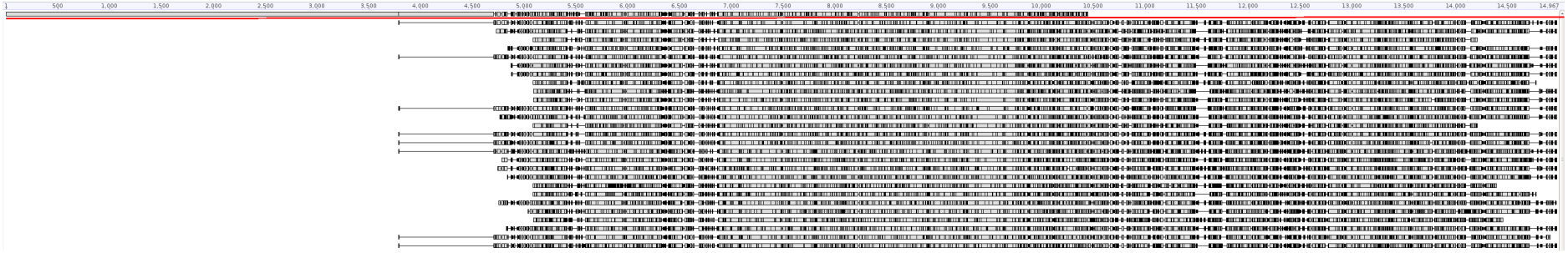
Figure 4: The second raw contig has two problems: its middle has an unusually high density of SNPs – it looks like junk – and its right-hand end has a few straggly bases that were badly aligned. Correction deleted both of these things, giving a nicer looking alignment.

(a) Raw contigs and existing references

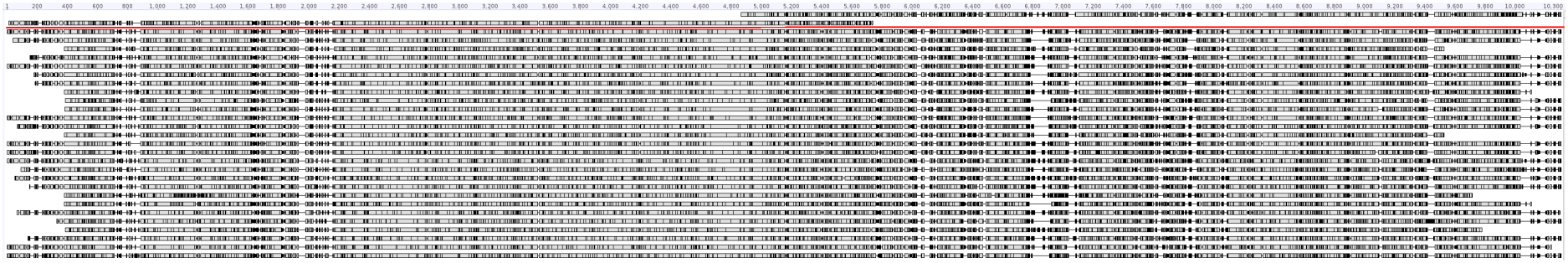


(b) Corrected contigs and existing references

Figure 5: The raw contig has a deletion so large that misassembly is more plausible than this being a genuine deletion. Correction split the contig in two so that the gap will be filled with sequence from the closest existing reference.

