Alvis Documentation

Richard Leggett, Samuel Martin

Sep 08, 2023

Contents:

CHAPTER 1

Installation

Alvis requires Java to run, and a LaTeX compiler to create PDFs from the tex files that Alvis produces, or a browser to view the svg files that Alvis produces. First, download the repository from https://github.com/SR-Martin/alvis.git.

A jar file is included in the repository, and is all that is necessary to run Alvis. This can be found at `/path/to/Alvis/dist/Alvis.jar`, usage instructions are in the *Usage* section.

If you wish to compile Alvis yourself, you may do so using the file `build.xml` found in the Alvis root directory, and ant, as follows. In the terminal, type:

```
cd /path/to/Alvis/
ant compile jar
```

Then, to run Alvis:

```
cd dist
Java -jar Alvis.jar
```

You can also build a linux executable directly (thanks to rsuchecki for providing this functionality):

```
cd /path/to/Alvis/
ant linux-bin
```

CHAPTER 2

Usage

Alvis can be run from the command line using the .jar file. In the terminal, navigate to the dist directory in the Alvis project, and type e.g:

```
Java -jar Alvis.jar -type alignment -inputfmt paf -outputfmt tex \
        -in /path/to/alignments/alignments.paf -outdir /path/to/out/ -out prefix
```

This will create the file prefix_alignment.tex in the directory /path/to/out. The file prefix_alignment.tex can then be compiled by your favourite tex compiler and viewed as a pdf. The options used in this command form the minimum set required for Alvis to work, and are described in the following table.

Option <argument>	Description
`-type <diagram>`	Type of diagram to be produced.
`-inputfmt <format>`	Format of the alignment file to use.
`-outputfmt <format>`	Format of the output diagram (default: tex).
`-in <path>`	Full file path to alignment file.
`-outdir <dir>`	Directory to write output to. Alvis will create this directory if it does not exist (default: ./alvis_output).
`-out <prefix>`	Prefix to use for output file names (default: diagram).

The following are optional.

Option <argument>	Description
`-filter`	Filter small alignments to remove noise.
`-minAlignmentPC <int>`	Minimum size of alignments (as % of the query length) to keep when filtering.
`-chimeras`	Only display alignments from chimeras.
`-printChimeras`	Output a text file containing alignments belonging to chimeras.
`-chimeraPositions`	Include alignment start and end positions in the file created by `-printChimeras`.
`-minChimeraCoveragePC <int>`	Minimum coverage of query (as % of the query length) for identifying chimeras.
`-minChimeraAlignmentPC <int>`	Minimum size of alignments (as % of the query length) for identifying chimeras.
`-alignmentQueryName <query sequence id>`	ID of query sequence for exapnded contig alignment diagram.
`-alignmentTargetName <target sequence id>`	ID of target sequence for expanded contig alignment diagram.
`-blastfmt <format string>`	Format string as given to blast. Must be used if `-inputfmt` is `blast`.
`-tsizes <targets string>`	List of target names and sizes, separated by whitespace. Required if `inputfmt` is `sam`.
`-binsize <int>`	Size (in bp) of bins for genome coverage diagrams.
`-coverageType <type>`	The type of coverage heatmap to produce. Must be either “square” or “long”.
`-randomcolours <seed>`	Randomly generate reference sequence colours for contig alignment diagram, with seed.
`-colourfile <filename>`	Name of file specifying hex colour for each reference sequence. Each line must be in the format <ref sequence> <hex colour>

2.1 Diagrams

The type of diagram is specified with the `-type <diagram>` option etc. The currently available diagrams are:

- `alignment`
- `contigAlignment`
- `coveragemap`
- `genomecoverage`

Detailed information for each diagram can be found in the *Diagrams* section.

2.2 Input Formats

Alvis accepts a variety of alignment formats. These are specified by the option `-inputfmt <format>`, where `<format>` is one of the following: = - `paf` - `psl` - `blast` - `coords` - `tiling` - `sam`

2.2.1 Coords

A coords file can be created from a mummer .delta file using the `show-coords` command in the MUMer package. For this file to work with alvis, the `-B` option must be specified (see here for more details).

2.2.2 Tiling

Similarly, a tiling file can be created from a mummer .delta file using the `show-tiling` command. In this case, the `-a` option must be specified (see here for more details).

2.2.3 BLAST

If BLAST is given as the format, the input file must have been created by BLAST using the tabular option, and the following fields must be present in some order:

- `qseqid`
- `sseqid`
- `qstart`
- `qend`
- `qlen`
- `sstart`
- `send`
- `slen`

The parameter passed to blast after the `-outfmt` option must also be given to Alvis after the `-blastfmt` option. For example, the following command could be used:

```
Java -jar Alvis.jar -type alignment -inputfmt blast -outputfmt tex \
        -in /path/to/alignments/alignments.blast -outdir /path/to/out/ -out prefix \
        -blastfmt '6 qseqid sseqid qstart qend qlen sstart ssend slen'
```

if the file alignment.blast was created with:

```
blastn -db nt -query query.fa -out alignments.blast -outfmt '6 qseqid sseqid qstart␣
↪qend qlen sstart ssend slen'
```

2.2.4 SAM

When using a SAM file, Alvis will attempt to find the target contig sizes from the header section. If this unavailable, the user can supply these values through the `-tsizes` option, by typing a space-separaed list of target names and their sizes. E.g. `-tsizes 'Chr1 34964571 Chr2 22037565 Chr3 25499034 Chr4 20862711 Chr5 31270811'`.

2.3 Output Formats

Alvis can currently output most diagrams in two formats: SVG and laTeX. These are specified by the `-outputfmt <format>` option, where `<format>` is one of `tex` and `svg`. Note that if `tex` is specified, the user must compile the .tex file that is created to obtain a PDF. Currently the diagrams are drawn with the tikz library, so the user must have this installed.

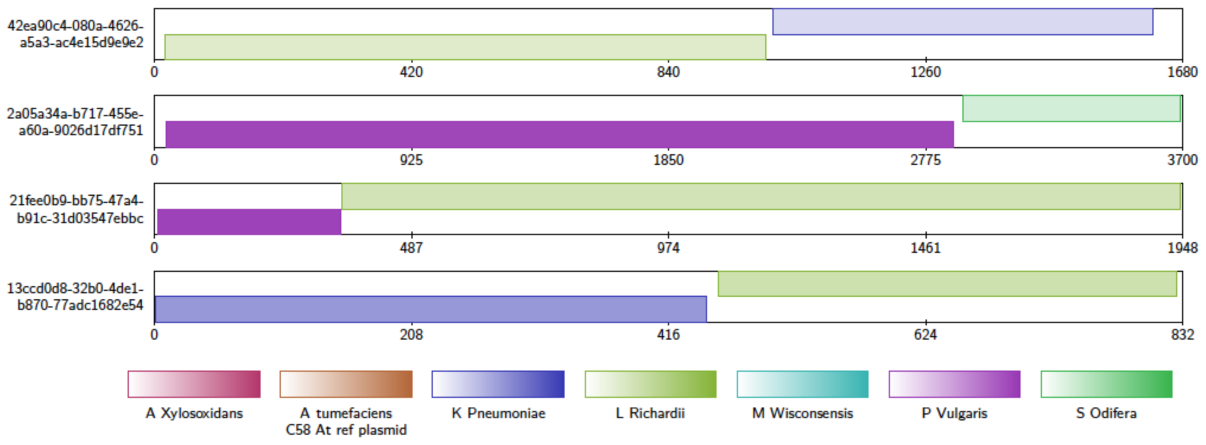
The following table shows the accepted input and output formats for each diagram.

