Contents

1 Dependencies 3

2 Installation 5
   2.1 Install from the pre-compiled package ............................ 5
   2.2 Obtain source code and compile .................................. 6

3 Usage convention 7
   3.1 Naming ................................................................. 7
   3.2 General usage ........................................................ 7

4 List of tools 9
   4.1 jsa.seq.stats: Show statistics of sequences ..................... 9
   4.2 jsa.seq.sort: Sort the sequences in a file ...................... 10
   4.3 jsa.seq.extract: Extract subsequences from a genome .......... 10
   4.4 jsa.seq.split: Split multiple sequence file .................... 11
   4.5 jsa.seq.join: Join multiple sequences into one file .......... 11
   4.6 jsa.seq.annovcf: Annotate a vcf file ............................ 12
   4.7 jsa.seq.gff2fasta: Extract gene sequences ...................... 12
   4.8 jsa.seq.emalign: Align two sequences using EM .................. 13
   4.9 jsa.hts.countReads: Count reads from bam files ............... 13
   4.10 jsa.hts.errorAnalysis: Error analysis of sequencing data .... 14
   4.11 jsa.hts.n50: Compute N50 of an assembly ....................... 15
   4.12 npReader: real-time conversion and analysis of Nanopore sequencing data 15
   4.13 jsa.np.filter: Filter sequencing data .......................... 18
   4.14 jsa.np.rtSpeciesTyping: Bacterial species typing with Oxford Nanopore sequencing 19
   4.15 jsa.np.rtMLST: Multi-locus Sequencing Typing in real-time with Nanopore sequencing 20
   4.16 jsa.np.rtStrainTyping: Bacterial strain typing with Oxford Nanopore sequencing 21
   4.17 jsa.np.rtResistGenes: Antibiotic resistance gene identification in real-time with Nanopore sequencing 22
   4.18 npScarf: real-time scaffolder using SPAdes contigs and Nanopore sequencing reads 23
   4.19 barcode: real-time de-multiplexing Nanopore reads from barcode sequencing 26
   4.20 jsa.util.streamServer: Receiving streaming data over a network 28
   4.21 jsa.util.streamClient: Streams data over a network .......... 29
   4.22 XMas: Robust estimation of genetic distances with information theory 29
   4.23 jsa.phylo.normalise: Normalise branch length of a phylogeny 30
   4.24 capsim: Simulating the Dynamics of Targeted Capture Sequencing with CapSim 31
   4.25 Expert Model: tool for compression of genomic sequences ...... 32
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Credits</td>
<td>35</td>
</tr>
<tr>
<td>License</td>
<td>37</td>
</tr>
<tr>
<td>Indices and tables</td>
<td>39</td>
</tr>
</tbody>
</table>
Japsa is a free, open source *Java Package for Sequence Analysis*. It contains a range of analysis tools that biologists and bioinformaticians would routinely use but may not be available elsewhere. It also provides a Java library to be incorporated in other Java projects.

The package aims to be lightweight (fast and memory efficient) and to use the least possible dependencies. Its tools have a consistent command line interface and support reading from/writing to streams whenever possible.

Japsa and its source code is licensed under the BSD license and is available on GitHub.

Contents:
CHAPTER 1

Dependencies

Japsa has the following dependencies, which are included in the package.

- JFreeChart >= 1.0.19 (http://www.jfree.org/jfreechart/) and JCommon (http://www.jfree.org/jcommon/)
- colloquial.jar, now java-arithcode (https://github.com/bob-carpenter/java-arithcode)
- common-math.jar >=3.3.0 (http://commons.apache.org/proper/commons-math/)
- htsjdk >=1.126
- guava >=16.0
- JHDF5 >= 18

Naturally, it also requires a Java Runtime Environment >=1.8 (java) installed to compile and run the package.

Some tools in the package have additional dependencies (not included in the package), as follows:


Japsa is provided with a ready to run package at every stable release. These pre-built releases are compiled with javac 1.8 to ensure compatibility. If you wish to use the latest version of Japsa, or use japsa compiled with your version of Java runtime, you will need to build from source, which requires:

- Java Development Kit (javac) >= 1.8
- make
- git/wget (optional)
- mvn (optional if you want to build with maven)

Note that a specific tool may require extra dependencies (such as bwa etc). Check the documentation for individual tools for more detailed information.
CHAPTER 2

Installation

There are two methods to install Japsa in your computer. The first method (using pre-compiled package in JDK 1.8) is straight-forward and can be used for any operating systems, including Windows. The second method (compile from source code) requires some extra tools (make and JDK) but may yield better runtime performance as the package will be compiled with the same version of the Java Virtual Machine used to run.

2.1 Install from the pre-compiled package

Pre-compiled package of Japsa is made available under each release. Installation from this will not require extra build tools such as javac, git, and make.

Just download a JapsaRelease (e.g., from https://github.com/mdcao/japsa/releases), unpack the tarball and run the install.sh script (install.bat for Windows) in the release directory:

```bash
tar zxvf JapsaRelease.tar.gz
cd JapsaRelease
./install.sh
```

The installation will ask for specific details to install the package. If you agree with its suggestion, just type Enter. The questions are:

- **Directory to install japsa**: Enter a directory to install japsa
- **Default memory allocated to jvm**: Enter a default amount of memory allocated to the Java Virtual Machine. This value should be smaller than the size of your computer. This value, however, can be changed for each specific invocation of a program.
- **Enforce your jvm to run on server mode**: Type y if your java support running in server mode.
- **Path to HDF library**: Enter path to HDF library. Generally, you need to have HDFViewer (https://www.hdfgroup.org/products/java/release/download.html) installed, and enter the path to file libhdf5.so (on Linux/Unix/Mac) or to jhdf5.dll (Windows). This is only required if you intend to use npReader(jsa.np.f5reader). Note that we tested with hdfj version 2.10.1, which you can download from https://support.hdfgroup.org/ftp/HDF5/releases/HDF-JAVA/hdf-java-2.10.1/bin/
2.2 Obtain source code and compile

This installation method is recommended as japsa will be compiled with the same Java version used to run it. This method however requires Java Development Kit and Make to be installed. This method has not been tested with Windows.

