MiXCR Documentation

Release SNAPSHOT

MiLaboratory

Jul 24, 2017

Getting started

MiXCR is a universal framework that processes big immunome data from raw sequences to quantitated clonotypes. MiXCR efficiently handles paired- and single-end reads, considers sequence quality, corrects PCR errors and identifies germline hypermutations. The software supports both partial- and full-length profiling and employs all available RNA or DNA information, including sequences upstream of V and downstream of J gene segments.

MiXCR is free for academic and non-profit use (see *License*).

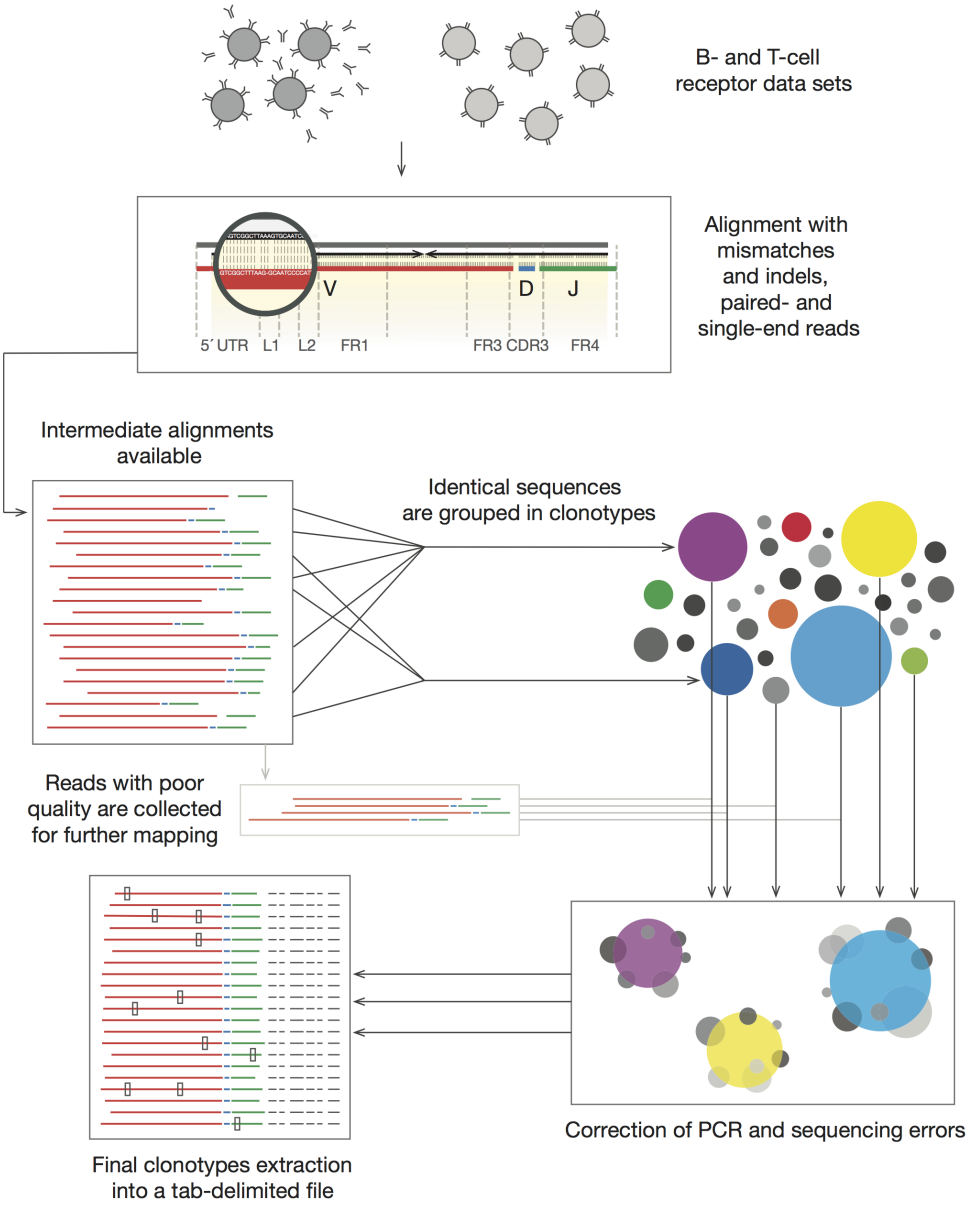

Fig. 1: *MiXCR pipeline. The workflow from IG or T-cell receptor data sets to final clonotypes is shown*

Getting started Main actions Special cases In-depth topics

CHAPTER 1

Installation

System requirements

- Any Java-enabled platform (Windows, Linux, Mac OS X)
- Java version 8 or higher (download from Oracle web site)
- 1–16 Gb RAM (depending on number of clones in the sample)

Installation on Mac OS X / Linux using Homebrew

Homebrew is a simple package manager developed for Mac OS X and also ported to Linux. To install MiXCR using Homebrew just type the following commands:

```
brew tap milaboratory/all
brew install mixcr
```

Installation on Mac OS X / Linux / FreeBSD from zip distribution

- Check that you have Java **1.8+** installed on your system by typing `java -version`. Here is the example output of this command:

```
> java -version
java version "1.8.0_66"
Java(TM) SE Runtime Environment (build 1.8.0_66-b17)
Java HotSpot(TM) 64-Bit Server VM (build 25.66-b17, mixed mode)
```

- download latest binary distributaion of MiXCR from the release page on GitHub
- unzip the archive

- add extracted folder of MiXCR distribution to your `PATH` variable or add symbolic link for `mixcr` script to your `bin/` folder (e.g. `~/bin/` in Ubuntu and many other popular linux distributions)

Installation on Windows

Currently there is no execution script or installer for Windows. Still MiXCR can easily be used by direct execution from the jar file.

- check that you have Java **1.7+** installed on your system by typing `java -version`
- download latest binary distributaion of MiXCR from the release page on GitHub
- unzip the archive
- use `mixcr.jar` from the archive in the following way:

```
> java -Xmx4g -Xms3g -jar path_to_mixcr\jar\mixcr.jar ...
```

For example:

```
> java -Xmx4g -Xms3g -jar C:\path_to_mixcr\jar\mixcr.jar align input.fastq.
↪gz output.vdjca
```

To use mixcr from `jar` file one need to substitute `mixcr` command with `java -Xmx4g -Xms3g -jar path_to_mixcr\jar\mixcr.jar` in all examples from this manual.

CHAPTER 2

Quick start

Overview

Typical MiXCR workflow consists of three main processing steps:

- *align*: align sequencing reads to reference V, D, J and C genes of T- or B- cell receptors
- *assemble*: assemble clonotypes using alignments obtained on previous step (in order to extract specific gene regions e.g. CDR3)
- *export*: export alignment (`exportAlignments`) or clones (`exportClones`) to human-readable text file

MiXCR supports the following formats of sequencing data: `fasta`, `fastq`, `fastq.gz`, paired-end `fastq` and `fastq.gz`. As an output of each processing stage, MiXCR produces binary compressed file with comprehensive information about entries produced by this stage (alignments in case of `align` and clones in case of `assemble`). Each binary file can be converted to a human-readable/parsable tab-delimited text file using `exportAlignments` and `exportClones` commands.

Basic parameters

There are many parameters that user can change to adapt MiXCR for particular needs. While all these parameters are optional there is a set of parameters that are worth considering before running the analysis:

- `-OvParameters.geneFeatureToAlign` sets the gene feature of V gene used for alignment. Applied on the *alignment* stage. Choice of the value for this parameter depends on the type of starting material and library preparation strategy used. There are three options covering most of the cases (see *Gene Features* for the full list):
  - `VRegion` **(default)** is generally suitable for majority of use cases, on the other hand if you have some additional information about your library it is a good idea to use one of the values mentioned below instead of default. Don't change the default value if your library is prepared using multiplex PCR on the V gene side.

  - `VTranscript` if RNA was used as a starting material and some kind of non-template-specific technique was used for further amplification on the 5'-end of RNA (e.g. 5'RACE) (see *example*). Using of this option is useful for increasing of sequencing information utilization from 5'-end of the molecule, which in turn helps to increase accuracy of V gene identification.
  - `VGene` if DNA was used as a starting material and 5' parts of V gene (including V intron, leader sequence and 5'UTR) are supposed to be present in your data. Using of this option is useful for increasing of sequencing information utilization from 5'-end of the molecule, which in turn helps to increase accuracy of V gene identification.

  Use `VTranscript` or `VGene` if you plan to assemble full-length clonotypes (including all FRs and CDRs) of T- or B- cell receptors.

- The `-OassemblingFeatures` parameter sets the region of TCR/BCR sequence which will be used to assemble clones. Applyed on the *assembly* stage. By default its value is `CDR3` which results in assembling of clones by the sequence of *Complementarity Determining Region 3*. To analyse full length sequences use `VDJRegion` as a value for the `assemblingFeatures` (see *Gene Features* for more details).
- Another important parameter is `--species`, it sets the target organism. This parameter is used on the *align* stage. Possible values are `hsa` (or `HomoSapiens`) and `mmu` (or `MusMusculus`). Default value is `hsa`. This parameter should be supplied on the alignment stage (see *example*).

The following sections describes common use cases

Examples

Default workflow

Tip: Parameters used in this example are particularly suitable for analysis of **multiplex-PCR** selected fragments of T-/B- cell receptor genes.