	Input Formats						Output Formats	
	blast	coords	tiling	paf	psl	sam	svg	tex
Alignment Diagram	✓	✓	✓	✓	✓		✓	✓
Contig Alignment Diagram	✓	✓	✓	✓	✓		✓	✓
Coverage Map Diagram	✓	✓	✓	✓	✓	✓	✓	✓
Genome Coverage Diagram	✓	✓	✓	✓	✓	✓	✓	

2.4 Filtering

The user can filter alignments using the `-filter` option. This will cause alvis to ignore all alignments with length less than `-minAlignmentPC` % of the reference contig size (set to 0.5% by default). Note that this option is currently only used by the alignment diagram and the contig alignment diagram.

When using the `-chimera` option in conjunction with the contig alignment diagram, alvis will display only those alignments that it thinks could be a chimera. These are chosen when a query sequence is at least 90% covered by exactly two non-overlapping alignments, either from different reference sequences, or different loci of the same reference sequence. Each of these alignments must have a length of at least 10% of the query sequence. These values may be adjusted by the user with the `-minChimeraCoveragePC` and `-minChimeraAlignmentPC` options. The user should be aware that sequences are assumed to be non-circular; chimeras may be found when a read covers the join of a circular sequence.



Additionally, when the `-printChimeras` option is specified, a text file named `chimeras.txt` is written to the output directory. This is a tab-seperated values file, where each line describes a potential chimera. Each line has the following fields.

Column	Type	Description
1	String	Query sequence name.
2	int	Approximate position of chimera join on query sequence.
3	String	Target sequence name for first alignment.
4	String	Target sequence name for second alignment.

If the option `-chimeraPositions` is also specified, then for each chimera, the start and end positions of the two alignments forming the chimera are also written to the file `chimeras.txt`, relative to both the query sequence (i.e. the read) and the target sequence (i.e. the reference). In this case, each line has the following fields.

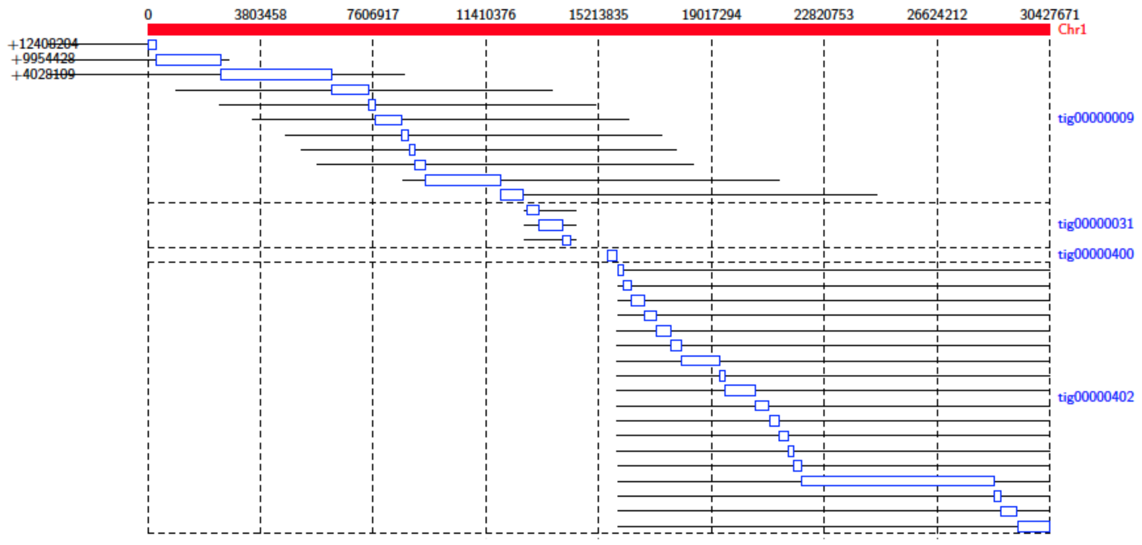
Column	Type	Description
1	String	Query sequence name.
2	int	Approximate position of chimera join on query sequence.
3	String	Target sequence name for first alignment.
4	String	Target sequence name for second alignment.
5	int	Start position on query of the first alignment.
6	int	End position on query of the first alignment.
7	int	Start position on query of the second alignment.
8	int	End position on query of the second alignment.
9	int	Start position on target of the first alignment.
10	int	End position on target of the first alignment.
11	int	Start position on target of the second alignment.
12	int	End position on target of the second alignment.

CHAPTER 3

Diagrams

3.1 Alignment Diagram

The Alignment Diagram draws alignments between a target (or reference) sequence (drawn in red), and query sequences. Each query sequence is represented by a line underneath the target sequence, with the alignment drawn as a box aligned to the target sequence.