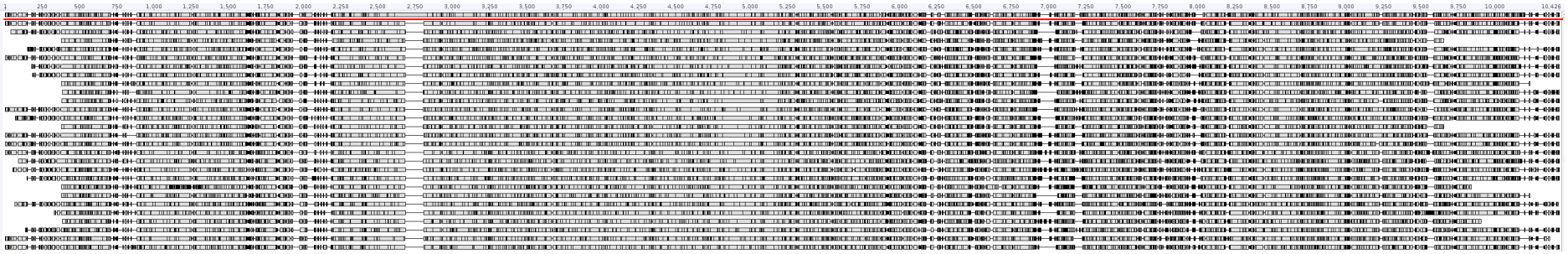


(a) Raw contigs and existing references

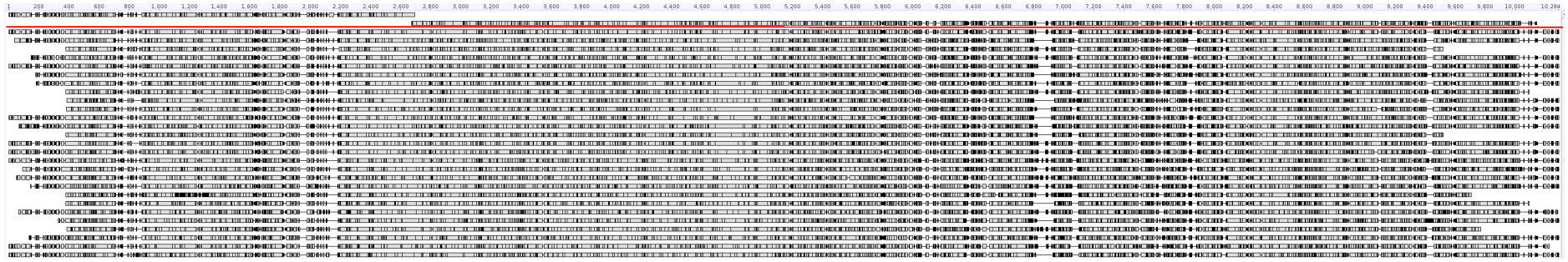


(b) Corrected contigs and existing references

Figure 6: The right half of the raw contig aligns nicely to the left half of the reference genomes; the left half of the raw contig is thus forced to just overhang the left-hand end of all the references. Correction split the contig in two, allowing its left half to be aligned independently, and we see it belongs on the right-hand side of the genome.

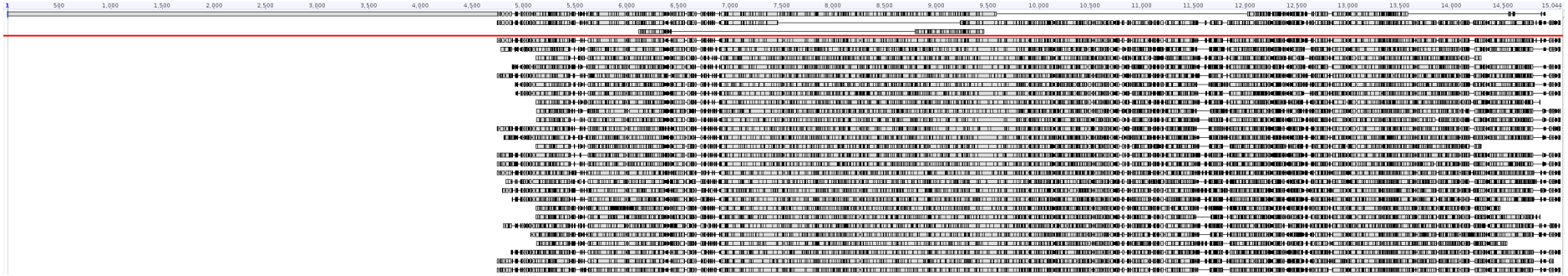


(a) Raw contigs and existing references

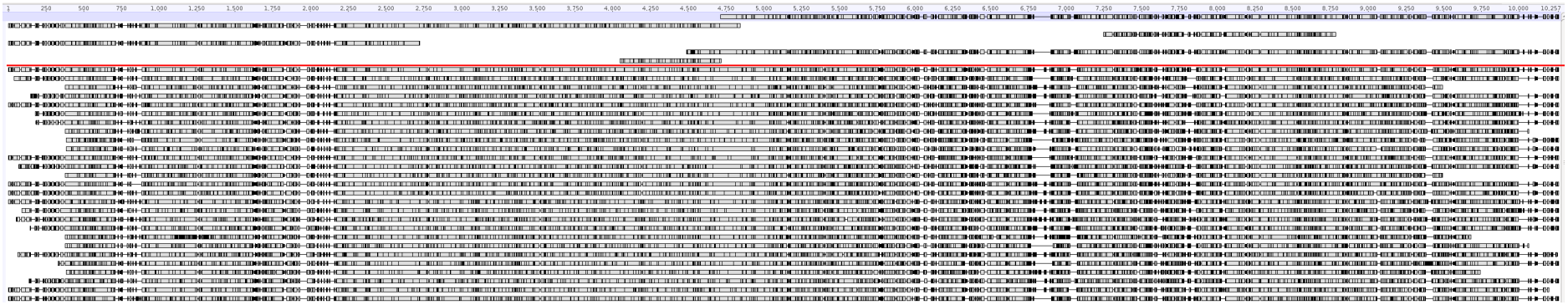


(b) Corrected contigs and existing references

Figure 7: The raw contig has a large insertion at a conserved part of the genome where none of the references have indels: one expects that this is a misassembly. Correction split the contig in two and deleted the insertion.