MiXCR can be used with the default parameters in most cases by executing the following sequence of commands:

```
> mixcr align input_R1.fastq input_R2.fastq alignments.vdjca

... Building alignments

> mixcr assemble alignments.vdjca clones.clns

... Assembling clones

> mixcr exportClones --chains IGH clones.clns clones.txt

... Exporting clones to tab-delimited file
```

The value of only one parameter is changed from its default in this snippet (`--chains IGH`) to tell MiXCR to export only IGH sequences. However even this parameter can be omitted (in this case MiXCR will export all T-/B-cell receptor sequences, that have been found in the sample). We reccomend always specify `--chain` parameter at the `exportClones step` to simplify further analysis.

The file produced (`clone.txt`) will contain a tab-delimited table with information about all clonotypes assembled by CDR3 sequence (clone abundance, CDR3 sequence, V, D, J genes, etc.). For full length analysis and other useful features see examples below.

Analysis of data obtained using 5'RACE-based amplification protocols

Consider MiXCR workflow in more detail on analysis of paired-end sequenced cDNA library of IGH gene prepared using 5'RACE-based protocol (i.e. on read covers CDR3 with surroundings and another one covers 5'UTR and downstream sequence of V gene):

1. *Align* raw sequences to reference sequences of segments (V, D, J) of IGH gene:

```
> mixcr align -OvParameters.geneFeatureToAlign=VTranscript \
  --report alignmentReport.log input_R1.fastq input_R2.fastq alignments.vdjca
```

   Here we specified non-default value for gene feature used to align V genes (`-OvParameters.geneFeatureToAlign=VTranscript`) in order to utilize information from both reads, more specifically to let MiXCR align V gene's 5'UTRS and parts of coding sequence on 5'-end with sequence from read opposite to CDR3. MiXCR can also produce report file (specified by optional parameter `--report`) containing run statistics which looks like this:

```
Analysis Date: Mon Aug 25 15:22:39 MSK 2014
Input file(s): input_r1.fastq,input_r2.fastq
Output file: alignments.vdjca
Command line arguments: align --report alignmentReport.log input_r1.fastq␣
↪input_r2.fastq alignments.vdjca
Total sequencing reads: 323248
Successfully aligned reads: 210360
Successfully aligned, percent: 65.08%
Alignment failed because of absence of V hits: 4.26%
Alignment failed because of absence of J hits: 30.19%
Alignment failed because of low total score: 0.48%
```

   One can convert binary output produced by `align` (`output.vdjca`) to a human-readable text file using *exportAlignments* command.

2. *Assemble* clonotypes:

```
> mixcr assemble --report assembleReport.log alignments.vdjca clones.clns
```

   This will build clonotypes and additionally correct PCR and sequencing errors. By default, clonotypes will be assembled by CDR3 sequences; one can specify another gene region by passing additional command line arguments (see *assemble documentation*). The optional report `assembleReport.log` will look like:

```
Analysis Date: Mon Aug 25 15:29:51 MSK 2014
Input file(s): alignments.vdjca
Output file: clones.clns
Command line arguments: assemble --report assembleReport.log alignments.
↪vdjca clones.clns
Final clonotype count: 11195
Total reads used in clonotypes: 171029
Reads used, percent of total: 52.89%
Reads used as core, percent of used: 92.04%
Mapped low quality reads, percent of used: 7.96%
Reads clustered in PCR error correction, percent of used: 0.04%
Clonotypes eliminated by PCR error correction: 72
Percent of reads dropped due to the lack of clonal sequence: 2.34%
Percent of reads dropped due to low quality: 3.96%
Percent of reads dropped due to failed mapping: 5.87%
```

3. *Export* binary file with a list of clones (`clones.clns`) to a human-readable text file:

```
> mixcr exportClones --chains IGH clones.clns clones.txt
```

This will export information about clones with default set of fields, e.g.:

Clone count	Clone fraction	...	V hits	J hits		14. seq. CDR3	AA. seq. CDR3...
4369	2.9E-3	...	IGHV4-39*00(1388)	IGHJ6 *00(131)	TGTGTGAG...	CVRHKPM...	...
3477	2.5E-3	...	IGHV4-34*00(1944)	IGHJ4 *00(153)	TGTGCGAT...	CAIWDVGL...	...
...	...	...	...	...	...	...	...

where dots denote text not shown here (for compactness). For the full list of available export options see *export* documentation.

Each of the above steps can be customized in order to adapt the analysis pipeline for a specific research task (see below).

High quality full length IG repertoire analysis

For the full length cDNA-based immunoglobulin repertoire analysis we generally recommend to prepare libraries with unique molecular identifiers (UMI) and sequence them using asymmetric paired-end 350 bp + 100 bp Illumina MiSeq sequencing (see Nature Protocols paper). This approach allows to obtain long-range high quality sequencing and efficiently eliminate PCR and sequencing errors using MiGEC software. The resulting high quality data can be further processed by MiXCR for the efficent full length IGH or IGL repertoire extraction:

1. Merging paired-end reads and *alignment*:

   MiXCR's `align` subcommand performs paired-end reads merging and alignment to reference V/D/J and C genes. We recommend using *KAligner2* (currently in beta testing) for the full length immunoglobulin profiling:

   ```
   > mixcr align -p kaligner2 -s hsa -r alignmentReport.txt -
   ↪OreadsLayout=Collinear \
     -OvParameters.geneFeatureToAlign=VTranscript read_R1.fastq.gz read_R2.
   ↪fastq.gz \
     alignments.vdjca
   ```

   Option `-s` allows to specify species (e.g. homo sapiens - `hsa`, mus musculus - `mmu`). Parameter `-OreadsLayout` allow us to set paired-end reads orientation (`Collinear`, `Opposite`, `Unknown`). Note, that after MiGEC analysis paired-end read pairs are in `Collinear` orientation.

   Instead of KAligner2, default MiXCR aligner can be used as well, but it may miss immunoglobulin subvariants that contain several nucleotide-lengths indels within the V gene segment.

2. *Assemble* clones:

   ```
   > mixcr assemble -p default_affine -r assembleReport.txt -
   ↪OassemblingFeatures=VDJRegion \
     -OseparateByC=true -OqualityAggregationType=Average \
   ```

```
  -OclusteringFilter.specificMutationProbability=1E-5 -
↪OmaxBadPointsPercent=0 \
  alignments.vdjca clones.clns
```

`default_affine` parameter is specifically required for the data aligned using KAligner2 (use this option only if `-p kaligner2` was used on the alignemnt step)

`-OseparateByC=true` separates clones with different antibody isotype.

Set `-OcloneClusteringParameters=null` parameter to switch off the frequency-based correction of PCR errors.

Depending on data quality, one can adjust input threshold by changing the parameter `-ObadQualityThreshold` to improve clonotypes extraction.

See "Assembler parameters" section of documentation for the advanced quality filtering parameters.

3. *Export* clones:

```
> mixcr exportClones -c IGH -o -t clones.clns clones.txt
```

where options `-o` and `-t` filter off the out-of-frame and stop codon containing clonotypes, respectively, and `-c` indicates which chain will be extracted (e.g. `IGH`, `IGL`).

Analysis of RNA-Seq data

For detailed description please see *here*.

MiXCR allows to extract TCR and BCR CDR3 repertoires from RNA-Seq data. Extraction efficiency depends on the abundance of T or B cells in a sample, and also on the sequencing length. 2x150 bp or 2x100 bp paired-end sequencing is recommended. However, even from the paired-end 2x50 bp RNA-Seq data, information on the major clonotypes present (e.g. in a tumor sample) can usually be extracted. The analysis can be perfromed in the following way:

1. *Align* reads:

```
> mixcr align -p rna-seq -OallowPartialAlignments=true data_R1.fastq.gz data_
↪R2.fastq.gz alignments.vdjca
```

All `mixcr align` parameters can also be used here (e.g. `-s` to specify organism).

`-OallowPartialAlignments=true` option preserves partial alignments for their further use in `assemblePartial`.

2. *Assemble parial reads*:

```
> mixcr assemblePartial alignments.vdjca alignmentsRescued.vdjca
```

To obtain more assembled reads containing full CDR3 sequence it is recommended to perform several iterations of reads assembling using `mixcr assemblePartial` action. `-p` parameter is required for several iterations. In our experience, the best result is obtained after the second iteration:

```
> mixcr assemblePartial alignments.vdjca alignmentsRescued_1.vdjca

> mixcr assemblePartial alignmentsRescued_1.vdjca alignmentsRescued_2.vdjca
```

3. Extend TCR alignments with uniquely determined V and J genes and having incomplete coverage of CDR3s using germline sequences:

```
> mixcr extendAlignments alignmentsRescued_2.vdjca alignmentsRescued_2_
↪extended.vdjca
```

4. *Assemble* clones:

```
> mixcr assemble alignmentsRescued_2_extended.vdjca clones.clns
```

All `mixcr assemble` parametrs can also be used here.

- For poor quality data it is recommended to decrease input quality threshold (e.g. `-ObadQualityThreshold=15`).
- To make error correction algorithms to combine clone abundancies add the following option: `-OaddReadsCountOnClustering=true`

5. *Exporting* clones:

```
> mixcr exportClones -c TRA -o -t clones.clns clones.txt
```

One can specify immune receptor chain of interest to extract (`-c TRA` or `-c TRB`, etc) and exclude out-of-frame (option `-o`) and stop codon containing variants (option `-t`).