This diagram type accepts the following input formats

- paf
- psl
- coords

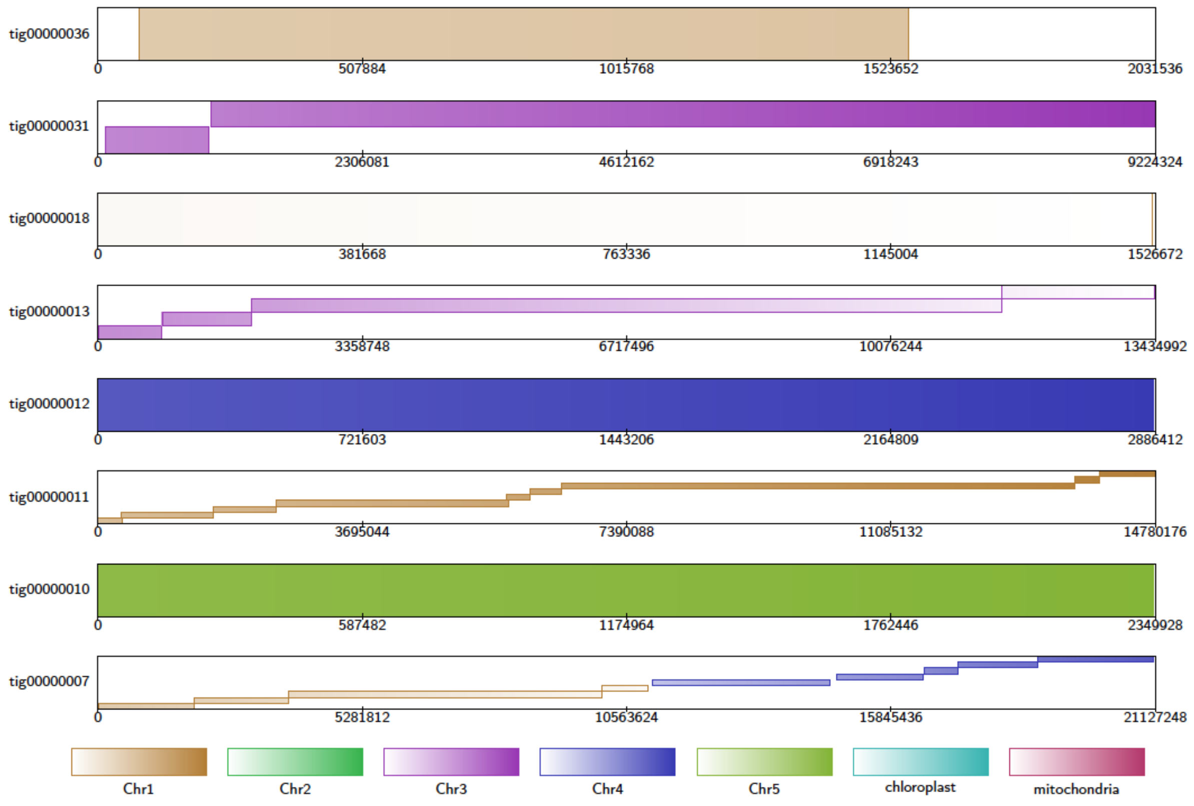
- tiling
- blast

and outputs the following formats

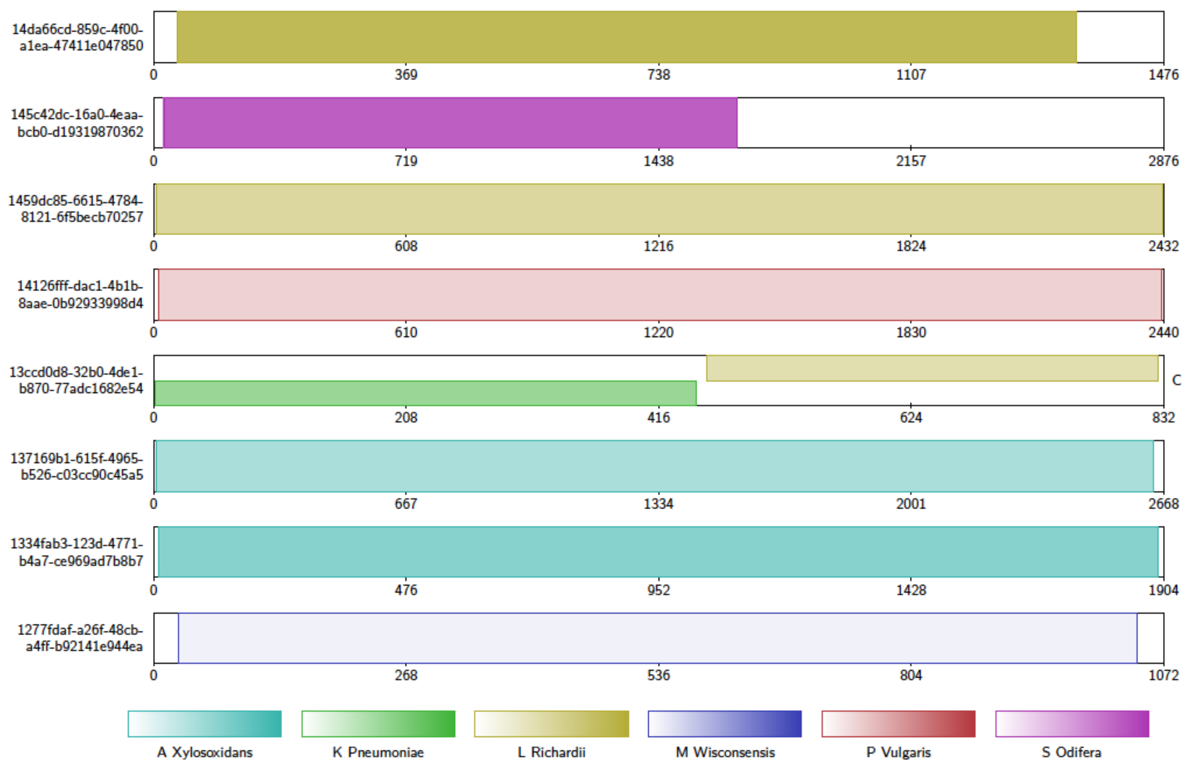
- tex
- svg.

3.2 Contig Alignment Diagram

For each query contig, the Contig Alignment Diagram draws a rectangle representing the query, containing the most prominent alignments to the reference contigs. These alignments are colour coded by target contig, and shaded to give an indication of the position in the target, and the orientation of the alignment. If the `-filter` option is not used, only the longest 20 alignments are drawn for each query contig.



If the alignment file contains alignments of reads to references, Alvis will highlight reads that it thinks are chimeric with a "C". In the diagram below, the fifth read appears to be a chimera of K. pneumoniae and L. richardii.



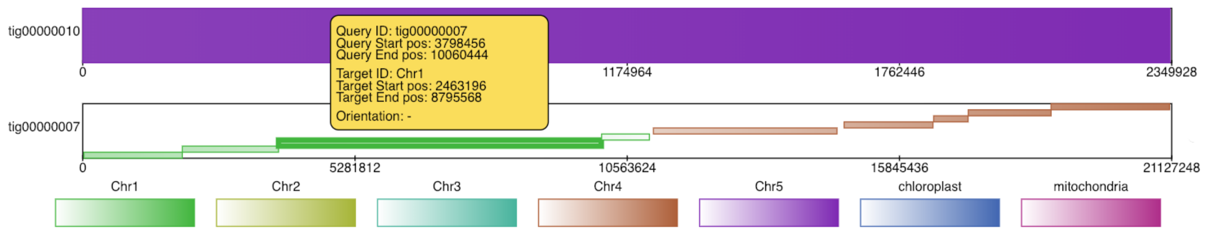
This diagram type accepts the following input formats

- paf
- psl
- coords
- tiling
- blast

and outputs the following formats

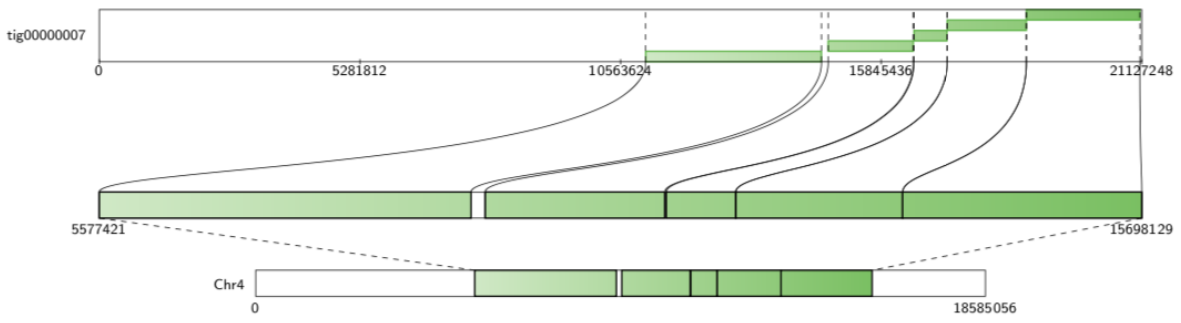
- tex
- svg.

If the user chooses svg as the output format, the diagram produced contains embedded javascript to make it interactive. In an internet browser, the user may click alignments to highlight them and see further details about the alignment.