First, download the latest source code:

```bash
git clone https://github.com/mdcao/japsa
cd japsa
```
or download from a release (Check out https://github.com/mdcao/japsa/releases for the latest releases):

```bash
wget https://github.com/mdcao/japsa/releases/download/v1.7-08a/JapsaRelease.tar.gz
tar zxvf JapsaRelease.tar.gz
cd JapsaRelease
```

and run ‘make’ to compile and install japsa:

```bash
make install
  [INSTALL_DIR=~/.usr/local \]
  [MXMEM=7000m \]
  [SERVER=true \]
  [JLP=/usr/lib/jni]
```

This will install japsa according the directives:

- **INSTALL_DIR**: specifies the directory to install japsa
- **MXMEM**: specifies the default memory allocated to the java virtual machine
- **SERVER**: specifies whether to launch the java virtual machine in server mode
- **JLP**: specifies paths to `libjhdf5` (needed for npReader)

If any of the above directives are not specified, the installation will ask during the installation.

To uninstall Japsa, run the following in the japsa directory:

```bash
make uninstall INSTALL_DIR=~/.usr/local
```

where INSTALL_DIR points the directory Japsa was installed.
3.1 Naming

For a list of tools, type:

```
jsa
```

This will output the version of Japsa installed, the java version was used to compile the package, together with a list of tools and a brief description for each tool.

As you probably noticed from looking at the list of tools, every tool’s name starts with `jsa`, followed by group (e.g., hts) and the specific tool function. In a shell that allows auto-completion, one can hit the tab key to see the list of tools.

3.2 General usage

For the usage of a tool, type the tool name followed by --help. E.g.,

```
jsa.seq.sort --help
```

which will print out:

```
Sort sequences based on their lengths

Usage: jsa.seq.sort [options]
Options:
  --input=s  Name of the input file, - for standard input (REQUIRED)
  --output=s Name of the output file, - for standard output (REQUIRED)
  --alphabet=s Alphabet of the input file. Options: DNA (DNA=DNA16), DNA4 (ACGT), DNA5 (ACGTN), DNA16 and Protein (default='DNA')
```

(continues on next page)
To specify an option, one can use a single dash (-) or a double dash (–). One can even shorten the option to a non-ambiguous prefix. For example:

```
jsa.seq.sort -i=input.fas -o output.fasta --a DNA --sortKey=length --reverse -n true
```

In the above example, the option *input* is shorten to -i, *output* to -a. One can have an equal side (-i=input.fas) or without (-o output.fas). For boolean option, by presence of the option specifies a true value (e.g., –reverse), or one can change it with true/false.

If there are more than one options sharing the same prefix, for example *input* and *infile*, one has to use longer prefixes to identify the two (–inp and –inf).
This chapter presents the list of tools provided by Japsa. We are in the process of documenting 40+ tools, so stay tuned.

4.1 *jsa.seq.stats*: Show statistics of sequences

*jsa.seq.stats* shows the compositional statistics of sequences in file or from standard input). It is included in the Japsa package. Please see check the installation page for instructions.

4.1.1 Synopsis

*jsa.seq.stats*: Show statistical composition of sequences

4.1.2 Usage

```
jsa.seq.stats [options]
```

4.1.3 Options

- **--input=s**: Name of the input file, - for standard input (REQUIRED)
- **--alphabet=s**: Alphabet of the input file. Options: DNA (DNA=DNA16), DNA4 (ACGT), DNA5(ACGTN), DNA16 and Protein (default='DNA')
- **--help**: Display this usage and exit (default='false')
4.2 *jsa.seq.sort*: Sort the sequences in a file

*jsa.seq.sort* sort the sequences from a file or from a standard input into some order. *jsa.seq.sort* is included in the Japsa package. Please see check the installation page for instructions.

4.2.1 Synopsis

*jsa.seq.sort*: Sort sequences based on their lengths

4.2.2 Usage

```
jsa.seq.sort [options]
```

4.2.3 Options

```
--input=s      Name of the input file, - for standard input (REQUIRED)
--output=s    Name of the output file, - for standard output (REQUIRED)
--alphabet=s  Alphabet of the input file. Options: DNA (DNA=DNA16), DNA4 (ACGT), DNA5(ACGTN), DNA16 and Protein (default='DNA')
--number      Add the order number to the beginning of contig name (default='false')
--pad         Pad – this only applied for number (default='false')
--reverse     Reverse sort order (default='false')
--sortKey=s   Sort key (default='length')
--help        Display this usage and exit (default='false')
```

4.3 *jsa.seq.extract*: Extract subsequences from a genome

*jsa.seq.extract* is included in the Japsa package. Please see check the installation page for instructions.

4.3.1 Synopsis

*jsa.seq.extract*: Extract subsequences

4.3.2 Usage

```
jsa.seq.extract [options] <chr:start-end> <chr:start-end> ...
```
4.3.3 Options

- **--input=s** Name of the input file, - for standard input (REQUIRED)
- **--output=s** Name of the output file, - for standard output (REQUIRED)
- **--alphabet=s** Alphabet of the input file. Options: DNA (DNA=DNA16), DNA4 (ACGT), DNA5(ACGTN), DNA16 and Protein (default='DNA')
- **--reverse** Reverse complement the subsequence (default='false')
- **--format=s** format of the output file (jsa and fasta) (default='fasta')
- **--help** Display this usage and exit (default='false')

4.4 *jsa.seq.split*: Split multiple sequence file

*jsa.seq.split* splits a file containing multiple sequences to each file containing a sequence. It is included in the Japsa package. Please see the installation page for instructions.

4.4.1 Synopsis

*jsa.seq.split*: Break a multiple sequence files to each sequence per file

4.4.2 Usage

```
jsa.seq.split [options]
```

4.4.3 Options

- **--input=s** Name of the input file, - for standard input (REQUIRED)
- **--alphabet=s** Alphabet of the input file. Options: DNA (DNA=DNA16), DNA4 (ACGT), DNA5(ACGTN), DNA16 and Protein (default='DNA')
- **--output=s** Prefix of the output files (default='out_')
- **--format=s** Format of output files. Options: japsa or fasta (default='fasta')
- **--help** Display this usage and exit (default='false')

4.5 *jsa.seq.join*: Join multiple sequences into one file

4.5.1 Synopsis

*jsa.seq.join*: Join multiple sequences into one
4.5.2 Usage

```bash
jsa.seq.join [options] file1 file2 ...
```
4.7.1 Synopsis