Assembling of CDR3-based clonotypes for mouse TRB sample

This example shows how to perform routine assembly of clonotypes (based on CDR3 sequence) for mouse TRB library (aligning is performed for all possible genes - TRA/B/D/G and IGH/L/K, but only TRB clones are exported in the final table at the end).

```
> mixcr align --species mmu input_R1.fastq input_R2.fastq alignments.vdjca
```

Other analysis stages can be executed without any additional parameters:

```
> mixcr assemble alignments.vdjca clones.clns

> mixcr exportClones --chains TRB clones.clns clones.txt
```

Saving links between initial reads and clones

In this example we demonstrate how to extract initial read headers for assembled clonotypes. On the `align` step additional `--save-description` option should be specified in order to store headers from reads in the resulting `.vdjca` file:

```
> mixcr align --save-description input_R1.fastq input_R2.fastq alignments.vdjca
```

On the `assemble` stage it is necessary to specify file for the index (which stores mapping from reads to clonotypes):

```
> mixcr assemble --index indexFile alignments.vdjca clones.clns
```

Having this, it is possible to export original read headers with corresponding clone IDs:

```
> mixcr exportAlignments -cloneId indexFile -descrR1 -descrR2 alignments.vdjca␣
↪alignments.txt
```

The resulting file `alignments.txt` will looks like:

Clone ID	Description R1	Description R2
10	header_1_R1	header_1_R2
	header_2_R1	header_2_R2
2313	header_3_R1	header_3_R2
88142	header_5_R1	header_5_R2
...	...	...

CHAPTER 3

Alignment

The `align` command aligns raw sequencing reads to reference V, D, J and C genes of T- and B- cell receptors. It has the following syntax:

```
mixcr align [options] input_file1 [input_file2] output_file.vdjca
```

MiXCR supports `fasta`, `fastq`, `fastq.gz` and paired-end `fastq` and `fastq.gz` input. In case of paired-end reads two input files should be specified.

Command line parameters

The following table contains description of command line options for `align`:

Option	Default value	Description
-h, --help		Print help message.
-r {file} --report ...		Report file name. If this option is not specified, no report file be produced.
- {chain} --chains ...	ALL	Target immunological chain list separated by ",". Available values: IGH, IGL, IGK, TRA, TRB, TRG, TRD, IG (for all immunoglobulin chains), TCR (for all T-cell receptor chains), ALL (for all chains) . It is highly recomended to use the default value for this parameter in most cases at the align step. Filltering is also possible at the export step.
-s {speciesName} --species ...	HomoSapiens	Species (organism). Possible values: hsa (or HomoSapiens), mmu (or MusMusculus), rat (currently only TRB, TRA and TRD are supported), or any species from imported IMGT ® library import as described here *import segments*
-p {parameterName} --parameters ...	default	Preset of parameters. Possible values: default and rna-seq. The rna-seq preset are specifically optimized for analysis of Rna-Seq data *(see below)*
-t {numberOfThreads} --threads ...	number of available CPU cores	Number of processing threads.
-n {numberOfReads} --limit ...		Limit number of sequences that will be analysed (only first -n sequences will be processed from input file(s)).
-a, --save-description		Copy read(s) description line from .fastq or .fasta to .vdjca file (can be then exported with -descrR1 and -descrR2 options in *exportAlignments* action).
-v, --write-all		Write alignment results for all input reads: including empty results for non-aligned reads.
-g, --save-reads		Copy read(s) from .fastq or .fasta to .vdjca file (this is required for exporting reads aggregated by clones; see *this section*).
--not-aligned-R1		Write all not aligned reads (R1) to the specified file.
--not-aligned-R2		Write all not aligned reads (R) to the specified file.
-Oparameter=value		Overrides default value of aligner parameter (see next subsection).

All parameters are optional.

Aligner parameters

MiXCR uses a wide range of parameters that controls aligner behaviour. There are some global parameters and gene-specific parameters organized in groups: vParameters, dParameters, jParameters and cParameters. Each group of parameters may contain further subgroups of parameters etc. In order to override some parameter value one can use -O followed by fully qualified parameter name and parameter value (e.g. -Ogroup1.group2.parameter=value).

One of the key MiXCR features is ability to specify particular *gene regions* which will be extracted from reference and used as a targets for alignments. Thus, each sequencing read will be aligned to these extracted reference regions. Parameters responsible for target gene regions are:

Parameter	Default value	Description
`vParameters.geneFeatureToAlign`	`VRegion`	region in V gene which will be used as target in `align`
`dParameters.geneFeatureToAlign`	`DRegion`	region in D gene which will be used as target in `align`
`jParameters.geneFeatureToAlign`	`JRegion`	region in J gene which will be used as target in `align`
`cParameters.geneFeatureToAlign`	`CExon1`	region in C gene which will be used as target in `align`

It is important to specify these gene regions such that they will fully cover target clonal gene region which will be used in *assemble* (e.g. CDR3).

One can override default gene regions in the following way:

```
mixcr align -OvParameters.geneFeatureToAlign=VTranscript input_file1 [input_file2]␣
↪output_file.vdjca
```

Other global aligner parameters are:

Parameter	Default value	Description
`allowChimeras`	`false`	Accept alignments with different loci of V and J genes (by default such alignments are dropped).
`minSumScore`	`120.0`	Minimal total alignment score value of V and J genes.
`maxHits`	`5`	Maximal number of hits for each gene type: if input sequence align to more than `maxHits` targets, then only top `maxHits` hits will be kept.
`minimalClonalSequenceLength`	`12`	Minimal clonal sequence length (e.g. minimal sequence of CDR3 to be used for clone assembly)
`vjAlignmentOrder` (*only for single-end analysis*)	`VThenJ`	Order in which V and J genes aligned in target (possible values `JThenV` and `VThenJ`). Parameter affects only *single-read* alignments and alignments of overlapped paired-end reads. Non-overlaping paired-end reads are always processed in `VThenJ` mode. `JThenV` can be used for short reads (~100bp) with full (or nearly full) J gene coverage.
`relativeMinVFR3CDR3Score` (*only for paired-end analysis*)	`0.7`	Relative minimal alignment score of `FR3+VCDR3Part` region for V gene. V hit will be kept only if its `FR3+VCDR3Part` part aligns with score greater than `relativeMinVFR3CDR3Score * maxFR3CDR3Score`, where `maxFR3CDR3Score` is the maximal alignment score for `FR3+VCDR3Part` region among all of V hits for current input reads pair.
`readsLayout` (*only for paired-end analysis*)	`Opposite`	Relative orientation of paired reads. Available values: `Opposite`, `Collinear`, `Unknown`.

One can override these parameters in the following way:

```
mixcr align -OmaxHits=3 input_file1 [input_file2] output_file.vdjca
```

V, J and C aligners parameters

MiXCR uses same types of aligners to align V, J and C genes (`KAligner` from MiLib; the idea of `KAligner` is inspired by this article). These parameters are placed in `parameters` subgroup and can be overridden using

e.g. `-OjParameters.parameters.mapperKValue=7`. The following parameters for V, J and C aligners are available:

Parameter	Default V value	Default J value	Default C value	Description
`mapperKValue`	5	5	5	Length of seeds used in aligner.
`floatingLeftBound`	true	true	false	Specifies whether left bound of alignment is fixed or float: if `floatingLeftBound` set to false, the left bound of either target or query will be aligned. Default values are suitable in most cases.
`floatingRightBound`	true	true	false	Specifies whether right bound of alignment is fixed or float: if `floatingRightBound` set to false, the right bound of either target or query will be aligned. Default values are suitable in most cases. If your target molecules have no primer sequences in J Region (e.g. library was amplified using primer to the C region) you can change value of this parameter for J gene to `false` to increase J gene identification accuracy and overall specificity of alignments.
`minAlignmentLength`	15	15	15	Minimal length of aligned region.
`maxAdjacentIndels`	2	2	2	Maximum number of indels between two seeds.
`absoluteMinScore`	40.0	40.0	40.0	Minimal score of alignment: alignments with smaller score will be dropped.
`relativeMinScore`	0.87	0.87	0.87	Minimal relative score of alignments: if alignment score is smaller than `relativeMinScore * maxScore`, where `maxScore` is the best score among all alignments for particular gene type (V, J or C) and input sequence, it will be dropped.
`maxHits`	7	7	7	Maximal number of hits: if input sequence align with more than `maxHits` queries, only top `maxHits` hits will be kept.

These parameters can be overridden like in the following example:

```
mixcr align -OvParameters.parameters.minAlignmentLength=30 \
            -OjParameters.parameters.relativeMinScore=0.7 \
            input_file1 [input_file2] output_file.vdjca
```

Scoring used in aligners is specified by `scoring` subgroup of parameters. It contains the following parameters:

Parameter	Default value	Description
`subsMatrix`	**`simple(match = 5,`** `mismatch = -9)`	Substitution matrix. Available types: • `simple` — a matrix with diagonal elements equal to `match` and other elements equal to `mismatch` • `raw` — a complete set of 16 matrix elements should be specified; for example: `raw(5,-9,-9,-9,-9, 5,-9,-9,-9,-9,5,-9, -9,-9,-9,5)` (*equivalent to the default value*)
`gapPenalty`	`-12`	Penalty for gap.

Scoring parameters can be overridden in the following way:

```
mixcr align -OvParameters.parameters.scoring.gapPenalty=-20 input_file1 [input_file2]␣
↪output_file.vdjca
```

```
mixcr align -OvParameters.parameters.scoring.subsMatrix=simple(match=4,mismatch=-11) \
            input_file1 [input_file2] output_file.vdjca
```

D aligner parameters

The following parameters can be overridden for D aligner:

Parameter	Default value	Description
`absoluteMinScore`	30.0	Minimal score of alignment: alignments with smaller scores will be dropped.
`relativeMinScore`	0.85	Minimal relative score of alignment: if alignment score is smaller than `relativeMinScore * maxScore`, where `maxScore` is the best score among all alignments for particular sequence, it will be dropped.
`maxHits`	3	Maximal number of hits: if input sequence align with more than `maxHits` queries, only top `maxHits` hits will be kept.