Query ID: tig00000007
Query Start pos: 3798456
Query End pos: 10060444
Target ID: Chr1
Target Start pos: 2463196
Target End pos: 8795568
Orientation: -

Finally, if the user specifies a query contig and a reference contig by using the `-alignmentQueryName` and `-alignmentTargetName` options, a detailed diagram containing only these alignments is produced.



3.2.1 Colours

The default colour scheme for contig alignment diagrams is a colour-blind friendly palette of 7 colours. The user can instead choose to have colours randomly generated (this is unlikely to be colour-blind friendly) using the option `-randomcolours <seed>`, specifying a seed for the RNG for reproducability. The user can also specify the colour of each reference sequence using the option `-colourfile <filename>`, where `filename` is the path to a plain text file in which each line specifies a reference sequence and a colour in hex format. All reference sequences appearing in the alignment must be included in this file.

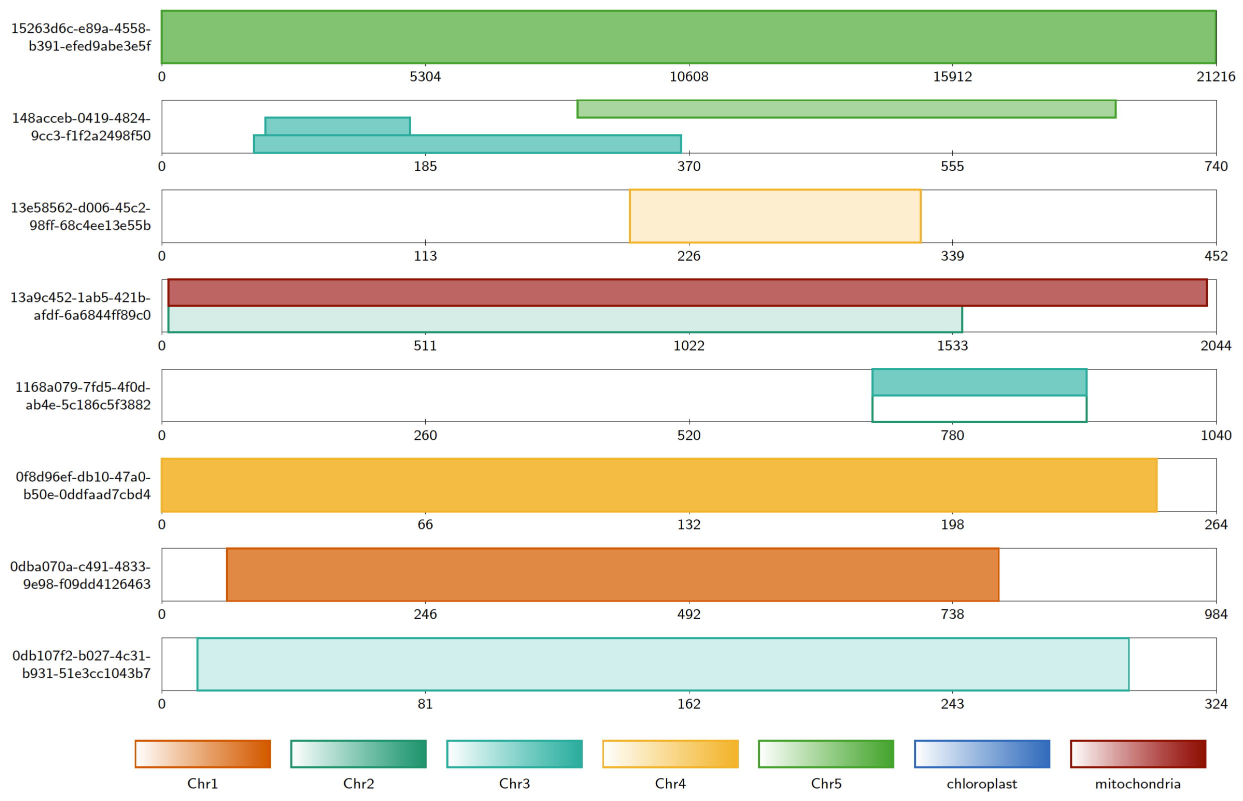
For example, if `colourfile.txt` is the file

```
Chr1     #d95f02
Chr2     #1b9e77
Chr3     #29b6a6
Chr4     #f5b935
Chr5     #4bac35
chloroplast      #3778c2
mitochondria     #990000
```

then the command:

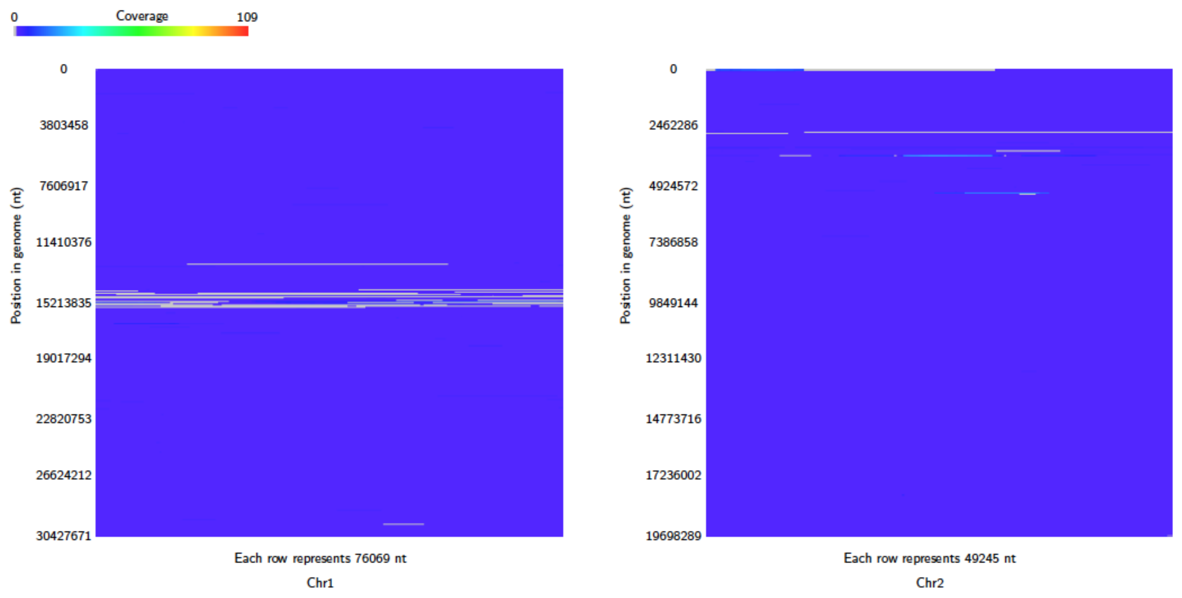
```
Java -jar Alvis.jar -inputfmt paf -outputfmt tex -type contigalignment -in mapping.
↪paf  -out example -colourfile colourfile.txt
```

will produce something like

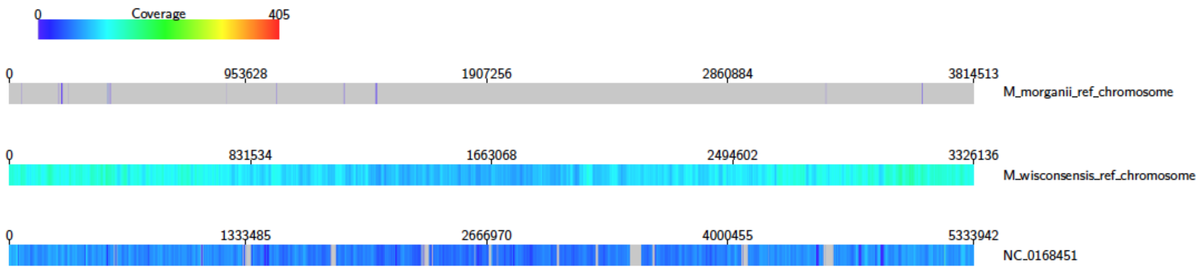


3.3 Coverage Map Diagram

Coverage of each target contig is counted by alignment for each query contig. To avoid counting the same query region multiple times, alignments with overlapping query coordinates are filtered by choosing the longest alignment. For each target contig a heatmap image is produced in which each pixel represents the coverage of a single position in the target contig. These are arranged in a tex or svg file. Note that each heatmap image is a fixed size, so the pixel scale is adjusted to fit. By default, square images are produced where the rows are read top to bottom, from left to right, as below.