`jsa.seq.gff2fasta`: Extract sequences from a gff annotation

4.7.2 Usage

```
jsa.seq.gff2fasta [options]
```

4.7.3 Options

- `-sequence=s` The sequence (whole chromosome) (REQUIRED)
- `-gff=s` Annotation file in gff format (REQUIRED)
- `-type=s` types of features to be extracted (all, gene, CDS etc) (default='gene')
- `-flank=i` Size of flanking regions (default='0')
- `-output=s` Name of the output file, - for standard output (REQUIRED)
- `--help` Display this usage and exit (default='false')

4.8 `jsa.seq.emalign` Align two sequences using EM

4.8.1 Synopsis

`jsa.seq.emalign`: Get the best alignment of 2 sequences using Expectation-Maximisation on Finite State Machine

4.8.2 Usage

```
jsa.seq.emalign [options] seq1 seq2
```

4.8.3 Options

- `--iteration=i` Number of iteration (default='5')
- `--help` Display this usage and exit (default='false')

4.9 `jsa.hts.countReads`: Count reads from bam files

4.9.1 Synopsis

`jsa.hts.countReads`: Count the number of reads in some regions from a sorted, indexed bam file
4.9.2 Usage

\texttt{jlsa.hls.countReads [options]}

4.9.3 Options

\begin{itemize}
  \item \texttt{--bamFile=s} Name of the bam file (REQUIRED)
  \item \texttt{--bedFile=s} Name of the regions file in bed format (REQUIRED)
  \item \texttt{--output=s} Name of output file, - for standard out. (default=-)
  \item \texttt{--flanking=i} Size of the flanking regions, effectively expand the region by flanking (default=0)
  \item \texttt{--qual=i} Minimum quality (default=0)
  \item \texttt{--filterBits=i} \textbf{Filter reads based on flag. Common values:} 0 no filter 256 exclude secondary alignment 1024 exclude PCR/optical duplicates 2048 exclude supplementary alignments (default=0)
  \item \texttt{--contained} Count reads contained in the region (default=false)
  \item \texttt{--overlap} Count number of read overlap with the region (default=false)
  \item \texttt{--span} Count reads span the region (default=false)
  \item \texttt{--help} Display this usage and exit (default=false)
\end{itemize}

4.10 \texttt{jlsa.hls.errorAnalysis}: Error analysis of sequencing data

\texttt{jlsa.hls.errorAnalysis} assesses the error profile of sequencing data by getting the numbers of errors (mismatches, indels etc) from a bam file. Obviously, it does not distinguish sequencing errors from mutations, and hence consider mutations as errors. It is best to use with the bam file from aligning sequencing reads to a reliable assembly of the sample.

4.10.1 Synopsis

\texttt{jlsa.hls.errorAnalysis}: Error analysis of sequencing data

4.10.2 Usage

\begin{itemize}
  \item \texttt{jlsa.hls.errorAnalysis [options]}
\end{itemize}

4.10.3 Options

\begin{itemize}
  \item \texttt{--bamFile=s} Name of bam file (REQUIRED)
  \item \texttt{--reference=s} Name of reference genome (REQUIRED)
  \item \texttt{--pattern=s} Pattern of read name, used for filtering (default=null)
  \item \texttt{--qual=i} Minimum quality required (default=0)
\end{itemize}
4.11  \textit{jsa.hts.n50}: Compute N50 of an assembly

4.11.1 Synopsis

\textit{jsa.hts.n50}: Compute N50 of an assembly

4.11.2 Usage

\begin{verbatim}
jsa.hts.n50 [options]
\end{verbatim}

4.11.3 Options

\begin{itemize}
\item \textbf{--input=s} Name of the file (REQUIRED)
\item \textbf{--help} Display this usage and exit (default=’false’)
\end{itemize}

4.12 \textit{npReader}: real-time conversion and analysis of Nanopore sequencing data

\textit{npReader} (jsa.np.npreader) is a program that extracts Oxford Nanopore sequencing data from FAST5 files, performs an initial analysis of the date and streams them to real-time analysis pipelines. These pipelines can run on the same computer or on computing clouds/high performance clusters.

\textit{npReader} is included in the Japsa package. It requires JAVA HDF5 INTERFACE (JHI5) library to be installed prior to setting up Japsa. Details of installation as follows:

\textbf{On Windows/Mac}

1. Download and install HDF-View from https://www.hdfgroup.org/products/java/release/download.html. Note the folder that the JHI library is installed, e.g., \texttt{C:\Program Files\HDF\Group\HDFView\2.11.0\bin}

2. Follow the instructions to install Japsa on http://japsa.readthedocs.org/en/latest/install.html. Upon prompting for “Path to HDF library”, enter the above path.

\textbf{On Linux}

You can either install the JHI5 library by downloading the software from https://www.hdfgroup.org/products/java/JNI/jhi5/index.html or from your Linux distribution software repository, such as:

\begin{verbatim}
sudo apt-get install libjhdf5-jni
\end{verbatim}

The library is typically installed to /usr/lib/jni. Enter this path when prompted for “Path to HDF library” during installation of Japsa.

4.12.1 Synopsis


4.12.2 Usage

```bash
jsa.np.npreader [options]
```

4.12.3 Options

- `--GUI` Run with a Graphical User Interface (default='false')
- `--realtime` Run the program in real-time mode, i.e., keep waiting for new data from Metrichor agent (default='false')
- `--folder=s` The folder containing base-called reads (default='null')
- `--fail` Get sequence reads from fail folder (default='false')
- `--output=s` Name of the output file, - for stdout (default='')
- `--streams=s` Stream output to some servers, format “IP:port,IP:port” (no spaces) (default='null')
- `--format=s` Format of sequence reads (fastq or fasta) (default='fastq')
- `--minLength=i` Minimum read length (default='1')
- `--number` Add a unique number to read name (default='false')
- `--stats` Generate a report of read statistics (default='false')
- `--time` Extract the sequencing time of each read – experimental (default='false')
- `--exhaustive` Whether to traverse the input directory exhaustively (albacore) or lazily (metrichor) (default='false')
- `--barcode=s` The file containing all barcode sequences for demultiplexing. (default='null')
- `--help` Display this usage and exit (default='false')

4.12.4 See also

jsa.np.filter, jsa.util.streamServer, jsa.util.streamClient, jsa.np.rtSpeciesTyping, jsa.np.rtStrainTyping, jsa.np.rtResistGenes

4.12.5 Usage examples

A summary of npReader usage can be obtained by invoking the –help option:

```bash
jsa.np.npreader --help
```

The simplest way to run npReader in GUI mode is by typing:

```bash
jsa.np.npreader
```
and specify various options in the GUI. All of these options can be specified from the command line:

```
jsa.np.npreader -GUI -realtime -folder c:\Downloads\ -fail -output myrun.fastq ----minLength 200 --stats
```

npReader can run natively on a Windows laptop that runs the Metrichor agent. It can stream sequence data to multiple analysis pipelines on the same computer and/or on high performance clusters and computing clouds.