One can override these parameters like in the following example:

```
mixcr align -OdParameters.absoluteMinScore=10 input_file1 [input_file2] output_file.
↪vdjca
```

Scoring parameters for D aligner are the following:

Parameter	Default value	Description
`type`	`affine`	Type of scoring. Possible values: `affine`, `linear`.
`subsMatrix`	**`simple(match = 5,`** `mismatch = -9)`	Substitution matrix. Available types: • `simple` — a matrix with diagonal elements equal to `match` and other elements equal to `mismatch` • `raw` — a complete set of 16 matrix elements should be specified; for example: `raw(5,-9,-9,-9,-9, 5,-9,-9,-9,-9,5,-9, -9,-9,-9,5)` (*equivalent to the default value*)
`gapOpenPenalty`	`-10`	Penalty for gap opening.
`gapExtensionPenalty`	`-1`	Penalty for gap extension.

These parameters can be overridden in the following way:

```
mixcr align -OdParameters.scoring.gapExtensionPenalty=-5 input_file1 [input_file2]␣
↪output_file.vdjca
```

CHAPTER 4

Assemble clones

The `assemble` command builds a set of clones using alignments obtained with *align* command in order to extract specific gene regions (e.g. CDR3). The syntax of `assemble` is the following:

```
mixcr assemble [options] alignments.vdjca output.clns
```

The following flowchart shows the pipeline of `assemble`:

This pipeline consists of the following steps:

1. The assembler sequentially processes records (aligned reads) from input `.vdjca` file produced by *align*. On the first step, assembler tries to extract gene feature sequences from aligned reads (called *clonal sequence*) specified by `assemblingFeatures` parameter (`CDR3` by default); the clonotypes are assembled with respect to *clonal sequence*. If aligned read does not contain clonal sequence (e.g. `CDR3` region), it will be dropped.
2. If clonal sequence contains at least one nucleotide with low quality (less than `badQualityThreshold` parameter value), then this record will be deferred for further processing by *mapping procedure*. If fraction of low quality nucleotides in deferred record is greater than `maxBadPointsPercent` parameter value, then this record will be finally dropped. Records with clonal sequence containing only good quality nucleotides are used to build core clonotypes by grouping records by equality of clonal sequences (e.g. CDR3). The sequence quality of the resulting core clonotype will be equal to the total of qualities of the assembled reads. Each core clonotype has two main properties: clonal sequence and `count` — a number of records aggregated by this clonotype.
3. After the core clonotypes are built, MiXCR runs *mapping procedure* that processes records deferred on the previous step. *Mapping* is aimed on rescuing of quantitative information from low quality reads. For this, each deferred record is mapped onto already assembled clonotypes: if there is a fuzzy match, then this record will be aggregated by the corresponding clonotype; in case of several matched clonotypes, a single one will be randomly chosen with weights equal to clonotype counts. If no matches found, the record will be finally dropped.
4. After clonotypes are assembled by initial assembler and mapper, MiXCR proceeds to *clustering*. The clustering algorithm tries to find fuzzy matches between clonotypes and organize matched clonotypes in hierarchical tree (*cluster*), where each child layer is highly similar to its parent but has significantly smaller `count`. Thus, clonotypes with small counts will be attached to highly similar "parent" clonotypes with significantly greater count. The typical cluster looks as follows:

After all clusters are built, only their heads are considered as final clones. The maximal depths of cluster, fuzzy matching criteria, relative counts of parent/childs and other parameters can be customized using `clusteringStrategy` parameters described below.

5. The final step is to align clonal sequences to reference V,D,J and C genes. Since the `assemblingFeatures` are different from those used in `align`, it is necessary to rebuild alignments for clonal sequences. This alignments are built by more accurate aligner (since all hits are known in advance); thus, better alignments will be built for each clonal sequence.
6. The result is written to the binary output file (`.clns`) with a comprehensive information about clones.

Command line parameters

The command line options of `assemble` are the following:

Option	Default value	Description
`-h, --help`		Print help message.
`-r {file} --report ...`		Report file name. If this option is not specified, no report file be produced.
`-t {numberOfProcessors} --threads ...`	number of available CPU cores	Number of processing threads.
`-i {indexFile} --index ...`		Specify file which will store information about particular reads aggreagated by each clone (mapping readId -> cloneId).
`-Oparameter=value`		Overrides default value of assembler `parameter` (see next subsection).

All parameters are optional.

Assembler parameters

MiXCR uses a wide range of parameters that controls assembler behaviour. There are some global parameters and parameters organized in groups for each stage of assembling: `cloneClusteringParameters` and `cloneFactoryParameters`. Each group of parameters may contain further subgroups of parameters etc. In order to override some parameter value one can use `-O` followed by fully qualified parameter name and parameter value (e.g. `-Ogroup1.group2.parameter=value`).

One of the key MiXCR features is ability to assemble clonotypes by sequence of custom *gene region* (e.g. `FR3+CDR3`); target clonal sequence can even be disjoint. This region can be specified by `assemblingFeatures` parameter, as in the following example:

```
mixcr assemble -OassemblingFeatures="[V5UTR+L1+L2+FR1,FR3+CDR3]" alignments.vdjca␣
↪output.clns
```

(**note**: `assemblingFeatures` must cover `CDR3`).

Other global parameters are:

Parameter	Default value	Description
`minimalClonalSequenceLength`	12	Minimal length of clonal sequence
`badQualityThreshold`	20	Minimal value of sequencing quality score: nucleotides with lower quality are considered as "bad". If sequencing read contains at least one "bad" nucleotide within the target gene region, it will be deferred at initial assembling stage, for further processing by mapper.
`maxBadPointsPercent`	0.7	Maximal allowed fraction of "bad" points in sequence: if sequence contains more than `maxBadPointsPercent` "bad" nucleotides, it will be completely dropped and will not be used for further processing by mapper. Sequences with the allowed percent of "bad" points will be mapped to the assembled core clonotypes. Set `-OmaxBadPointsPercent=0` in order to completely drop all sequences that contain at least one "bad" nucleotide.
`qualityAggregationType`	`Max`	Algorithm used for aggregation of total clonal sequence quality during assembling of sequencing reads. Possible values: `Max` (maximal quality across all reads for each position), `Min` (minimal quality across all reads for each position), `Average` (average quality across all reads for each position), `MiniMax` (all letters has the same quality which is the maximum of minimal quality of clonal sequence in each read).
`minimalQuality`	0	Minimal allowed quality of each nucleotide of assembled clone. If at least one nucleotide in the assembled clone has quality lower than `minimalQuality`, this clone will be dropped (remember that qualities of reads are aggregated according to selected aggregation strategy during core clonotypes assembly; see `qualityAggregationType`).
`addReadsCountOnClustering`	`false`	Aggregate cluster counts when assembling final clones: if `addReadsCountOnClustering` is `true`, then all children clone counts will be added to the head clone; thus head clone count will be a total of its initial count and counts of all its children. Refers to further clustering strategy (see below). Does not refer to mapping of low quality sequencing reads described above.

One can override these parameters in the following way:

```
mixcr assemble -ObadQualityThreshold=10 alignments.vdjca output.clns
```

In order to prevent mapping of low quality reads (filter them off) one can set `maxBadPointsPercent` to zero:

```
mixcr assemble -OmaxBadPointsPercent=0 alignments.vdjca output.clns
```

Separation of clones with same CDR3 (clonal sequence) but different V/J/C genes

Since v1.8 MiXCR can separates clones with equal clonal sequence and different V, J and C (e.g. do distinguish clones with different IG isotype) genes.

To make analysis more robust to sequencing errors there is an additional clustering step to shrink artificial diversity generated by this separation mechanism.

The following criteria are used on this pre-clusterization step: more abondant clone (`clone1`) absorbs smaller clone (`clone2`) if `clone2.count < clone1.count * maximalPreClusteringRatio` (`cloneX.count` denotes number of reads in corresponding clone)and `clone2` contain top V/J/C gene from `clone1` in it's corresponding gene list.

The following parameter control separation behaviour and pre-clusterization:

Parameter	Default value	Description
maximalPreClusteringRatio	1.0	See conditions for clustering above for more inforamtion.
separateByV	false	If false clones with equal clonal sequence but different V gene will be merged into single clone.
separateByJ	false	If false clones with equal clonal sequence but different J gene will be merged into single clone.
separateByC	false	If false clones with equal clonal sequence but different C gene will be merged into single clone.

Example, in order to separate IG clones by isotypes use the following options:

```
mixcr assemble -OseparateByC=true alignments.vdjca output.clns
```

Clustering strategy

Parameters that control clustering procedure are placed in cloneClusteringParameters parameters group which determines the rules for the frequency-based correction of PCR and sequencing errors:

Parameter	Default value	Description
searchDepth	2	Maximum number of cluster layers (not including head).
allowedMutationsInNRegions	1	Maximum allowed number of mutations in N regions (non-template nucleotides in VD, DJ or VJ junctions): if two fuzzy matched clonal sequences will contain more than allowedMutationsInNRegions mismatches in N-regions, they will not be clustered together (one cannot be a direct child of another).
searchParameters	twoMismatchesOrIndels	Parameters that control fuzzy match criteria between clones in adjacent layers. Available predefined values: oneMismatch, oneIndel, oneMismatchOrIndel, twoMismatches, twoIndels, twoMismatchesOrIndels, ... , fourMismatchesOrIndels. By default, twoMismatchesOrIndels allows two mismatches or indels (not more than two errors of both types) between two adjacent clones (parent and direct child).
clusteringFilter.specificMutationProbability	1E-3	Probability of a single nucleotide mutation in clonal sequence which has non-hypermutation origin (i.e. PCR or sequencing error). This parameter controls relative counts between two clones in adjacent layers: a smaller clone can be attached to a larger one if its count smaller than count of parent multiplied by (clonalSequenceLength * specificMutationProbability) ^ numberOfMutations.