If the `coverageType` option is set to `long`, then the heatmap consists of one row for each target, read left to right.



This diagram type accepts the following input formats

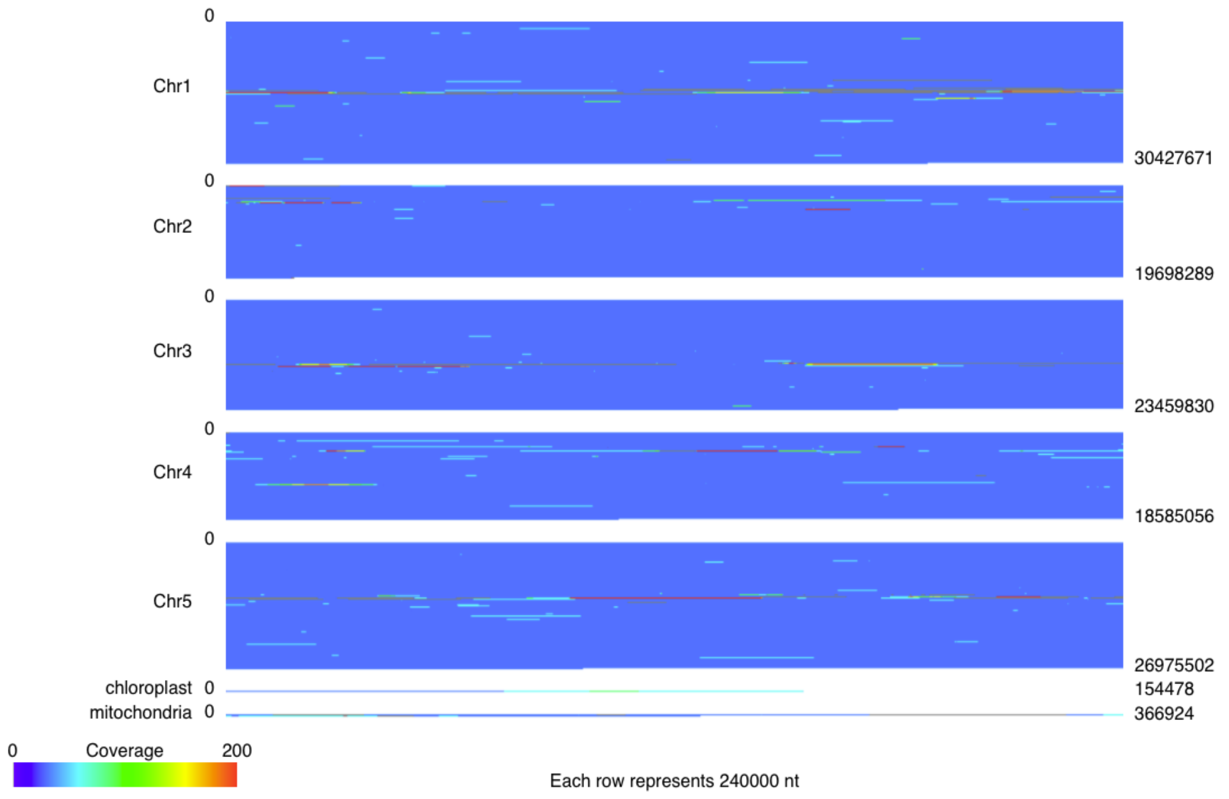
- paf
- psl
- coords
- tiling
- blast
- sam

and outputs the following formats

- tex
- svg.

3.4 Genome Coverage Diagram

Alignments are binned based on their position in the target contigs, and counted to calculate the coverage of each bin. By default, the bin size is 30 bp, but this can be set using the `-binsize` option. As in the Coverage Map diagram, alignments that overlap in the query contig are filtered. One heatmap is produced showing the coverage over all the target contigs. Unlike the Coverage Map Diagram, the scale for the heatmaps remains the same across all the target contigs.



This diagram type accepts the following input formats

- paf
- psl
- coords
- tiling
- blast
- sam

and outputs the following formats

- svg.

CHAPTER 4

Examples

4.1 Nanopore Example

This section takes the user through the basic functionality using example data that can be found in the github repository. First, download the repository and extract the files from `.../Alvis/tutorial_data/tutorial_data.tar` as follows:

```
mkdir /Users/.../Alvis_example
tar -C  /Users/.../Alvis_example -xvf /path/to/Alvis/tutorial_data/tutorial_data.tar
```

The Alvis_example directory now contains a directory containing three text files in the PAF format produced with minimap2:

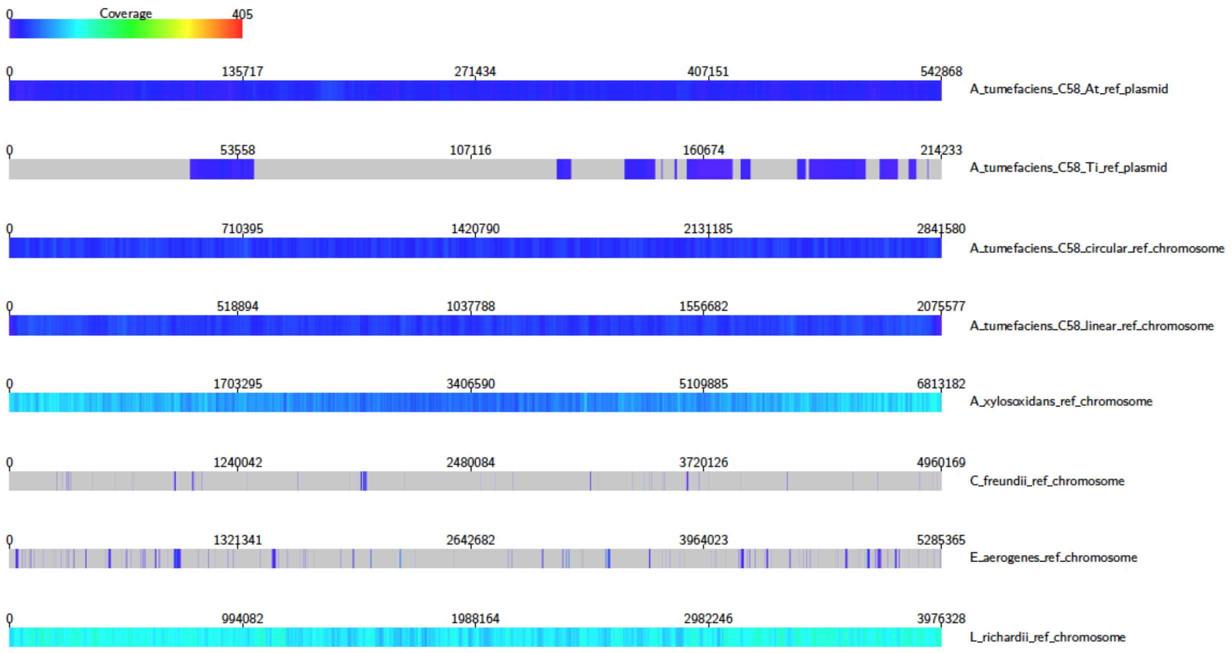
- alignments.paf: a file of alignments between the reads of a nanopore sequencing run and a set of bacterial reference genomes.
- alignments_sample.paf: a small sample of alignments from alignments.paf.
- assembly_mapping.paf: contains alignments between an assembly of Klebsiella Pneumoniae and a reference genome.