Start several analysis pipelines on some remote machines. Such a pipeline can be to count how many reads aligned to chromosomes A and B:

```
jsa.util.streamServer --port 3456 | \n bwa mem -t 8 -k11 -W20 -r10 -A1 -B1 -O1 -E1 -L0 -Y -K 10000 index - | \n awk -F "\t" 'BEGIN{A=0;B=0;N++;} NF>4 \n {if ($3=="chrA") A++; if ($3=="chrB") B++; \n if (NR %100==0) \n {print "At " NR " reads, " A " aligned to chr A; " B " aligned to chr B"} \n }'
```

In this pipeline, the `jsa.util.streamServer` program receives stream data from `npReader` and forwards to `bwa`, which aligns the data to a reference and in turn streams the alignment in sam format to the awk program to perform a simple analysis of counting reads aligned to chrA and chrB.

The Japsa package contains several real-time analysis (`jsa.np.speciesTyping`, `jsa.np.geneStrainTyping`, `jsa.np.resistGenes`). They can be used to set up analysis pipelines, such as:

```
jsa.util.streamServer --port 3457 \n bwa mem -t 8 -k11 -W20 -r10 -A1 -B1 -O1 -E1 -L0 -Y -K 10000 index - | \n jsa.np.speciesTyping -bam - --index speciesIndex -output output.dat
```

Once these pipelines are ready, npReader can start streaming data off the MinION and the Metrichor agent to these pipelines:

```
jsa.np.npreader -realtime -folder c:\Downloads\ -fail -output myrun.fastq ----minLength 200 --streams server1IP:3456,server2IP:3457
```

One can run npReader on a computing cloud if the download folder (containing base-called data) can be mounted to the cloud. In such case, npReader can direct stream data to the pipelines without the need of `jsa.util.streamServer`:

```
jsa.np.npreader -realtime -folder c:\Downloads\ -fail -output - | \n bwa mem -t 8 -k11 -W20 -r10 -A1 -B1 -O1 -E1 -L0 -Y -K 10000 index - | \n jsa.np.speciesTyping -bam - --index speciesIndex -output output.dat
```

npReader now supports barcode sequencing demultiplex. For this analysis, it requires a FASTA file of barcode tag sequences and will classify output sequences based on alignment. User can specify the threshold for alignment confidence from the GUI. Demultiplexing results are illustrated as prefix Barcode:<sample>:<score>| added to each output sequence name.

```
jsa.np.npreader -GUI -barcode barcode.fasta
```

Japsa also provides `jsa.np.filter`, a tool to bin sequence data in groups of the user’s liking. Like any other streamline tools, `jsa.np.filter` can run behind `jsa.util.streamServer` on a remote machine, or can get data directly from npReader via pipe:

```
jsa.np.npreader -realtime -folder c:\Downloads\ -fail -output - | \n jsa.np.filter -input - -lenMin 2000 --qualMin 10 -output goodreads.fq
```
One can also use `tee` to group data into different bins *in real-time* with `jsa.np.filter`:

```
jsa.np.npreader -realtime -folder c:\Downloads\ -fail -output - | \
  tee >(jsa.np.filter -input - -lenMax 2000 -output 0k2k.fq) \ 
  >(jsa.np.filter -lenMin 2000 -lenMax 4000 -input - -output 2k4k.fq) \ 
  >(jsa.np.filter -lenMin 4000 -lenMax 6000 -input - -output 4k6k.fq) \ 
  >(jsa.np.filter -lenMin 6000 -input - -output 6k.fq) \
> all.fq
```

These bins can also be piped/streamed to different analysis pipelines as above.

### 4.13 `jsa.np.filter`: Filter sequencing data

`jsa.np.filter` filters sequencing data based on sequence read type, length and quality. Examples of its usage can be found on `jsa.np.npreader`.

#### 4.13.1 Synopsis

`jsa.np.filter`: Filter nanopore reads data from fastq file

#### 4.13.2 Usage

```
jsa.np.filter [options]
```

#### 4.13.3 Options

- **--input=s**  Name of the input file, - for standard input (REQUIRED)
- **--output=s**  Name of the output file, - for standard output (REQUIRED)
- **--lenMin=i**  Minimum sequence length (default='0')
- **--lenMax=i**  Minimum sequence length (default='2147483647')
- **--qualMin=d**  Minimum average quality (default='0.0')
- **--qualMax=d**  Maximum average quality (default='1000.0')
- **--group=s**  Group need to be extracted, leave blank for selecting all groups (default='')
- **--excl2D**  Exclude 2D reads (default='false')
- **--exclTemp**  Exclude template reads (default='false')
- **--exclComp**  Exclude complement reads (default='false')
- **--format=s**  Format of the output file (default='fastq')
- **--help**  Display this usage and exit (default='false')

#### 4.13.4 See also

`jsa.np.npreader`, `jsa.util.streamServer`, `jsa.util.streamClient`
4.14 *jsa.np.rtSpeciesTyping*: Bacterial species typing with Oxford Nanopore sequencing

*jsa.np.rtSpeciesTyping* identify proportions of species from a DNA sample using Oxford Nanopore sequencing in real-time. It reads data in SAM/BAM format of the alignments of sequence reads to a collection of species genomes.

We provide a genome collection of nearly 1500 bacterial species on http://data.genomicsresearch.org/Projects/npAnalysis/. Refer to the documentation at https://github.com/mdcao/npAnalysis/ for more details.