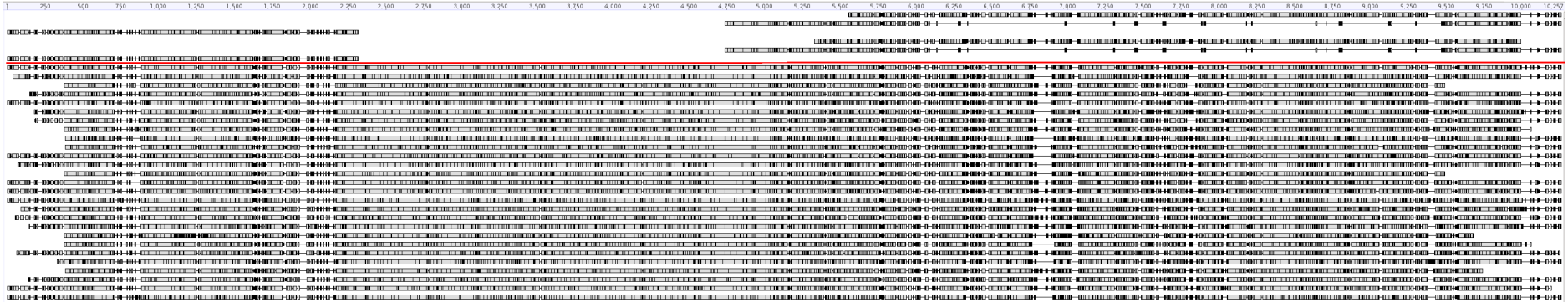


(a) Raw contigs and existing references

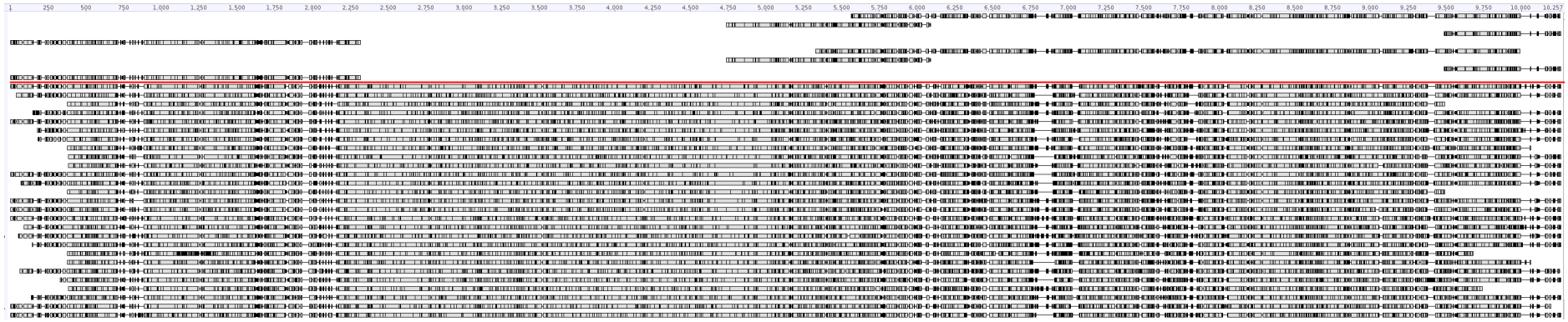


(b) Corrected contigs and existing references

Figure 8: Oh boy. The first raw contig has nearly 5000 bases of sequence extending out past the left-hand of the genome, then a stretch of nicely aligned sequence, then an unrealistically large deletion, then a stretch of nicely aligned sequence, then a little bit of badly aligned sequence. The second and third raw contigs both have unrealistically large deletions too. Correction (a) split contigs in two at unrealistically large deletions, (b) split off that large overhanging bit of the genome from raw contig one into a separate contig that aligned nicely to the right half of the genome, (c) trimmed off that little bit of badly aligned sequence from raw contig one, (d) deleted the left-hand part of the third contig (either because it didn't have a blast hit or because it did not pass the length threshold after being separated from the rest of the contig).