One can override these parameters in the following way:

```
mixcr assemble -OcloneClusteringParameters.searchParameters=oneMismatchOrIndel␣
↪alignments.vdjca output.clns
```

In order to turn off clustering one should use the following parameters:

```
mixcr assemble -OcloneClusteringParameters=null alignments.vdjca output.clns
```

CHAPTER 5

Export

In order to export result of alignment or clones from binary file (`.vdjca` or `.clns`) to a human-readable text file one can use `exportAlignments` and `exportClones` commands respectively. The syntax for these commands is:

```
mixcr exportAlignments [options] alignments.vdjca alignments.txt
```

```
mixcr exportClones [options] clones.clns clones.txt
```

The resulting tab-delimited text file will contain columns with different types of information. If no options specified, the default set of columns, which is sufficient in most cases, will be exported. The possible columns are (see below for details): aligned sequences, qualities, all or just best hit for V, D, J and C genes, corresponding alignemtns, nucleotide and amino acid sequences of gene region present in sequence etc. In case of clones, the additional columns are: clone count, clone fraction etc.

One can customize the list of fields that will be exported by passing parameters to export commands. For example, in order to export just clone count, best hits for V and J genes with corresponding alignments and `CDR3` amino acid sequence, one can do:

```
mixcr exportClones -count -vHit -jHit -vAlignment -jAlignment -aaFeature CDR3 clones.
↪clns clones.txt
```

The columns in the resulting file will be exported in the exact same order as parameters in the command line. The list of available fields will be reviewed in the next subsections. For convenience, MiXCR provides two predefined sets of fields for exporting: `min` (will export minimal required information about clones or alignments) and `full` (used by default); one can use these sets by specifying `--preset` option:

```
mixcr exportClones --preset min clones.clns clones.txt
```

One can add additional columns to preset in the following way:

```
mixcr exportClones --preset min -qFeature CDR2 clones.clns clones.txt
```

One can also put all export fields in the file like:

```
-vHits
-dHits
-feature CDR3
...
```

and pass this file to export command:

```
mixcr exportClones --preset-file myFields.txt clones.clns clones.txt
```

Command line parameters

The list of command line parameters for both `exportAlignments` and `exportClones` is the following:

Option	Description
`-h, --help`	print help message
`-f, --fields`	list available fields that can be exported
`-p, --preset`	select predefined set of fields to export (`full` or `min`)
`-pf, --preset-file`	load file with a list of fields to export
`-lf, --list-fields`	list availabel fields that can be exported
`-s, --no-spaces`	output short versions of column headers which facilitates analysis with Pandas, R/DataFrames or other data tables processing library

The line parameters are only for `exportClones`:

`-c, --chains`	Limit output to specific locus (e.g. TRA or IGH). Clone fractions will be recalculated accordingly.
`-o, --filter-out-of-frames`	Exclude out of frames (fractions will be recalculated)
`-t, --filter-stops`	Exclude sequences containing stop codons (fractions will be recalculated)
`-m, --minimal-clone-count`	Filter clones by minimal read count.
`-q, --minimal-clone-fraction`	Filter clones by minimal clone fraction.

Available fields

The following fields can be exported both for alignments and clones:

Field name	Description
`-targets`	Number of targets
`-vHit`	Best V hit
`-dHit`	Best D hit
`-jHit`	Best J hit
`-cHit`	Best C hit
`-vGene`	Best V hit gene name (e.g. TR
`-dGene`	Best D hit gene name (e.g. TR
`-jGene`	Best J hit gene name (e.g. TRE
`-cGene`	Best C hit gene name (e.g. TR

Field name	Description
-vFamily	Best V hit family name (e.g. T
-dFamily	Best D hit family name (e.g. T
-jFamily	Best J hit family name (e.g. TR
-cFamily	Best C hit family name (e.g. T
-vHitScore	Score for best V hit
-dHitScore	Score for best D hit
-jHitScore	Score for best J hit
-cHitScore	Score for best C hit
-vHitsWithScore	All V hits with score
-dHitsWithScore	All D hits with score
-jHitsWithScore	All J hits with score
-cHitsWithScore	All C hits with score
-vHits	All V hits
-dHits	All D hits
-jHits	All J hits
-cHits	All C hits
-vGenes	All V gene names (e.g. TRBV
-dGenes	All D gene names (e.g. TRBV
-jGenes	All J gene names (e.g. TRBV1
-cGenes	All C gene names (e.g. TRBV
-vFamilies	All V gene family anmes (e.g.
-dFamilies	All D gene family anmes (e.g.
-jFamilies	All J gene family anmes (e.g. T
-cFamilies	All C gene family anmes (e.g.
-vAlignment	Best V alignment
-dAlignment	Best D alignment
-jAlignment	Best J alignment
-cAlignment	Best C alignment
-vAlignments	All V alignments
-dAlignments	All D alignments
-jAlignments	All J alignments
-cAlignments	All C alignments
-nFeature <gene_feature>	Nucleotide sequence of specifi
-qFeature <gene_feature>	Quality string of specified gene
-aaFeature <gene_feature>	Amino acid sequence of specif
-minFeatureQuality <gene_feature>	Minimal quality of specified ge
-avrgFeatureQuality <gene_feature>	Average quality of specified ge
-lengthOf <gene_feature>	S length of specified gene featu
-nMutations <gene_feature>	Extract nucleotide mutations fo
-nMutationsRelative <gene_feature> <relative_to_gene_feature>	Extract nucleotide mutations fo
-aaMutations <gene_feature>	Extract amino acid mutations f
-aaMutationsRelative <gene_feature> <relative_to_gene_feature>	Extract amino acid mutations f
-mutationsDetailed <gene_feature>	Detailed list of nucleotide and
-mutationsDetailedRelative <gene_feature> <relative_to_gene_feature>	Detailed list of nucleotide and
-positionOf <reference_point>	S position of specified referenc
-defaultAnchorPoints	Outputs a list of default referen
-vIdentityPercents	V alignment identity percents
-dIdentityPercents	D alignment identity percents
-jIdentityPercents	J alignment identity percents

Field name	Description
-cIdentityPercents	C alignment identity percents
-vBestIdentityPercent	Vbest alignment identity perce
-dBestIdentityPercent	Dbest alignment identity perce
-jBestIdentityPercent	Jbest alignment identity percer
-cBestIdentityPercent	Cbest alignment identity perce

The following fields are specific for alignments:

Field name	Description
-readId	Id of read corresponding to alignment
-sequence	Aligned sequence (initial read), or 2 sequences in case of paired-end reads
-quality	Initial read quality, or 2 qualities in case of paired-end reads
-descrR1	Description line from initial .fasta or .fastq file of the first read (only available if –save-description was used in align command)
-descrR2	Description line from initial .fasta or .fastq file of the second read (only available if –save-description was used in align command)
-cloneId <index_file>	To which clone alignment was attached.
-cloneIdWithMappingType <index_file>	To which clone alignment was attached with additional info on mapping type.

The following fields are specific for clones:

Field name	Description
-cloneId	Unique clone identifier
-count	Clone count
-fraction	Clone fraction
-sequence	Aligned sequence (initial read), or 2 sequences in case of paired-end reads
-quality	Initial read quality, or 2 qualities in case of paired-end reads
-readIds <index_file>	Read IDs aggregated by clone.

Default anchor point positions

Positions of anchor poins produced by -defaultAnchorPoints option are outputted as a colon separated list. If anchor point is not covered by target sequence nothing is printed for it, but flanking colon symbols are preserved to maintain positions in array. See example:

```
:::::::::108:117:125:152:186:213:243:244:
```

If there are several target sequences (e.g. paired-end reads or multi-part clonal sequnce), the array is outputted for each target sequence. In this case arrays are sepparated by comma:

```
2:61:107:107:118::::::::::::,:::::::::103:112:120:147:181:208:238:239:
```

Even if there are no anchor points in one of the parts:

```
::::::::::::::::,:::::::::108:117:125:152:186:213:243:244:
```

The following table shows the correspondance between anchor point and positions in default anchor point array:

Anchors point	Zero-based position	One-based position
V5UTRBeginTrimmed	0	1
V5UTREnd / L1Begin	1	2
L1End / VIntronBegin	2	3
VIntronEnd / L2Begin	3	4
L2End / FR1Begin	4	5
FR1End / CDR1Begin	5	6
CDR1End / FR2Begin	6	7
FR2End / CDR2Begin	7	8
CDR2End / FR3Begin	8	9
FR3End / CDR3Begin	9	10
Number of 3' V deletions (negative value), or length of 3' V P-segment (positive value)	10	11
VEndTrimmed, next position after last aligned nucleotide of V gene	11	12
DBeginTrimmed, position of first aligned nucleotide of D gene	12	13
Number of 5' D deletions (negative value), or length of 5' D P-segment (positive value)	13	14
Number of 3' D deletions (negative value), or length of 3' D P-segment (positive value)	14	15
DEndTrimmed, next position after last aligned nucleotide of D gene	15	16
JBeginTrimmed, position of first aligned nucleotide of J gene	16	17
Number of 3' J deletions (negative value), or length of 3' J P-segment (positive value)	17	18
CDR3End / FR4Begin	18	19
FR4End	19	20
CBegin	20	21
CExon1End	21	22

The following regular expressions can be used to parse content of this field in Python:

- for length analysis, or analysis of raw alignments:

```
^(?P<V5UTRBegin>-?[0-9]*):(?P<L1Begin>-?[0-9]*):(?P<VIntronBegin>-?[0-
↪9]*):(?P<L2Begin>-?[0-9]*):(?P<FR1Begin>-?[0-9]*):(?P<CDR1Begin>-?[0-
↪9]*):(?P<FR2Begin>-?[0-9]*):(?P<CDR2Begin>-?[0-9]*):(?P<FR3Begin>-?[0-
↪9]*):(?P<CDR3Begin>-?[0-9]*):(?P<V3Deletion>-?[0-9]*):(?P<VEnd>-?[0-
↪9]*):(?P<DBegin>-?[0-9]*):(?P<D5Deletion>-?[0-9]*):(?P<D3Deletion>-?[0-
↪9]*):(?P<DEnd>-?[0-9]*):(?P<JBegin>-?[0-9]*):(?P<J5Deletion>-?[0-
↪9]*):(?P<CDR3End>-?[0-9]*):(?P<CBegin>-?[0-9]*):(?P<CExon1End>-?[0-9]*)
↪$
```

snipped for Pandas:

```
import pandas as pd
data = pd.read_table("exported.txt", low_memory=False)
anchorPointsRegex="^(?P<V5UTRBegin>-?[0-9]*):(?P<L1Begin>-?[0-9]*):(?P
↪<VIntronBegin>-?[0-9]*):(?P<L2Begin>-?[0-9]*):(?P<FR1Begin>-?[0-9]*):(?
↪P<CDR1Begin>-?[0-9]*):(?P<FR2Begin>-?[0-9]*):(?P<CDR2Begin>-?[0-9]*):(?
↪P<FR3Begin>-?[0-9]*):(?P<CDR3Begin>-?[0-9]*):(?P<V3Deletion>-?[0-
↪9]*):(?P<VEnd>-?[0-9]*):(?P<DBegin>-?[0-9]*):(?P<D5Deletion>-?[0-
↪9]*):(?P<D3Deletion>-?[0-9]*):(?P<DEnd>-?[0-9]*):(?P<JBegin>-?[0-
↪9]*):(?P<J5Deletion>-?[0-9]*):(?P<CDR3End>-?[0-9]*):(?P<CBegin>-?[0-
↪9]*):(?P<CExon1End>-?[0-9]*)$"
data = pd.concat([data, d.refPoints.str.extract(anchorPointsRegex,␣
↪expand=True).apply(pd.to_numeric)], axis=1)
```

- simplified regular expression with the smaller number of fields, can be used for analysis of CDR3-assembled clonotypes:

```
^(?:-?[0-9]*:){8}(?:-?[0-9]*):(?P<CDR3Begin>-?[0-9]*):(?P<V3Deletion>-?
↪[0-9]*):(?P<VEnd>-?[0-9]*):(?P<DBegin>-?[0-9]*):(?P<D5Deletion>-?[0-
↪9]*):(?P<D3Deletion>-?[0-9]*):(?P<DEnd>-?[0-9]*):(?P<JBegin>-?[0-
↪9]*):(?P<J5Deletion>-?[0-9]*):(?P<CDR3End>-?[0-9]*):(?:-?[0-9]*:){2}(?
↪:-?[0-9]*)$
```

snipped for Pandas:

```
import pandas as pd
data = pd.read_table("exported.txt", low_memory=False)
anchorPointsRegex="^^(?:-?[0-9]*:){8}(?:-?[0-9]*):(?P<CDR3Begin>-?[0-
↪9]*):(?P<V3Deletion>-?[0-9]*):(?P<VEnd>-?[0-9]*):(?P<DBegin>-?[0-
↪9]*):(?P<D5Deletion>-?[0-9]*):(?P<D3Deletion>-?[0-9]*):(?P<DEnd>-?[0-
↪9]*):(?P<JBegin>-?[0-9]*):(?P<J5Deletion>-?[0-9]*):(?P<CDR3End>-?[0-
↪9]*):(?:-?[0-9]*:){2}(?:-?[0-9]*)$"
data = pd.concat([data, d.refPoints.str.extract(anchorPointsRegex,␣
↪expand=True).apply(pd.to_numeric)], axis=1)
```

Examples

Export only best V, D, J hits and best V hit alignment from `.vdjca` file:

```
mixcr exportAlignments -vHit -dHit -jHit -vAlignment input.vdjca test.txt
```

Best V hit	Best D hit	Best J hit	Best V alignment
IGHV4-34*00		IGHJ4*00	262|452|453|47|237|SC268GSC271ASC275G|956.1,58|303|450|56|301|SG72TSA73CSG136TSA144CSA158CSG171T|331.0|
IGHV2-23*00	IGHD2*1	IGHJ6*00	262|452|453|47|237|SC268GSC271ASC275G|956.1,58|303|450|56|301|SG72TSA73CSG136TSA144CSA158CSG171T|331.0|

The syntax of alignment is described in *appendix*.

Exporting well formatted alignments for manual inspection

MiXCR allows to export resulting alignments after *align* step as a pretty formatted text for manual analysis of produced alignments and structure of library to facilitate optimization of analysis parameters and libraray preparation protocol. To export pretty formatted alignments use `exportAlignmentsPretty` command:

```
mixcr exportAlignmentsPretty --skip 1000 --limit 10 input.vdjca test.txt
```

this will export 10 results after skipping first 1000 records and place result into `test.txt` file. Skipping of first records is often useful because first sequences in fastq file may have lower quality then average reads, so first resulsts are not representative. It is possible to omit last paramenter with output file name to print result directly to standard output stream (to console), like this:

```
mixcr exportAlignmentsPretty --skip 1000 --limit 10 input.vdjca
```

Here is a summary of command line options:

Option	Description
`-h, --help`	print help message
`-n, --limit`	limit number of alignments; no more than provided number of results will be outputted
`-s, --skip`	number of results to skip
`-t, --top`	output only top hits for V, D, J nad C genes
`--cdr3-contains`	output only those alignemnts which CDR3 contains specified nucleotides (e.g. `--cdr3-contains TTCAGAGGAGC`)
`--read-contains`	output only those alignemnts for which corresonding reads contain specified nucleotides e.g. `--read-contains ATGCTTGCGCGCT`)
`--verbose`	use more verbose format for alignments (see below for example)

Results produced by this command has the following structure:

Using of `--verbose` option will produce alignments in s slightly different format:

Exporting reads aggregated by clones

MiXCR allows to preserve mapping between initial reads and final clonotypes. There are several options how to access this information.

In any way, first one need to specify additonal option `--index` for the *assemble* command:

```
mixcr assemble --index index_file alignments.vdjca output.clns
```

This will tell MiXCR to store mapping in the file `index_file` (actually two files will be created: `index_file` and `index_file.p` both of which are used to store the index; in further options one should specify only `index_file` without `.p` extension and MiXCR will automatically read both required files). Now one can use `index_file` in order to access this information. For example using `-cloneId` option for `exportAlignments` command:

```
mixcr exportAlignments -p min -cloneId index_file alignments.vdjca alignments.txt
```

will print additional column with id of the clone which contains corresponding alignment:

Best V hit	Best D hit	...	CloneId
IGHV4-34*00		...	321
IGHV2-23*00	IGHD2*21	...	
IGHV4-34*00	IGHD2*21	...	22143
...	...	...	...

For more information one can export mapping type as well:

```
mixcr exportAlignments -p min -cloneIdWithMappingType index_file alignments.vdjca␣
↪alignments.txt
```

which will give something like:

Best V hit	Best D hit	...	Clone mapping
IGHV4-34*00		...	321:core
IGHV2-23*00	IGHD2*21	...	dropped
IGHV4-34*00	IGHD2*21	...	22143:clustered
IGHV4-34*00	IGHD2*21	...	23:mapped
...	...	...	...

One can also export all read IDs that were aggregated by eah clone. For this one can use `-readIds` export options for `exportClones` action:

```
mixcr exportClones -c IGH -p min -readIds index_file clones.clns clones.txt
```

This will add a column with full enumeration of all reads that were absorbed by particular clone:

Clone ID	Clone count	Best V hit	...	Reads
0	7213	IGHV4-34*00	...	56,74,92,96,101,119,169,183...
1	2951	IGHV2-23*00	...	46,145,194,226,382,451,464...
2	2269	IGHV4-34*00	...	58,85,90,103,113,116,122,123...
3	124	IGHV4-34*00	...	240,376,496,617,715,783,813...
...		...	...	...

Note, that resulting txt file may be very huge since all read numbers that were successfully assembled will be printed.

Finally, one can export reads aggregated by each clone into separate `.fastq` file. For that one need first to specify additional `-g` option for *align* command:

```
mixcr align -g input.fastq alignments.vdjca.gz
```

With this option MiXCR will store original reads in the `.vdjca` file. Then one can export reads corresponding for particular clone with `exportReadsForClones` command. For example, export all reads that were assembled into the first clone (clone with cloneId = 0):

```
mixcr exportReadsForClones index_file alignments.vdjca.gz 0 reads.fastq.gz
```

This will create file `reads_clns0.fastq.gz` (or two files `reads_clns0_R1.fastq.gz` and `reads_clns0_R2.fastq.gz` if the original data were paired) with all reads that were aggregated by the first clone. One can export reads for several clones at a time:

```
mixcr exportReadsForClones index_file alignments.vdjca.gz 0 1 2 33 54 reads.fastq.gz
```

This will create several files (`reads_clns0.fastq.gz`, `reads_clns1.fastq.gz` etc.) for each clone with cloneId equal to 0, 1, 2, 33 and 54 respectively.