### 4.14.1 Synopsis

*jsa.np.rtSpeciesTyping*: Realtime species typing using Nanopore Sequencing data

### 4.14.2 Usage

```
jsa.np.rtSpeciesTyping [options]
```

### 4.14.3 Options

- `--output=s` Output file, - for standard output (default='output.dat’)
- `--bamFile=s` The bam file (REQUIRED)
- `--indexFile=s` indexFile (REQUIRED)
- `--qual=d` Minimum alignment quality (default='1.0’)
- `--twodonly` Use only two dimentional reads (default='false’)
- `--read=i` Minimum number of reads between analyses (default='50’)
- `--time=i` Minimum number of seconds between analyses (default='30’)
- `--web` Whether to use Web visualization. (default='false’)
- `--log` Whether to write mapping details to species2reads.map. (default='false’)
- `--help` Display this usage and exit (default='false’)

### 4.14.4 See also

`jsa.np.npreader, jsa.np.rtStrainTyping, jsa.np.rtResistGenes, jsa.util.streamServer, jsa.util.streamClient`

### 4.14.5 Usage examples

If there is a sam/bam file of aligning the Nanopore sequencing to the genome collection, the program can read from this

```
jsa.np.rtSpeciesTyping -bam alignment.sam -index SpeciesTyping/Bacteria/speciesIndex -read 50 -time 60 -out speciesTypingResults.out
```
This program can read data from the output stream of an alignment program to perform analysis in real-time. For example, one can create such a pipeline to listen on port 3456:

```
jsa.util.streamServer -port 3456 \\n| bwa mem -t 10 -k11 -W20 -r10 -A1 -B1 -O1 -E1 -L0 -Y -K 10000 SpeciesTyping/Bacteria/ \n| genomeDB.fasta - 2> /dev/null \\n| jsa.np.rtSpeciesTyping -bam - -index SpeciesTyping/Bacteria/speciesIndex --read 50 - \n| --time 60 -out speciesTypingResults.out 2> speciesTypingResults.log &
```

and streams data to this pipeline using npReader:

```
jsa.np.npreader -GUI -realtime -folder <DownloadFolder> -fail -output data.fastq - 
---stream serverAddress:3456
```

### 4.15 `jsa.np.rtMLST`: Multi-locus Sequencing Typing in real-time with Nanopore sequencing

`jsa.np.rtMLST` performs MLST typing from real-time sequencing with Nanopore MinION.

#### 4.15.1 Synopsis

`jsa.np.rtMLST`: Realtime Multi-Locus Strain Typing using Nanopore Sequencing data

#### 4.15.2 Usage

```
jsa.np.rtMLST [options]
```

#### 4.15.3 Options

- `--output=s` Output file (default='output.dat')
- `--mlstScheme=s` Path to mlst scheme (REQUIRED)
- `--bamFile=s` The bam file (default='null')
- `--qual=d` Minimum alignment quality (default='0.0')
- `--twodonly` Use only two dimensional reads (default='false')
- `--read=i` Minimum number of reads between analyses (default='50')
- `--time=i` Minimum number of seconds between analyses (default='30')
- `--help` Display this usage and exit (default='false')

#### 4.15.4 See also

jsa.np.npreader, jsa.npSpeciesTyping, jsa.npStrainTyping, jsa.np.rtResistGenes, jsa.util.streamServer, jsa.util.streamClient
4.15.5 Setting up

Refer to real-time analysis page at https://github.com/mdcao/npAnalysis/

4.16 jsa.np.rtStrainTyping: Bacterial strain typing with Oxford Nanopore sequencing

*jsa.np.rtStrainTyping* strain types a bacterial sample from Oxford Nanopore sequencing in real-time. It reads data in SAM/BAM format from a file or from a stream and identifies the genes in the samples. Based on the patterns of gene presence, it makes an inference of the strain, together with the confidence interval of 95%.

We provide the gene databases for three bacterial species K. pneumoniae, E. coli and S. aureus on http://data.genomicsresearch.org/Projects/npAnalysis/. Refer to the documentation at https://github.com/mdcao/npAnalysis/ for more details.

4.16.1 Synopsis

*jsa.np.rtStrainTyping*: Realtime strain typing using Nanopore sequencing data

4.16.2 Usage

```
jsa.np.rtStrainTyping [options]
```

4.16.3 Options

- **--geneDB=s** Path to the gene database (REQUIRED)
- **--bamFile=s** The bam file (REQUIRED)
- **--qual=d** Minimum alignment quality (default='0.0')
- **--twodonly** Use only two dimensional reads (default='false')
- **--read=i** Minimum number of reads between analyses (default='50')
- **--time=i** Minimum number of seconds between analyses (default='30')
- **--output=s** Output file (default='output.dat')
- **--help** Display this usage and exit (default='false')

4.16.4 See also

*jsa.np.npreader, jsa.np.rtSpeciesTyping, jsa.np.rtResistGenes, jsa.util.streamServer, jsa.util.streamClient*

4.16.5 Usage examples

If there is a sam/bam file of aligning the Nanopore sequencing to the gene database (ie, geneFam.fasta in one of the said databases), the program can read from this file (note, this is not real-time analysis):
### 4.17 jsa.np.rtResistGenes: Antibiotic resistance gene identification in real-time with Nanopore sequencing

**jsa.np.rtResistGenes** identifies antibiotic resistance genes from real-time sequencing with Nanopore MinION.

#### 4.17.1 Synopsis

**jsa.np.rtResistGenes**: Realtime identification of antibiotic resistance genes from Nanopore sequencing

#### 4.17.2 Usage

```bash
jsa.np.rtResistGenes [options]
```

#### 4.17.3 Options

- **--output=s**: Output file (default=’output.dat’)
- **--bamFile=s**: The bam file (default=’null’)
- **--score=d**: The alignment score threshold (default=’1.0E-4’)
- **--msa=s**: Name of the msa method, support poa, kalign, muscle and clustalo (default=’kalign’)
- **--tmp=s**: Temporary folder (default=’_tmpt’)
- **--resDB=s**: Path to resistance database (REQUIRED)
- **--qual=d**: Minimum alignment quality (default=’0.0’)
- **--twodonly**: Use only two dimensional reads (default=’false’)
- **--read=i**: Minimum number of reads between analyses (default=’50’)
- **--time=i**: Minimum number of seconds between analyses (default=’1800’)
- **--thread=i**: Number of threads to run (default=’4’)

---

This program can read data from the output stream of an alignment program to perform analysis in real-time. For example, one can create such a pipeline to listen on port 3457 and streams data to this pipeline using npReader:

```bash
jsa.util.streamServer -port 3457 \ \
| bwa mem -t 2 -k11 -W20 -r10 -A1 -B1 -O1 -E1 -L0 -Y -K 10000 -a StrainTyping/ \ \
| Escherichia_coli/geneFam.fasta - 2> /dev/null \ \
| jsa.np.rtStrainTyping -bam - -geneDB StrainTyping/Escherichia_coli/ -read 0 -time \ \
| -20 --out EcStrainTyping.dat 2> kPStrainTyping.log
```

```bash
jsa.np.npreader -GUI -realtime -folder <DownloadFolder> -fail -output data.fastq - \ 
--stream serverAddress:3457
```
4.17.4 See also

jsa.np.npreader, jsa.np.rtSpeciesTyping, jsa.np.rtStrainTyping, jsa.util.streamServer, jsa.util.streamClient

4.17.5 Setting up

Refer to the documentation at https://github.com/mdcao/npAnalysis/ for more details.