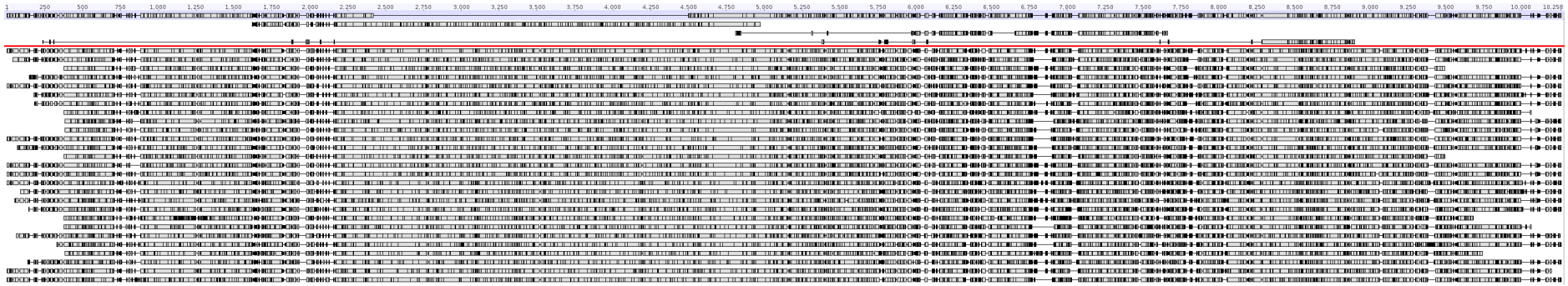


(a) Raw contigs and existing references

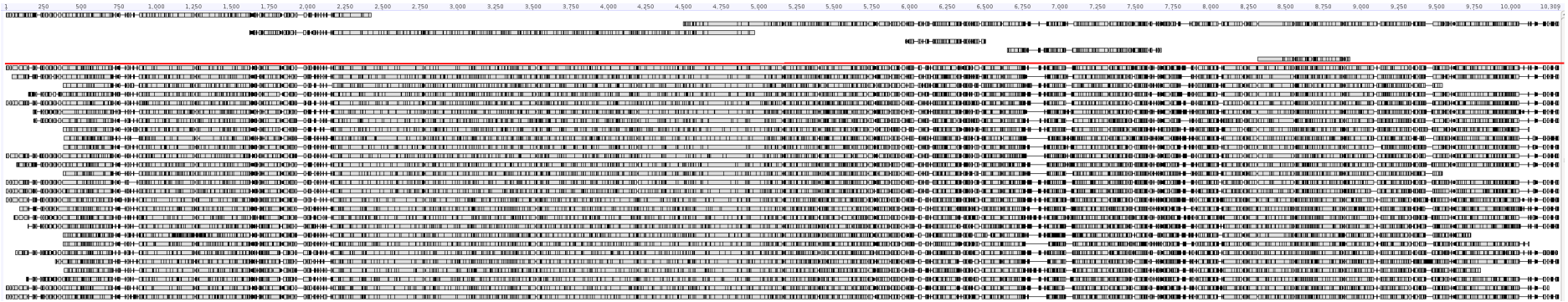


(b) Corrected contigs and existing references

Figure 9: Two of the raw contigs have the same problem: they look fine at either end, but those ends are separated by a big gap containing a small amount of badly aligned sequence. Correction splits the contig in two and removes those little bits of badly aligned sequence.



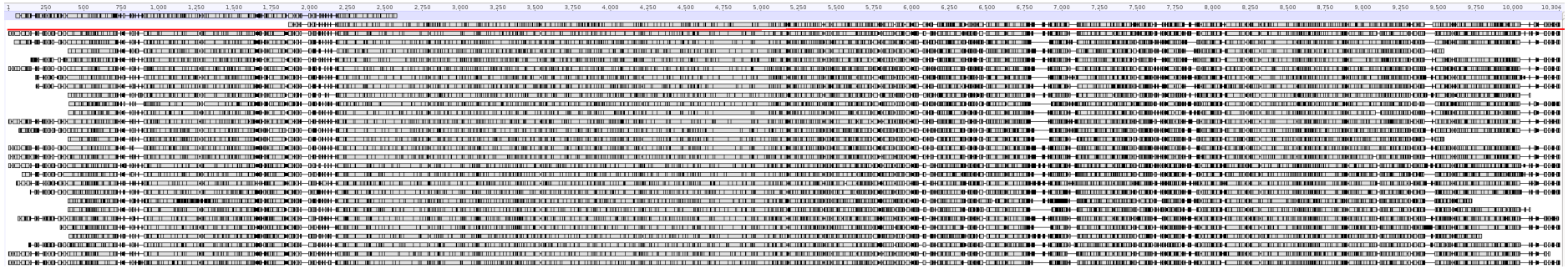
(a) Raw contigs and existing references



(b) Corrected contigs and existing references

Figure 10: Oh boy, again. The first raw and third contigs both have unrealistically large deletions; correction split them in two at the appropriate place. The third and fourth raw contigs both have a small number of badly aligned bases on their left-hand ends; correction deleted these.

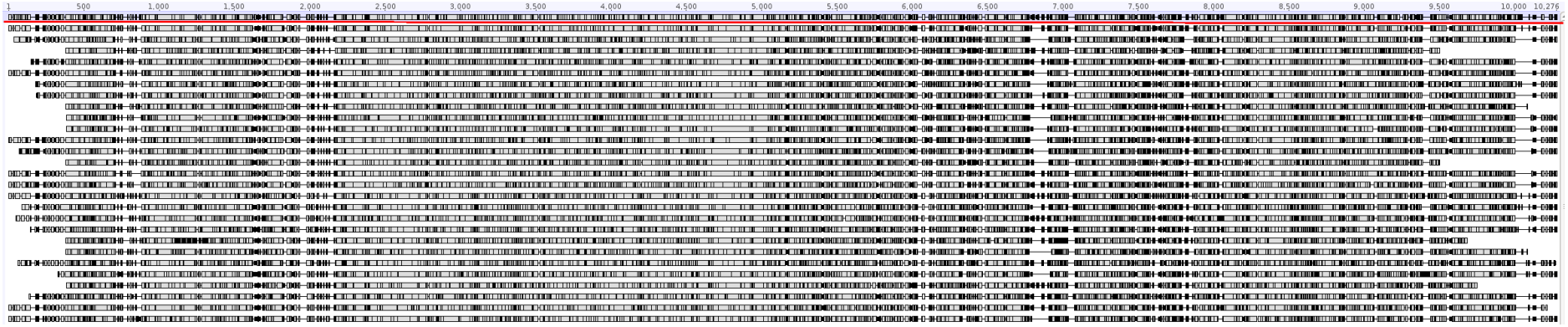
(a) Raw contigs and existing references



(b) Corrected contigs and existing references

Figure 11: That single raw contig looks fine from around position 3000 onwards, but it highly suspect before that: a dense collection of SNPs compared to the references, and an unrealistically large insertion. Correction split the raw contig in two, and reverse complemented the left-hand piece of contig: afterwards it aligns much more nicely.