First, we will create coverage maps to show which species were present in the sample, and their relative abundances. From the terminal, navigate to the `Alvis_example/tutorial_data` directory created in the previous step, and type:

```
mkdir output
Java -jar /path/to/Alvis/dist/Alvis.jar -inputfmt paf -outputfmt tex -type␣
↪coverageMap -coverageType long -in alignments.paf -outdir output/ -out example
```

Once this has executed, the output directory, `/Users/.../Alvis_example/output/` will contain a file called `example_coverageMap.tex`. This may be compiled with your favourite tex compiler to create a PDF of the coverage map diagrams, e.g.:

```
cd output/
pdflatex example_coverageMap.tex
cd ..
```



We can see clearly that e.g. A. Xylosoxidans and L. Richardii are relatively abundant in the sample, whereas C. Freundii appears to not be present at all.

To create an SVG file of this diagram instead, type:

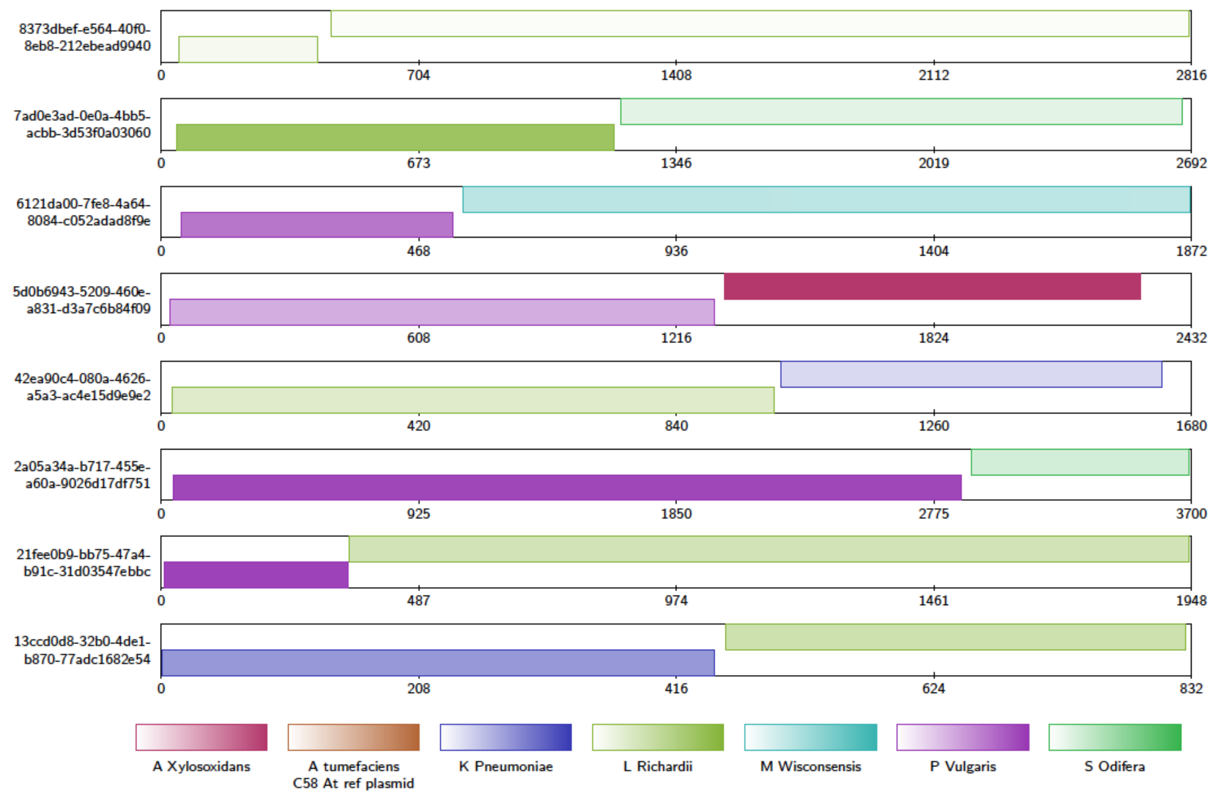
```
Java -jar /path/to/Alvis/dist/Alvis.jar -inputfmt paf -outputfmt svg -type␣
↪coverageMap -coverageType long -in alignments.paf -outdir output/ -out example
```

The SVG files created in the `output` directory can be viewed immediately with an internet browser.

Now we will look for chimeric reads from a subsample of the alignments, using Alvis' chimera filtering option. In the terminal, type:

```
Java -jar /path/to/Alvis/dist/Alvis.jar -inputfmt paf -outputfmt tex -type␣
↪contigAlignment -chimeras -in alignments_sample.paf  -outdir output/ -out example
```

A file called "example_contigAlignment.tex" is created in the output directory. This file contains a contig alignment diagram for each query that Alvis thinks could be a chimera. After compiling the TEX file as before, the user may inspect the alignments for each of these reads. Note that these alignments could also have been caused by similarity in the reference sequences.



You may also look for chimeric reads from the whole read set by using the above command and replacing the `-in` file with `alignments.paf`. This will produce a large tex file, which when compiled contains a contig alignment diagram for every chimeric read, and is 4,381 pages long. This is far too large to be inspected manually, so instead Alvis can write a plain text file for computer parsing, describing each chimera, using the following command:

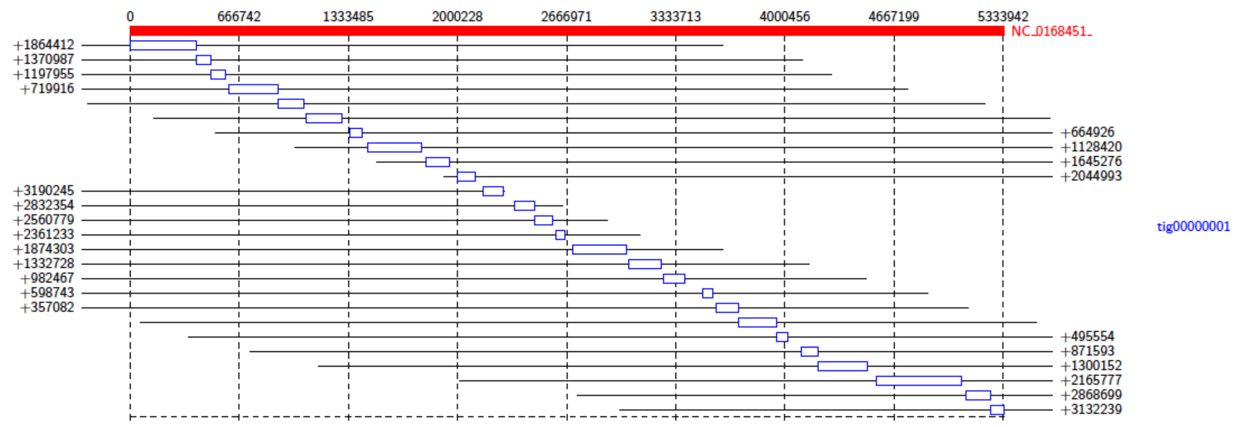
```
Java -jar /path/to/Alvis/dist/Alvis.jar -inputfmt paf -outputfmt tex -type␣
↪contigAlignment -chimeras -printChimeras -in alignments.paf  -outdir output/ -out␣
↪example
```

The file `chimeras.txt` (see *Filtering*) can then be found in the output directory.

The data package also contains a PAF file of alignments between an assembly of all the reads that mapped to K. Pneumoniae, and a reference genome. We will investigate the makeup of the assembly contigs. In the terminal, type:

```
Java -jar /path/to/Alvis/dist/Alvis.jar -inputfmt paf -outputfmt tex -type alignment -
↪filter -in assembly_mapping.paf  -outdir output/ -out example
```

As before, compile the TEX file produced to obtain a pdf.



This diagram shows us that almost all of the reference genome is covered by tig00000001. However, most of these alignments are transpositions.

4.2 Short Reads Example

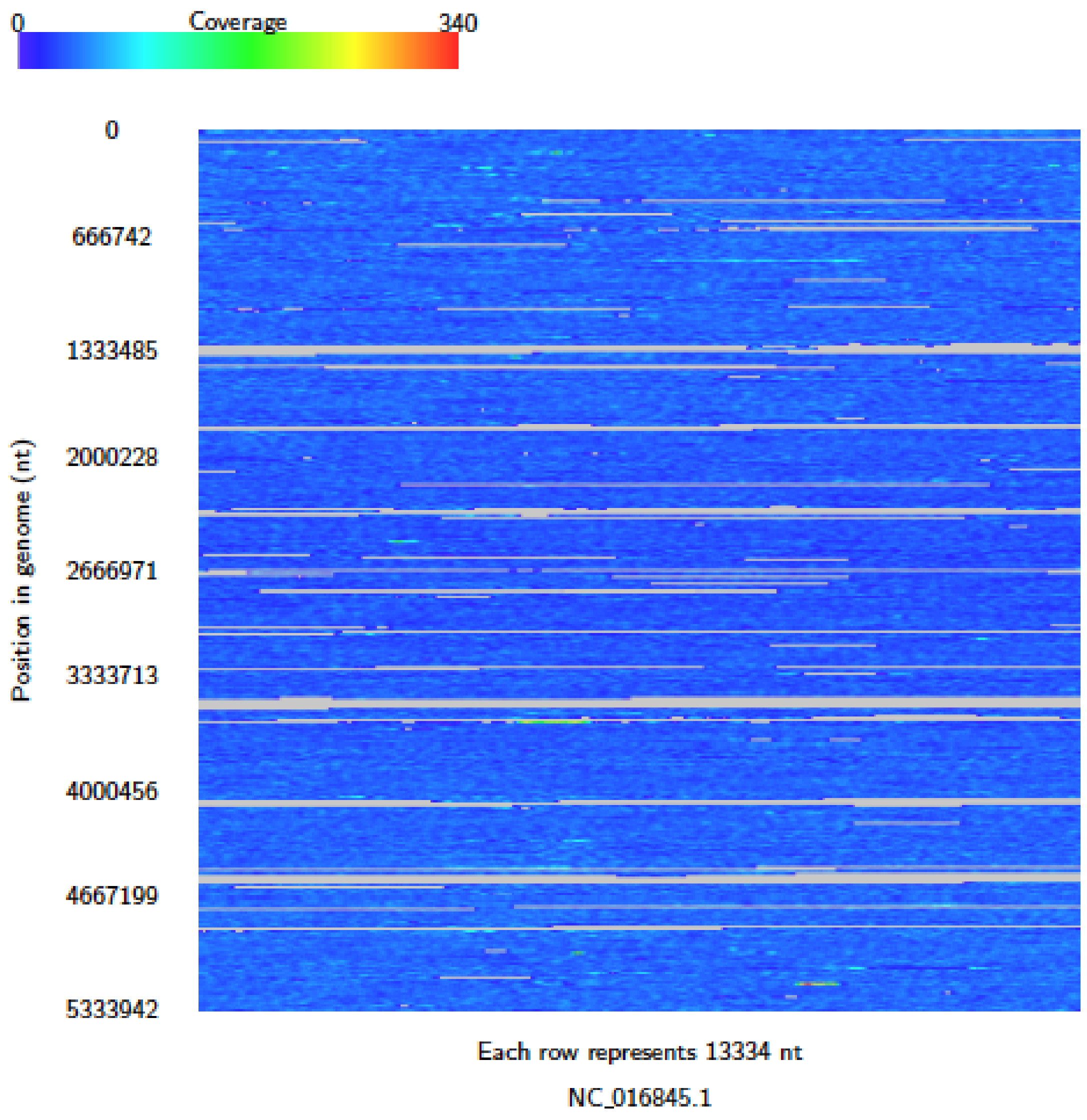
Alvis can also be used with short reads! In this section we give a tutorial on using Alvis to help analyse short read data. First, download the Klebsiella pneumoniae reads from run ERR2099161 found here, and the strain HS11286 reference fasta (NCBI Reference Sequence: NC_016845.1) found here to your working directory. We would like to know how well these reads cover the reference genome. Use your favourite read aligner to align the reads to the reference in SAM format e.g.

```
bowtie2-build klebsiella_pneumoniae.fasta klebsiella_pneumoniae
    bowtie2 -x klebsiella_pneumoniae -1 ERR2099161_1.fastq -2 ERR2099161_2.fastq -S␣
↪read_alignments.sam
```

Next use Alvis to build a coverage map diagram

```
Java -jar /path/to/Alvis/dist/Alvis.jar -inputfmt sam -outputfmt tex -type␣
↪coveragemap -in read_alignments.sam -outdir alvis_coveragemap/ -out kleb_reads
```

The coverage map shows that read coverage is quite low, with some large sections of the genome not covered at all.