4.18 npScarf: real-time scaffolder using SPAdes contigs and Nanopore sequencing reads

npScarf (jsa.np.npscarf) is a program that connect contigs from a draft genomes to generate sequences that are closer to finish. These pipelines can run on a single laptop for microbial datasets. In real-time mode, it can be integrated with simple structural analyses such as gene ordering, plasmid forming.

npScarf is included in the Japsa package.

4.18.1 Synopsis

jsa.np.npscarf: Experimental Scaffold and finish assemblies using Oxford Nanopore sequencing reads

4.18.2 Usage

```
jsa.np.npscarf [options]
```

4.18.3 Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--seqFile=s</td>
<td>Name of the assembly file (sorted by length) (REQUIRED)</td>
</tr>
<tr>
<td>--input=s</td>
<td>Name of the input file, - for stdin (REQUIRED)</td>
</tr>
<tr>
<td>--format=s</td>
<td>Format of the input: fastq/fasta or sam/bam (REQUIRED)</td>
</tr>
<tr>
<td>--index</td>
<td>Whether to index the contigs sequence by the aligner or not. (default='true')</td>
</tr>
<tr>
<td>--bwaExe=s</td>
<td>Path to bwa (default='bwa')</td>
</tr>
<tr>
<td>--bwaThread=i</td>
<td>Threads used by bwa (default='4')</td>
</tr>
<tr>
<td>--long</td>
<td>Whether report all sequences, including short/repeat contigs (default) or only long/unique/completed sequences. (default='false')</td>
</tr>
<tr>
<td>--spadesDir=s</td>
<td>Name of the output folder by SPAdes: assembly graph and paths will be used for better gap-filling. (default='null')</td>
</tr>
<tr>
<td>--prefix=s</td>
<td>Prefix for the output files (default='out')</td>
</tr>
<tr>
<td>--genes=s</td>
<td>Realtime annotation: name of annotated genes in GFF 3.0 format (default='null')</td>
</tr>
</tbody>
</table>
--resistGene=s  Realtime annotation: name of antibiotic resistance gene fasta file (default='null')
--insertSeq=s  Realtime annotation: name of IS fasta file (default='null')
--oriRep=s  Realtime annotation: name of fasta file containing possible origin of replication (default='null')
--minContig=i  Minimum contigs length that are used in scaffolding. (default='300')
--maxRepeat=i  Maximum length of repeat in considering species. (default='7500')
--cov=d  Expected average coverage of Illumina, <=0 to estimate (default='0.0')
--qual=i  Minimum quality (default='1')
--support=i  Minimum supporting long read needed for a link between markers (default='1')
--realtime  Process in real-time mode. Default is batch mode (false) (default='false')
--read=i  Minimum number of reads between analyses (default='50')
--time=i  Minimum number of seconds between analyses (default='10')
--verbose  Turn on debugging mode (default='false')
--help  Display this usage and exit (default='false')

4.18.4 See also
jsa.np.npreader, jsa.util.streamServer, jsa.util.streamClient

4.18.5 Usage examples
A summary of npScarf usage can be obtained by invoking the –help option:

```
jsa.np.npscarf --help
```

**Input**

npScarf takes two files as required input:

```
jsa.np.npscarf -seq <draft> -input <nanopore>
```

<draft> input is the FASTA file containing the pre-assemblies. Normally this is the output from running SPAdes on Illumina MiSeq paired end reads.

<nanopore> is either the long reads in FASTA/FASTQ file or SAM/BAM formatted alignments between them to <draft> file. We use BWA-MEM as the recommended aligner with the fixed parameter set as follow:

```
bwa mem -k11 -W20 -r10 -A1 -B1 -O1 -E1 -L0 -a -Y <draft> <nanopore> > <bam>
```

The input file format is specified by option –format. The default is FASTA/FASTQ in which the path to BWA version 0.7.11 or newer is required. Remember to always INDEXING the reference before running BWA.
bwa index <draft>

Missing this step would break down the whole pipeline.

Output

`npScarf` output is specified by `-prefix` option. The default prefix is ‘out’. Normally the tool generate two files: `prefix.fin.fasta` and `prefix.fin.japsa` which indicate the result scaffolders in FASTA and JAPSA format.

In realtime mode, if any annotation analysis is enabled, a file named `prefix.anno.japsa` is generated instead. This file contains features detected after scaffolding.

Real-time scaffolding

To run `npScarf` in streaming mode:

```
jsa.np.npreader -realtime -folder c:\Downloads\ -fail -output - | \
jsa.np.npscarf --realtime -bwaExe=<path_to_BWA> -bwaThread=10 -input - -seq <draft> > log.out 2>&1
```

For the same purpose, you can also invoke BWA-MEM explicitly as in the old version of `npScarf`. In this case, option `-format=SAM` must be presented as follow:

```
jsa.np.npreader -realtime -folder c:\Downloads\fail -output - | \
jsa.np.npscarf --realtime -bwaExe=<path_to_BWA> -bwaThread=10 -input - -seq <draft> -format=SAM -output - | \
```

or if you have the whole set of Nanopore long reads already and want to emulate the streaming mode:

```
jsa.np.timeEmulate -s 100 -i <nanopore> -output - | \
jsa.np.npscarf --realtime -bwaExe=<path_to_BWA> -bwaThread=10 -input - -seq <draft> > log.out 2>&1
```

Note that `jsa.np.timeEmulate` based on the field `timestamp` located in the read name line to decide the order of streaming data. So if your input `<nanopore>` already contains the field, you have to sort it:

```
jsa.seq.sort -i <nanopore> -o <nanopore-sorted> -sortKey=timestamp
```

or if your file does not have the `timestamp` data yet, you can manually make ones. For example:

```
cat <nanopore> | \
awk '{BEGIN{time=0.0}NR%4==1{printf "%s timestamp=%.2f\n", $0, time; time++}NR%4!=1
{print}" <nanopore with-time>
```
Real-time annotation