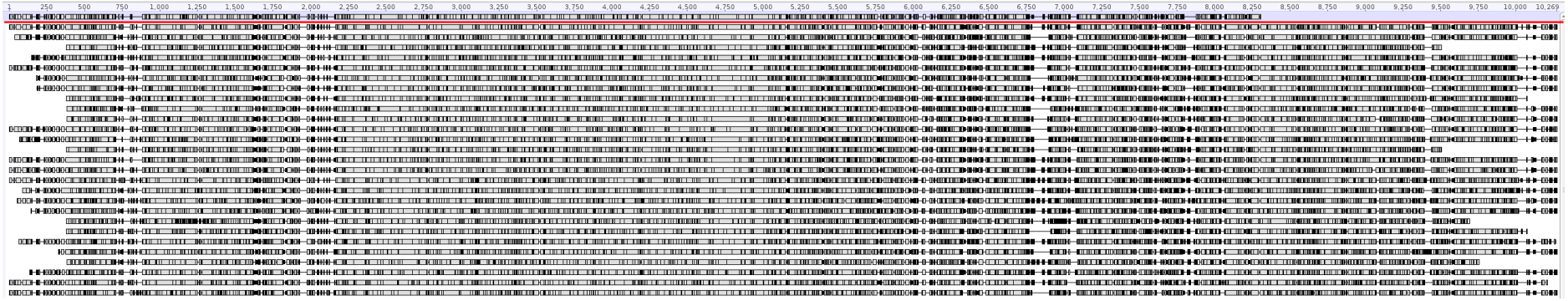
11.2 Raw contigs good, corrected contigs bad



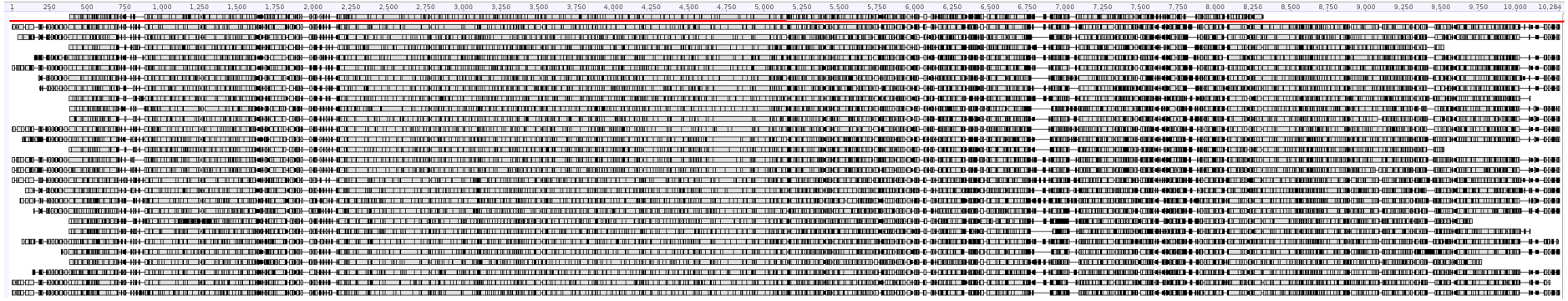
(a) Raw contigs and existing references

(b) Corrected contigs and existing references

Figure 12: Correction deleted the short bit of sequence at the hyper-variable region just before position 7000, and split the contig into to two. Given that the rest of the raw contig looked fine and the existing references are usually very different from each other, one expects that the hyper-variable sequence in the raw contig was probably fine and should not have been deleted. (Deleting it and splitting the contig in two means that the gap between the two corrected contigs would be filled in by the sequence of the closest existing reference.)