CHAPTER 6

Processing RNA-seq data

Overview

Analysis method and quirks described here will also be useful for users who want to extract TCR or Ig repertoire from sequencing data of any other type of non-enriched or randomly shred cDNA / gDNA library.

There are two main challenges of repertoire extraction from non-enriched and randomly-shred libraries:

- **Extraction and alignment of fragments of target molecules.** This procedure must be *sensitive* enough to detect and align sequences with very small parts of V or J genes, but at the same time must be very *selective* not to align non-target sequences homologous to TCR or Ig. Alignment of such sequences and treating them as TCRs or IGs bring a risk of introducing reproducible false-positive clonotypes into resulting clonesets, and may, in turn, lead to detection of false intersections between unlinked repertoires.

  MiXCR has a special set of alignment parameters (`-p rna-seq`), which was specifically optimized, and automatically and manually checked on tens of different datasets to give the best possible sensitivity keeping zero false-positive rate.

- **Assembly of overlapping fragmented sequencing reads into long-enough CDR3 containing contigs.** In contrast to sequencing reads from targeted IG or TCR libraries with very determined CDR3 position, reads from randomly shred libraries may cover only a part of CDR3. This fact is especially true for short-read data (like very common 50+50 RNA-Seq), where most part of target sequences only partially cover CDR3. In order to efficiently extract repertoire from such data one have to reconstruct initial CDR3s from fragments scattered all over the initial sequencing dataset. The main challenge of this procedure is, again, the possibility to introduce false-positive clones, namely to perform an overlap between two sequences from different clones. This false positives are not so dangerous as those described in the previous paragraph, but still may introduce certain biases. The problem is that it is very easy to make such false-overlaps as TCR or IG sequences consist mainly from conservative V, D and J regions. So overlapping must be done very carefully, taking into account the positions of all conserved regions.

  MiXCR has a special action to perform such an assembly of reads, partially covering CDR3 - `assemblePartial`. Basically it performs an overlap of already aligned reads from **.vdjca* file, realigns resulting contig, and checks if initial overlap has covered enough part of a non-template N region. Default thresholds in this procedure were optimized to assemble as many contigs as possible while producing zero false overlaps (no false overlaps were detected in all of the benchmarks we have performed).

In case of short reads input, even after `assemblePartial` many contigs/reads still only partially cover CDR3. A substantial fraction of such contigs needs only several nucleotides on the 5' or the 3' end to fill up the sequence up to a complete CDR3. These sequence parts can be taken from the germline, if corresponding V or J gene for the contig is uniquely determined (e.g. from second mate of a read pair). Such procedure is not safe for IGs, because of hypermutations, but for TCRs which have relatively conservative sequence near conserved `Cys` and `Phe`/`Trp`, it can reconstruct additional clonotypes with relatively small chance to introduce false ones. Described procedure is implemented in the action *extendAlignments*, by default it acts only on TCR sequences.

Analysis pipeline

MiXCR has all of the steps required to efficiently extract repertoire data from RNA-Seq and similar sequencing datasets, starting from raw `fastq(.gz)` files to final list of clonotypes for each immunological chain (`TRB`, `IGH`, etc..).

All default values for analysis parameters were carefully optimized, and should be suitable for most of the use-cases.

Prerequisite

There are only two things you must tell MiXCR for a successfull analysis. Both on the first `align` step.

1. **Species.** Using `-s ...` parameter. See *here*.
2. **Data source origin**. Genomic or transcriptomic. This affects which part of reference V gene seqeucnes will be used for alignment, with or without intron. By default transcriptomic source is assumed, so no additional parameters have to be specified for an analysis of RNA-Seq data. If your data has a genomic DNA origin add the following option to the `align` command:

```
-OvParameters.geneFeatureToAlign=VGeneWithP
```

   This option tells MiXCR to use unspliced reference sequences of V genes for alignments.

Typical analysis workflow

1. Align sequencing reads against reference V, D, J and C genes.

```
mixcr align -p rna-seq -s hsa -OallowPartialAlignments=true data_R1.fastq.gz␣
↪data_R2.fastq.gz alignments.vdjca
```

   For single-end data simply specify single input file:

```
mixcr align -p rna-seq -s hsa -OallowPartialAlignments=true data.fastq.gz␣
↪alignments.vdjca
```

   If your data has a genomic origin add `-OvParameters.geneFeatureToAlign=VGeneWithP` option.

   `-OallowPartialAlignments=true` option is needed to prevent MiXCR from filtering out partial alignments, that don't fully cover CDR3 (the default behaviour while processing targeted RepSeq data). MiXCR will try to assemble contigs using those alignments and reconstruct their full CDR3 sequence on the next step.

2. Perform two rounds of contig assembly (please see *here* for available parameters).

```
mixcr assemblePartial alignments.vdjca alignments_rescued_1.vdjca
mixcr assemblePartial alignments_rescued_1.vdjca alignments_rescued_2.vdjca
```

3. (optional) Perform extension of incomplete TCR CDR3s with uniquely determined V and J genes using germline sequences. As described in the *last paragraph of introduction*

```
mixcr extendAlignments alignments_rescued_2.vdjca alignments_rescued_2_
↪extended.vdjca
```

4. Assemble (see *here* for details) clonotypes

```
mixcr assemble alignments_rescued_2_extended.vdjca clones.clns
```

5. Export (see *here* for details) all clonotypes:

```
mixcr exportClones clones.clns clones.txt
```

or clonotypes for a specific immunological chain:

```
mixcr exportClones -c TRB clones.clns clones.TRB.txt
mixcr exportClones -c IGH clones.clns clones.IGH.txt
...
```

The resulting `*.txt` files will contain clonotypes along with comprehansive biological information like V, D, J and C genes, clone abundances, etc...

`assemblePartial` action

The following options are available for `assemblePartial`:

Parameter	Default value	Description
`kValue`	`12`	Length of k-mer taken from VJ junction region and used for searching potentially overlapping sequences.
`kOffset`	`-7`	Offset taken from `VEndTrimmed`/`JBeginTrimmed`.
`minimalAssembleOverlap`	`12`	Minimal length of the overlapped VJ region: two sequences can be potentially merged only if they have at least `minimalAssembleOverlap`-wide overlap in the VJJunction region. No mismatches are allowed in the overlapped region.
`minimalNOverlap`	`5`	Minimal number of non-template nucleotides (N region) that overlap region must cover to accept the overlap.

The above parameters can be specified in e.g. the following way:

```
mixcr assemblePartial -OminimalAssembleOverlap=10 alignments.vdjca alignmentsRescued.
↪vdjca
```

CHAPTER 7

Using external libraries for alignment

Tip: MiXCR utilases libraries in .json format (see https://github.com/repseqio for details).

Note: In some cases when using an external library mixcr will try to establish connection with NCBI over the internet.

IMGT library

Compiled IMGT library file for MiXCR can be downloaded at https://github.com/repseqio/library-imgt/releases. In order to use the library put the `.json` library file to `~/.mixcr/libraries` folder, to the directory from where mixcr is started or to `libraries/` subfolder of mixcr installation folder.

Tip: Use `mixcr -v` to see what folders mixcr uses to look for library .json file.

```
> mixcr -v

...

Library search path:
- built-in libraries
- /home/username/.
- /home/username/.mixcr/libraries
- /software/mixcr/libraries
```

```
> mixcr align --library imgt input_R1.fastq input_R2.fastq alignments.vdjca

... Building alignments
```

`--library` option specifies the library to use for alignment. If the short name is given (ex.‘‘–library imgt‘‘) mixcr will look for the latest version in the folder. Otherwise, to use one of the old versions give the full name including the version number (ex. `-library imgt.201631-4`)

```
> mixcr assemble alignments.vdjca clones.clns

... Assembling clones

> mixcr exportClones --chains IGH clones.clns clones.txt

... Exporting clones to tab-delimited file
```

CHAPTER 8

KAligner2: New aligner with big gaps support

Danger: This feature is provided for beta testing, and not recommended for production use!

To process data using new aligner, apply special parameter pre-sets as follows:

```
mixcr align -p kaligner2 ....
mixcr assemble ....
....
```

Any other parameters can also be provided along with `-p ...` option.

CHAPTER 9

Gene features and anchor points

There are several immunologically important parts of TCR/BCR gene (**gene features**). For example, such regions are three complementarity determining regions (`CDR1`, `CDR2` and `CDR3`), four framework regions (`FR1`, `FR2`, `FR3` and `FR4`) etc.

The key feature of MiXCR is the possibility to specify:

- regions of reference V, D, J and C genes sequences that are used in *alignment of raw reads*
- regions of sequence to be exported by *exportAlignments*
- regions of sequence to use as clonal sequence in *clone assembly*
- regions of clonal sequences to be exported by *exportClones*

For convenience, in MiXCR these regions can be specified in terms of above mentioned immunological gene features. The illustrated list of predefined gene features can be found below. The set of possible gene regions is not limited by this list:

- boundary points of gene features (called **anchor points**) can be used to specify begin and end of custom gene regions
- gene features can be concatenated (e.g. VTranscript = {V5UTRBegin:L1End}+{L2Begin:VEnd}).
- offsets can be added or subtracted from original positions of **anchor points** to define even more custom gene regions (for more detailed description see *gene feature syntax*)

Naming of gene features is based on IMGT convention described in *Lefranc et al. (2003), Developmental & Comparative Immunology 27.1 (2003): 55-77*.

Germline features

Features defined for germline genes are mainly used in *align* and *export*.

V Gene structure

Additionally to core gene features in V region (like `FR3`) we introduce `VGene`, `VTranscript` and `VRegion` for convenience.