The tool includes usecase for streaming annotation. Ones can provides database of antibiotic resistance genes and/or Origin of Replication in FASTA format for the analysis of gene ordering and/or plasmid identifying respectively:

```bash
jsa.np.timeEmulate -s 100 -i <nanopore> -output - | \n    jsa.np.npscarf --realtime -bwaExe=<path_to_bwa> -input - -seq <draft> -resistGene
    <-resistDB> -oriRep <origDB> > log.out 2>&1
```

Assembly graph

`npScarf` can read the assembly graph info from SPAdes to make the results more precise. The results might be slightly deviate from the old version in term of number of final contigs:

```bash
jsa.np.npscarf --spadesFolder=<SPAdes_output_directory> <options...>
```

where SPAdes_output_directory indicates the result folder of SPAdes, containing files such as contigs.fasta, contigs.paths and assembly_graph.fastg.

4.19 barcode: real-time de-multiplexing Nanopore reads from barcode sequencing

`barcode` (jsa.np.barcode) is a program that demultiplex the nanopore reads from Nanopore barcode sequencing. Downstream analysis can be invoked concurrently by an input script.

`barcode` is included in the Japsa package.

4.19.1 Synopsis

`jsa.np.barcode`: Clustering nanopore sequences based on barcode

4.19.2 Usage

```bash
jsa.np.barcode [options]
```

4.19.3 Options

- `--bcFile=s` Bar code file (REQUIRED)
- `--seqFile=s` Nanopore sequences file (REQUIRED)
- `--scriptRun=s` Invoke command script to run npScarf (default='null')
- `--threshold=d` Minimum identity(%) for barcode alignment (default='70.0')
- `--distance=d` Minimum identity(%) distance between the best alignment to others (default='4.0')
- `--twoends` Whether a read must contain barcode sequence from both ends or just one end (default) (default='false')
--print Print out demultiplexed reads to corresponding FASTA file or not.
    (default='false')
--help Display this usage and exit (default='false')

4.19.4 Usage examples

A summary of barcode usage can be obtained by invoking the --help option:

```
jsa.np.barcode --help
```

Input

barcode takes 2 files as required input:

```
jsa.np.barcode -seq <nanopore reads> -bc <barcode.fasta>
```

<nanopore reads> is either the long reads in FASTA/FASTQ file (after MinION sequencing is finished) or standard input (specified by "-", for real-time analysis).

<barcode.fasta> is the FASTA file of barcode sequences (given by ONT) with name correspond to the assigned sample id.

Missing any file would break down the whole pipeline.

In addition, one can provide <analysis_script> which is the script call for further action on the de-multiplexed reads. It always take one argument and be executable by invoking:

```
./analysis_script <id>
```

in which <id> is the identifier of a sample as given in the <barcode.fasta>. The script should read the standard input of long-read streams to do further analysis.

barcode allows user to set the minimum criteria of a hit with barcode reference to be considered valid. The default value is 70% for minimum identity. At the same time, 4% distance between the best hit and the second best is necessary for differentiation. Decreasing the thresholds will lead to more reads being clustered but with higher risk of false positive while more stringent parameters will generate less but more confident of demultiplexed reads.

User can also have control on the matching condition for barcode detection, either one-end match or both-end match. For the first case (default), only the a legal maximal hit from one end of a read is enough to label it while in the later case, we take into account a pair from both 5’ and 3’terminus. Thus the input for each use case should be different. The one-end option can take the simple FASTA file of Nanopore barcodes while the two-end need pairs of barcode to be specified (e.g. with _F and _R suffix). One of a typical use case for two-end matching is when we want to detect the super-barcode which includes also tail- and primer-sequences in pre-defined orientation.

Output

barcode output depends on the <analysis script> because the de-multiplexed reads are streamed directly to its dedicated process. If ones only interest in de-multiplexing alone, then the script should be as simple as to write stream to file. For example:

```
#!/bin/bash
while read line
do
    # code
```

(continues on next page)
4.20 `jsa.util.streamServer`: Receiving streaming data over a network

`jsa.util.streamServer` implements a server that listen at a specified port. Upon receiving data from a client, it forwards the stream data to standard output. `jsa.util.streamServer` and `jsa.util.streamClient` can be used to set up streaming applications such as real-time analyses. By default, the server listens on port 3456, unless specified otherwise.

4.20.1 Synopsis

`jsa.util.streamServer`: Listen for input from a stream and forward the streamed data to the standard output

4.20.2 Usage

```bash
jsa.util.streamServer [options]
```

4.20.3 Options

```
--port=i  Port to listen to (default=’3456’)
--help   Display this usage and exit (default=’false’)
```
4.20.4 See also

jsa.util.streamClient, jsa.np.filter, jsa.np.nreader

4.21 jsa.util.streamClient: Streams data over a network

*jsa.util.streamClient* streams data over the network to a listening server (*jsa.util.streamServer*).

4.21.1 Synopsis

*jsa.util.streamClient*: Forward data from a stream input or a file over the network to a jsa.util.streamServer

4.21.2 Usage

```
jsa.util.streamClient [options]
```

4.21.3 Options

- **--input=s**: Name of the input file, - for standard input (REQUIRED)
- **--server=s**: Stream output to one or more servers, format IP:port,IP:port (REQUIRED)
- **--help**: Display this usage and exit (default='false')

4.21.4 See also

jsa.util.streamServer, jsa.np.filter, jsa.np.nreader

4.22 XMas: Robust estimation of genetic distances with information theory

*XMas* (*jsa.phylo.xmas*) is a tool to measure genetics distances between aligned sequences. It reads in a list of sequences from a fasta file format, and outputs a distance matrix in the format required by the PHYLIP package to run neighbour joining.

*XMas* is included in the Japsa package. Please see check the installation page for instructions.