Next, perform an assembly of these reads using your favourite short read assembler, e.g.

```
spades.py -1 ERR2099161_1.fastq -2 ERR2099161_2.fastq -o assembly
```

From the coverage of the reads, we don't expect this assembly to be that great. Use the tool dnadiff (from MUMmer) to obtain a report and alignments on the assembly against the reference.

```
dnadiff klebsiella_pneumoniae.fasta assembly/contigs.fasta
```

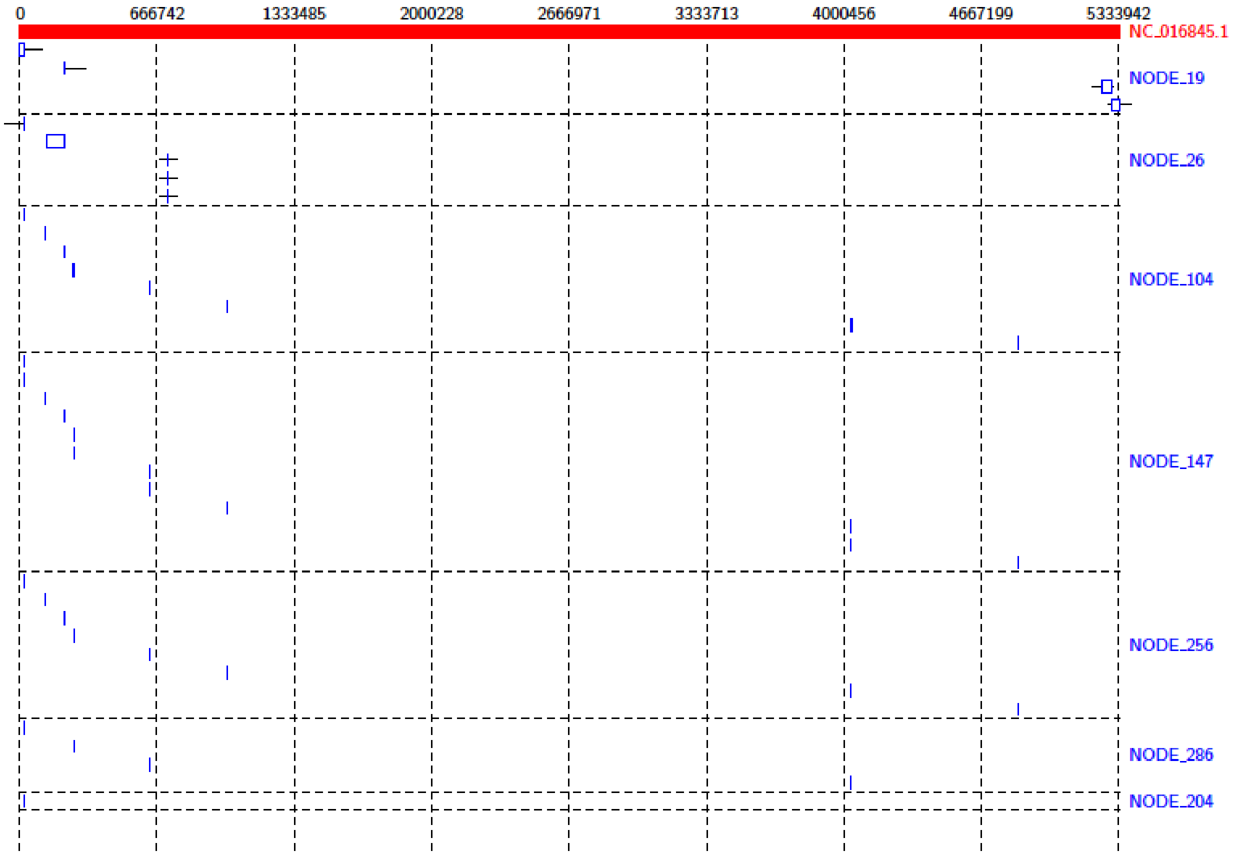
This produces a delta file (*out.delta*) from which Alvis can create a diagram. First we need to get it into a format that Alvis understands.

```
show-coords -B out.delta > out.coords
```

Then we can create an alignment diagram.

```
Java -jar /path/to/Alvis/dist/Alvis.jar -inputfmt coords -outputfmt tex -type␣
↪alignment -in out.coords -outdir alvis_assembly/ -out kleb_assembly
```

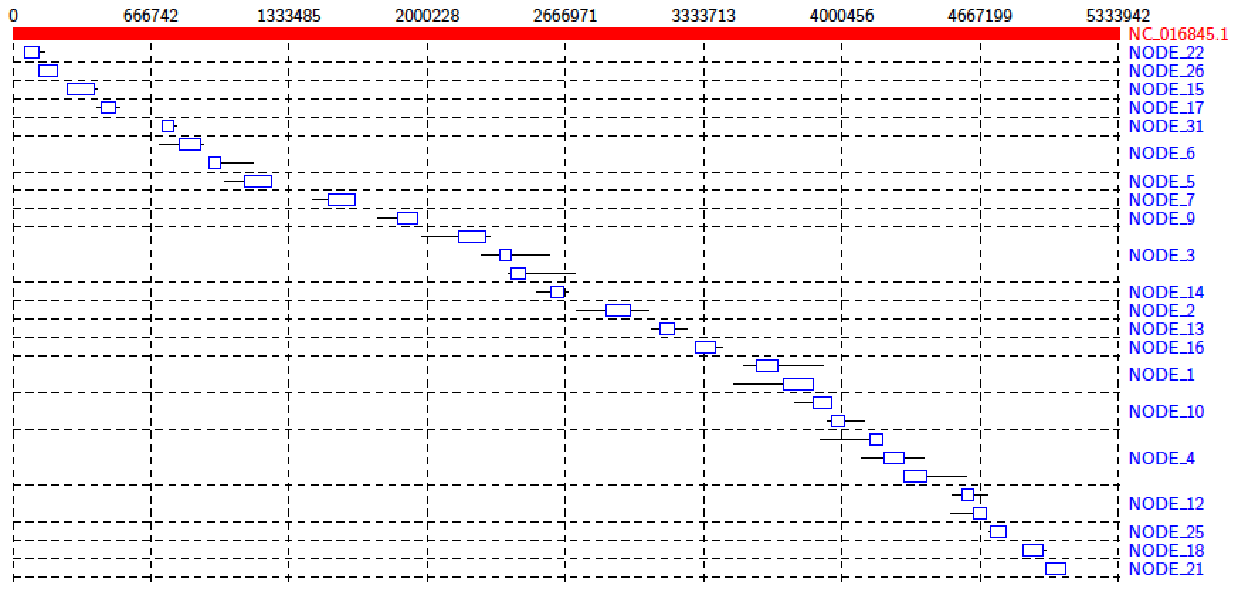
From the diagram we can see that the reference genome is covered sporadically across many contigs, most of which have very small alignments to different sections of the genome.



We can get a better idea of the contigs that are making a significant contribution by using the -filter option. Run the above command again, this time with the -filter option. (Note: this will overwrite the previous diagram.)

```
Java -jar /path/to/Alvis/dist/Alvis.jar -inputfmt coords -outputfmt tex -type␣
↪alignment -in out.coords -outdir alvis_assembly/ -out kleb_assembly -filter
```

This gives us a better idea of how the assembly covers the reference.


0
666742
1333485
2000228
2666971
3333713
4000456
4667199
5333942
NC_016845.1
NODE_22
NODE_26
NODE_15
NODE_17
NODE_31
NODE_6
NODE_5
NODE_7
NODE_9
NODE_3
NODE_14
NODE_2
NODE_13
NODE_16
NODE_1
NODE_10
NODE_4
NODE_12
NODE_25
NODE_18
NODE_21