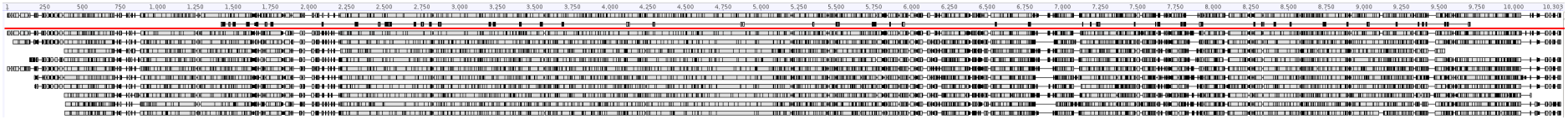


(a) Raw contigs and existing references

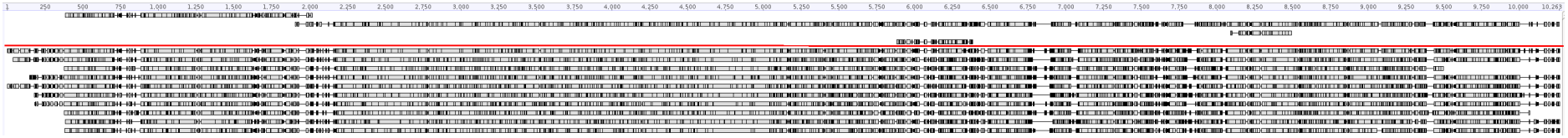


(b) Corrected contigs and existing references

Figure 13: The raw contig looks fine; correction trimmed off its left-hand end. This is because that bit of sequence was not present in the closest reference for this contig, and so was not spanned by the contig's blast hits to that reference. I hope to improve `shiver` in summer 2018 to prevent this problem.

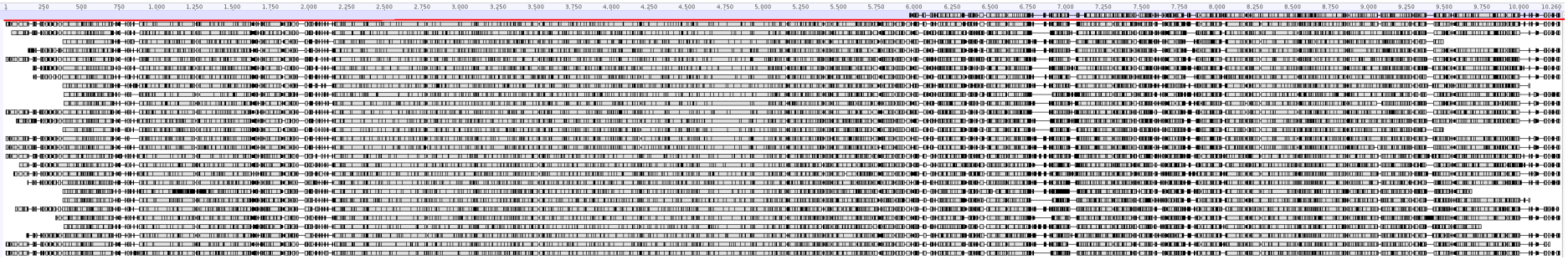


(a) Raw contigs and existing references

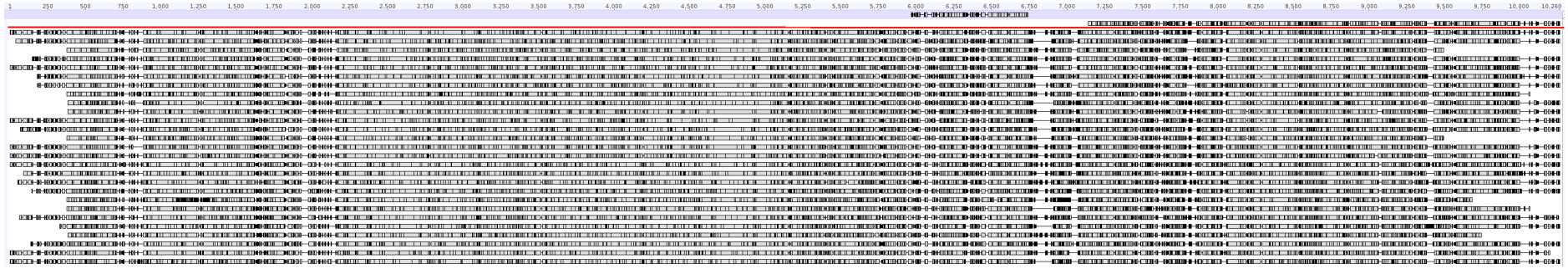


(b) Corrected contigs and existing references

Figure 14: The second raw contig looks to be total junk. Correction reverse complemented it, allowing it to be aligned well; however correction also (i) removed the left-hand end of the first raw contig, which seemed fine (see the caption of Fig. 13), (ii) split the first raw contig into two at around position 2000 due to an unusual insertion (recall that in all of these screenshots I'm only displaying a handful of the 181 references I used, to prevent a visual overload; however the fact that some of the positions at this part of the genome are gaps for all displayed sequences is indicative of some of the hidden references having insertions here, which is relevant). Given that the insertion is fairly small, at a part of the genome where indels are common, and inside a contig with no other obvious problems, forced to choose I would say it's more likely to be a genuine insertion than a misassembly. (A definitive answer would require mapping to a reference with the insertion and one without, and inspecting the reads.) So we want to keep the first raw contig without any modification. Since the bad (second) raw contig is wholly inside the good (first) raw contig, it will not be used for construction of the tailored reference, so we can simply use the raw alignment as it is and discard the cut alignment.