4.22.1 Synopsis

*jsa.phylo.xmas*: Generate a distance matrix from aligned sequences

4.22.2 Usage
4.22.3 Options

```
jsa.phylo.xmas [options]
```

**4.22.3 Options**

```
--input=s
Name of the input file, - for standard input (REQUIRED)
--output=s
Name of the file for output (distances in phylip format) (default='output')
--adapt
Use adaptive (default='false')
--help
Display this usage and exit (default='false')
```

4.22.4 Usage samples

At the moment, XMas is designed to worked with aligned sequences, with indels and wildcards (e.g., N) removed. XMas reads in these aligned sequences from a fasta file, and output the distances to a file in a format ready to run neighbour-joining with PHYLIP. For examples:

```
jsa.phylo.xmas -i sequences.fas -o infile
```

And run phylip neighbour-joining from the distances in *infile*:

```
phylip neighbor
```

4.23 *jsa.phylo.normalise*: Normalise branch length of a phylogeny

*jsa.phylo.normalise* scales the branches of a phylogeny so that their sum equates to a value.

*jsa.phylo.normalise* is included in the Japsa package. Please see check the installation page for instructions.

**4.23.1 Synopsis**

```
jsa.phylo.normalise [options]
```

**4.23.2 Usage**

```
jsa.phylo.normalise [options]
```

**4.23.3 Options**

```
--input=s
Name of the input file, - for standard input (REQUIRED)
--sum=d
Sum of branches after normalising (default='1.0')
--scale=d
Scale factor, if set to a positive number will override the sum parameter (default='0.0')
--output=s
Name of the file for output, - for stdout (default='-')
--help
Display this usage and exit (default='false')
```
4.24 **capsim**: Simulating the Dynamics of Targeted Capture Sequencing with CapSim

capsim (jsa.sim.capsim) is a tool to simulate target capture sequencing. Its simulates the dynamics of capture process

### 4.24.1 Synopsis

csa.sim.capsim: Simulate capture sequencing

### 4.24.2 Usage

```
jsa.sim.capsim [options]
```

### 4.24.3 Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--reference=s</td>
<td>Name of genome to be (REQUIRED)</td>
</tr>
<tr>
<td>--probe=s</td>
<td>File containing probes mapped to the reference in bam format (default='null')</td>
</tr>
<tr>
<td>--logFile=s</td>
<td>Log file (default='*')</td>
</tr>
<tr>
<td>--ID=s</td>
<td>A unique ID for the data set (default='*')</td>
</tr>
<tr>
<td>--miseq=s</td>
<td>Name of read file if miseq is simulated (default='null')</td>
</tr>
<tr>
<td>--pacio=s</td>
<td>Name of read file if pacbio is simulated (default='null')</td>
</tr>
<tr>
<td>--fmedian=i</td>
<td>Median of fragment size at shearing (default='2000')</td>
</tr>
<tr>
<td>--fsshape=d</td>
<td>Shape parameter of the fragment size distribution (default='6.0')</td>
</tr>
<tr>
<td>--smedian=i</td>
<td>Median of fragment size distribution (default='1300')</td>
</tr>
<tr>
<td>--sshape=d</td>
<td>Shape parameter of the fragment size distribution (default='6.0')</td>
</tr>
<tr>
<td>--tmedian=i</td>
<td>Median of target fragment size (the fragment size of the data). If specified, will override fmedian and smedian. Otherwise will be estimated (default='0')</td>
</tr>
<tr>
<td>--tshape=d</td>
<td>Shape parameter of the effective fragment size distribution (default='0.0')</td>
</tr>
<tr>
<td>--num=i</td>
<td>Number of fragments (default='1000000')</td>
</tr>
<tr>
<td>--pblen=i</td>
<td>PacBio: Average (polymerase) read length (default='30000')</td>
</tr>
<tr>
<td>--illen=i</td>
<td>Illumina: read length (default='300')</td>
</tr>
<tr>
<td>--ilmode=s</td>
<td>Illumina: Sequencing mode: pe = paired-end, mp=mate-paired and se=singled-end (default='pe')</td>
</tr>
<tr>
<td>--seed=i</td>
<td>Random seed, 0 for a random seed (default='0')</td>
</tr>
<tr>
<td>--help</td>
<td>Display this usage and exit (default='false')</td>
</tr>
</tbody>
</table>
4.24.4 Usage samples

4.25 Expert Model: tool for compression of genomic sequences

`jsa.xm.compress` in the implementation of the expert model (XM) algorithm for compression of genomics sequences. The source code is included in the Japsa package. Please see the installation page for instructions.

4.25.1 Synopsis

`jsa.xm.compress`: Compression of DNA/protein sequences

4.25.2 Usage

```
jsa.xm.compress [options] file1 file2 ...
```

4.25.3 Options

- `--hashSize=i` Hash size (default='11')
- `--context=i` Length of the context (default='15')
- `--limit=i` Expert Limit (default='200')
- `--threshold=d` Listen threshold (default='0.15')
- `--chance=i` Chances (default='20')
- `--binaryHash` Use binary hash or not (default='false')
- `--offsetType=s` Way of update offset/palindrome expert: possible value count, subs (default='counts')
- `--real=s` File name of the real compression (default='null')
- `--decode=s` File name of the encoded (default='null')
- `--output=s` The output file of decoded file (default='decoded')
- `--info=s` File name of the information content (default='null')
- `--markov=s` File name of the markov information content (default='null')
- `--optimise` Running in optimise mode, just report the entropy, recommended for long sequence (default='false')
- `--checkPoint=i` Frequency of check point (default='1000000')
- `--hashType=s` Type of Hash table: hash=hashtable, sft=SuffixTree, sfa = SuffixArray (default='hash')
- `--selfRep` Propose experts from the sequence to compressed? (default='true')
- `--help` Display this usage and exit (default='false')
4.25.4 Citation

If you find XM useful for your research, please cite

Japsa is currently maintained by Minh Duc Cao. The following people have contributed to the development of Japsa, including ideas, algorithms, implementation, documentation and feedback:

- Allen Day
- Lachlan Coin
- Son Hoang Nguyen
- Mikael Boden
- Lloyd Allison
- Trevor I Dix
- Hoang Anh Nguyen
- Julia Bernal
- David Powell
- Christopher Mears
- Micheal Hall

Japsa is developed mainly in Eclipse. Its repository is regularly pushed to GitHub and a bit lesser to GitBucket. Documentation is written in reStructuredText using Sphinx and hosted by Read the docs.
Japa’s source code is available under a BSD-like license:

Copyright (c) Minh Duc Cao, Monash Uni & UQ, All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the names of the institutions nor the names of the contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
CHAPTER 7

Indices and tables

- genindex
- modindex
- search