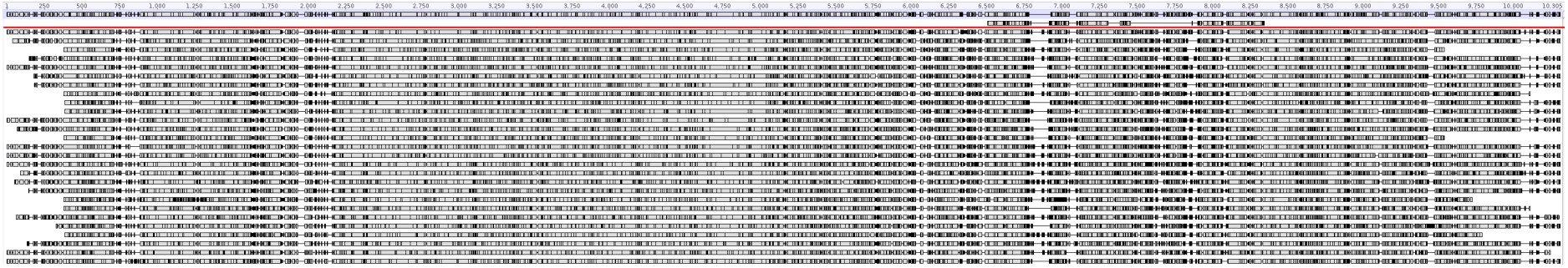


(a) Raw contigs and existing references

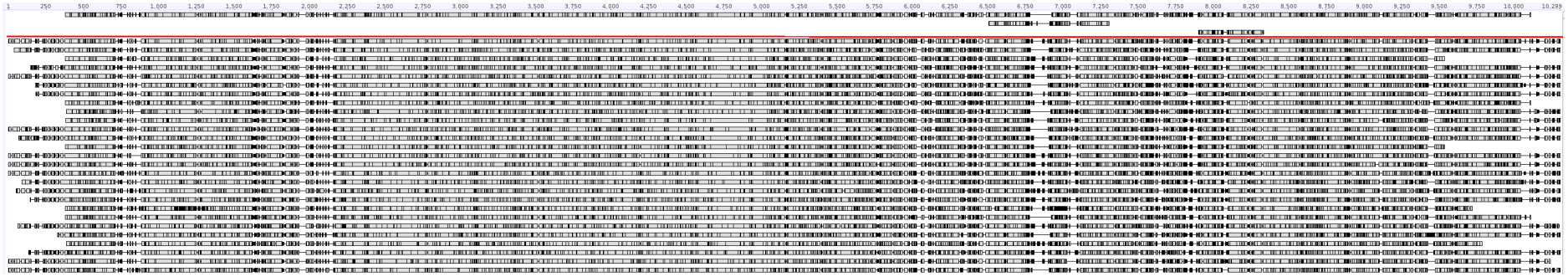


(b) Corrected contigs and existing references

Figure 15: Correction deleted the sequence inside the raw contig in positions 6750 – 7100, and split the contig into one piece either side of what was deleted (so that the deleted region is treated as missing information, not a genuine deletion). That bit of sequence was deleted because it did not blast to any of the existing references; however we can see that it is at a hyper-variable part of the genome, and only looks as different from the existing references as they are from each other. The contig also looks fine either side of this hyper-variable region – further suggestion that it is genuine sequence rather than misassembly. We want to use the raw contig rather than the cut ones.



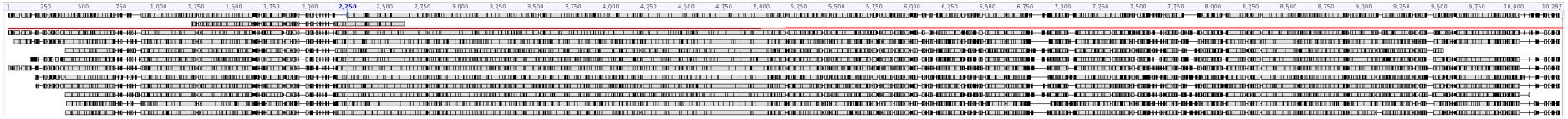
(a) Raw contigs and existing references



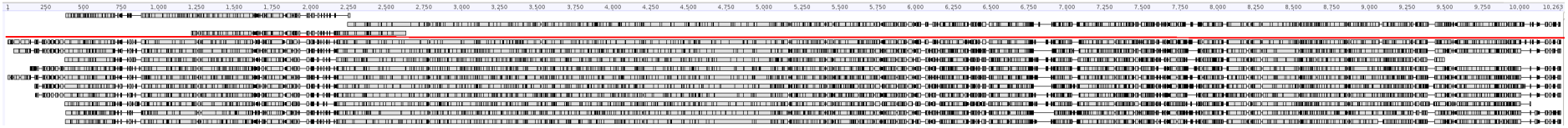
(b) Corrected contigs and existing references

Figure 16: Correction correctly split the second raw contig into two at the point of an unrealistically large deletion, and then deleted a few badly aligned bases from the right-hand end of the left-hand piece of contig following the split. However it also trimmed both ends of the first raw contig, which looked fine (see the caption of Fig. 13). Since that problematic second raw contig is wholly inside the fine-looking first raw contig, it will not affect construction of the tailored reference, and so we can safely use the raw alignment.

11.3 Raw contigs bad, corrected contigs bad



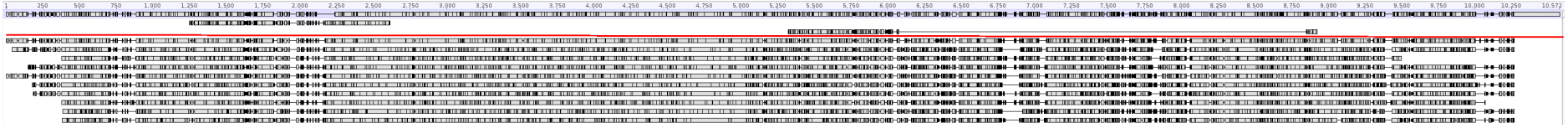
(a) Raw contigs and existing references



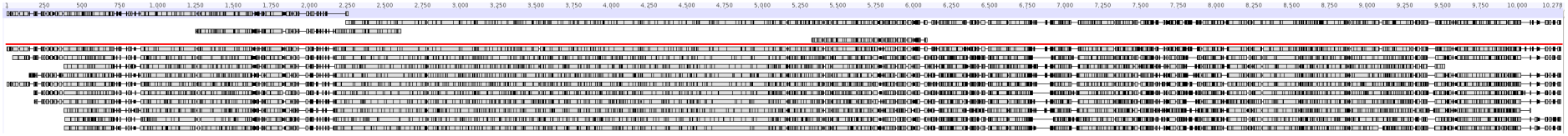
(b) Corrected contigs and existing references

(c) Manually corrected contigs and existing references

Figure 17: The first raw contig has a suspiciously large deletion just before position 2250, where all of the existing references have some sequence and even the second raw contig has sequence. If we used this alignment, that deletion would be treated as genuine and not filled in because it's inside the longest contig at that point. Correction nearly solved this problem, splitting the contig into two separate contigs, but at not quite the right place – the deletion is still there inside the first corrected contig because of a little bit of sequence aligned to the right of it. Also, correction removed some fine-looking sequence from the left-hand end of the first contig. So neither `SID_raw_wRefs.fasta` nor `SID_cut_wRefs.fasta` is ideal. To get an ideal alignment, I (i) created a duplicate of the `SID_raw_wRefs.fasta` file, (ii) created a duplicate of that first raw contig (by copying and pasting the text inside the file, absent a better way of doing this), (iii) deleted the sequence to the left of the large deletion in the first duplicate of the contig and deleted the sequence to the right in the second duplicate, using Geneious' ability to highlight and delete sequence in an alignment. The net effect of all this is to split the first raw contig in two at the point of the large deletion; that gap between the two resulting contigs will then be filled in using the sequence of the other contig.



(a) Raw contigs and existing references



(b) Corrected contigs and existing references

(c) Manually corrected contigs and existing references

Figure 18: Correction removed some sequence from the right-hand end of the first raw contig that was overhanging the end of the genome, and removed a short amount of badly aligned sequence from the end of the third raw contig. It also split the first raw contig in two at the point of an unrealistically large deletion, but as in the previous example it did not cut at quite the right place. Here this is easier to put right: manually deleting that little bit of badly aligned sequence at the end of the first corrected contig, so that the first two corrected contigs (the two parts that the first contig was split into) have a gap between them, which will be filled using the sequence of the other contig at this point.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

[7] A. Gall, B. Ferns, C. Morris, S. Watson, M. Cotten, M. Robinson, N. Berry, D. Pillay, and P. Kellam, "Universal amplification, next-generation sequencing, and assembly of hiv-1 genomes," *Journal of Clinical Microbiology*, vol. 50, no. 12, pp. 3838–3844, 2012.

[8] D. Bonsall, T. Golubchik, M. d. Cesare, M. Limbada, B. Kosloff, G. MacIntyre-Cockett, M. Hall, C. Wymant, M. A. Ansari, L. Abeler-Dörner, A. Schaap, A. Brown, E. Barnes, E. Piwowar-Manning, E. Wilson, L. Emel, R. Hayes, S. Fidler, H. Ayles, R. Bowden, and C. Fraser, "A comprehensive genomics solution for hiv surveillance and clinical monitoring in a global health setting," *bioRxiv*, 2018.

[9] A. Abecasis, A. Vandamme, and P. Lemey, "Sequence alignment in hiv computational analysis," *HIV Sequence Compendium 2006/2007*, 2007.

[10] K. McElroy, T. Thomas, and F. Luciani, "Deep sequencing of evolving pathogen populations: applications, errors, and bioinformatic solutions," *Microbial Informatics and Experimentation*, vol. 4, no. 1, pp. 1–14, 2014.

[11] *Geneious version 7.1 created by Biomatters. Available from http://www.geneious.com.*