D Gene structure

J Gene structure

Mature TCR/BCR gene features

Features described here (like `CDR3`) cannot not be used for *align*, since they are not defined for germline genes.

V(D)J junction structure

Important difference between rearranged TCR/BCR sequence and germline sequence of its segments lies in the fact that during V(D)J recombination exact cleavage positions at the end of V gene, begin and end of D gene and begin of J gene varies. As a result in most cases actual `VEnd`, `DBegin`, `DEnd` and `JBegin` anchor positions are not covered by alignment:

In order to use actual V, D, J gene boundaries we introduce four additional anchor positions: `VEndTrimmed`, `DBeginTrimmed`, `DEndTrimmed` and `JBeginTrimmed` and several named gene features: `VDJunction`, `DJJunction` and `VJJunction`. On the following picture one can see the structure of V(D)J junction:

If D gene is not found in the sequence or is not present in target locus (e.g. TRA), `DBeginTrimmed` and `DEndTrimmed` anchor points as well as `VDJunction` and `DJJunction` gene features are not defined.

Similar to `...Trimmed` anchor points in V(D)J junction there is a `V5UTRBeginTrimmed` anchor point representing left bound of alignment upstream start codon. This point is required because 5'UTR could have different length from transcript to transcript, and because library of gene segments inside MiXCR does'n have information on exact 5'UTR lengths.

Gene feature syntax

Syntax for gene features is the same everywhere. The best way to explain it is by example:

- to enter any gene feature mentioned above or listed in the next section just use its name: `VTranscript`, `CDR2`, `V5UTR` etc.
- to define a gene feature consisting of several concatenated features use `+`: `V5UTR+L1+L2+VRegion` is equivalent to `VTranscript`
- to create gene feature starting at anchor point `X` and ending at anchor point `Y` use {X:Y} syntax: `{CDR3Begin:CDR3End}` for `CDR3`.

- one can add or subtract offset from original position of anchor point using positive or negative integer value in brackets after anchor point name AnchorPoint(offset): `{CDR3Begin(+3):CDR3End}` for `CDR3` without first three nucleotides (coding conserved cysteine), `{CDR3Begin(-6):CDR3End(+6)}` for `CDR3` with 6 nucleotides downstream its left bound and 6 nucleotides upstream its right bound.
- one can specify offsets for predefined gene feature boundaries using GeneFeatureName(leftOffset, rightOffset) syntax: `CDR3(3,0)`, `CDR3(-6,6)` - equivalents of two examples from previous item
- all syntax constructs can be combined: `{L1Begin(-12):L1End}+L2+VRegion(0,+10)`.

List of predefined gene features

Gene Feature Name	Gene feature decomposition	Document
V5UTRGermline	{UTR5Begin:V5UTREnd}	5'UTR; ger
VTranscript	{UTR5Begin:L1End} + {L2Begin:VEnd}	V5UTR + E
VGene	{UTR5Begin:VEnd}	{V5UTRBe
VTranscriptWithP	{UTR5Begin:L1End} + {L2Begin:VEnd} + {VEnd:VEnd(-20)}	V5UTR + E
VGeneWithP	{UTR5Begin:VEnd} + {VEnd:VEnd(-20)}	{V5UTRBe
VDJTranscript	{UTR5Begin:L1End} + {L2Begin:FR4End}	First two ex
V5UTR	{V5UTRBeginTrimmed:V5UTREnd}	5'UTR in a
L1	{L1Begin:L1End}	Part of lide
VLIntronL	{L1Begin:L2End}	L1 + VInt
Exon1	{L1Begin:L1End}	First exon.
L	{L1Begin:L1End} + {L2Begin:L2End}	Full leader
VTranscriptWithout5UTR	{L1Begin:L1End} + {L2Begin:VEnd}	Exon1 + V
VTranscriptWithout5UTRWithP	{L1Begin:L1End} + {L2Begin:VEnd} + {VEnd:VEnd(-20)}	Exon1 + V
VDJTranscriptWithout5UTR	{L1Begin:L1End} + {L2Begin:FR4End}	First two ex
VIntron	{VIntronBegin:VIntronEnd}	Intron in V
L2	{L2Begin:L2End}	Part of lide
Exon2	{L2Begin:FR4End}	Full second
VExon2	{L2Begin:VEnd}	Second exo
VExon2Trimmed	{L2Begin:VEndTrimmed}	Second exo
VRegion	{FR1Begin:VEnd}	Full V Regi
VRegionWithP	{FR1Begin:VEnd} + {VEnd:VEnd(-20)}	Full V Regi
VRegionTrimmed	{FR1Begin:VEndTrimmed}	Full V Regi
FR1	{FR1Begin:FR1End}	Framework
VDJRegion	{FR1Begin:FR4End}	Full V, D, J
CDR1	{CDR1Begin:CDR1End}	CDR1 (Cor
FR2	{FR2Begin:FR2End}	Framework
CDR2	{CDR2Begin:CDR2End}	CDR2 (Cor
FR3	{FR3Begin:FR3End}	Framework
CDR3	{CDR3Begin:CDR3End}	CDR3 (Cor
VCDR3Part	{CDR3Begin:VEndTrimmed}	Part of V re
GermlineVCDR3Part	{CDR3Begin:VEnd}	Part of V re
ShortCDR3	{CDR3Begin(3):CDR3End(-3)}	CDR3 (Cor
VDJunction	{VEndTrimmed:DBeginTrimmed}	N region be
VJJunction	{VEndTrimmed:JBeginTrimmed}	Region betw
VPSegment	{VEnd:VEndTrimmed}	P-segment

Table 9

Gene Feature Name	Gene feature decomposition	Document
GermlineVPSegment	{VEnd:VEnd(-20)}	P-segment
DRegion	{DBegin:DEnd}	Full D Regi
DLeftPSegment	{DBeginTrimmed:DBegin}	Left P-segn
DCDR3Part	{DBeginTrimmed:DEndTrimmed}	Full D Regi
DJJunction	{DEndTrimmed:JBeginTrimmed}	N region be
DRightPSegment	{DEnd:DEndTrimmed}	Right P-seg
GermlineDPSegment	{DEnd:DBegin}	P-segment
DRegionWithP	{DEnd:DBegin} + {DBegin:DEnd} + {DEnd:DBegin}	Full D Regi
JRegion	{JBegin:FR4End}	Full J Regio
GermlineJCDR3Part	{JBegin:CDR3End}	Part of J reg
GermlineJPSegment	{JBegin(20):JBegin}	P-segment
JRegionWithP	{JBegin(20):JBegin} + {JBegin:FR4End}	Full J Regio
JPSegment	{JBeginTrimmed:JBegin}	P-segment
JRegionTrimmed	{JBeginTrimmed:FR4End}	Full J Regio
JCDR3Part	{JBeginTrimmed:CDR3End}	Part of J reg
FR4	{FR4Begin:FR4End}	Framework
CExon1	{CBegin:CExon1End}	First exon o
CRegion	{CBegin:CEnd}	Full C regio

List of predefined reference points

UTR5Begin	Beginning of IG/TCR transcript
V5UTREnd	End of 5'UTR, beginning of IG/TCR CDS as listed in database
V5UTRBeginTrimmed	End of 5'UTR, beginning of IG/TCR CDS as observed in the data
L1Begin	End of 5'UTR, beginning of IG/TCR CDS
L1End	End of first exon, beginning of V intron
VIntronBegin	End of first exon, beginning of V intron
VIntronEnd	End of V intron, beginning of second exon
L2Begin	End of V intron, beginning of second exon
L2End	End of lider sequence, beginning of sequence that codes IG/TCR protein, beginning of FR1.
FR1Begin	End of lider sequence, beginning of sequence that codes IG/TCR protein, beginning of FR1.
FR1End	End of FR1, beginning of CDR1
CDR1Begin	End of FR1, beginning of CDR1
CDR1End	End of CDR1, beginning of FR2
FR2Begin	End of CDR1, beginning of FR2
FR2End	End of FR2, beginning of CDR2
CDR2Begin	End of FR2, beginning of CDR2
CDR2End	End of CDR2, beginning of FR3
FR3Begin	End of CDR2, beginning of FR3
FR3End	End of FR3, beginning of CDR3
CDR3Begin	End of FR3, beginning of CDR3
VEndTrimmed	End of V region after V(D)J rearrangement (commonly inside CDR3)
VEnd	End of V region in genome
DBegin	Beginning of D region in genome
DBeginTrimmed	Beginning of D region after VDJ rearrangement
DEndTrimmed	End of D region after VDJ rearrangement
DEnd	End of D region in genome

Continued on next page

Table 9.2 – continued from previous page

JBegin	Beginning of J region in genome
JBeginTrimmed	Beginning of J region after V(D)J rearrangement
CDR3End	End of CDR3, beginning of FR4
FR4Begin	End of CDR3, beginning of FR4
FR4End	End of FR4
CBegin	Beginning of C Region
CExon1End	End of C Region first exon (Exon 3 of assembled TCR/IG gene)
CEnd	End of C Region

CHAPTER 10

Appendix

TCR/BCR refenrece sequences library

Default list and sequences of `V`, `D`, `J` and `C` genes used by MiXCR are taken from GenBank. Accession numbers of records used for each locus are listed in the following table:

Homo sapiens	`TRA/TRD`	NG_001332.2
	`TRB`	NG_001333.2
	`TRG`	NG_001336.2
	`IGH`	NG_001019.5
	`IGK`	NG_000834.1
	`IGL`	NG_000002.1
Mus musculus	`TRA/TRD`	NG_007044.1
	`TRB`	NG_006980.1
	`TRG`	NG_007033.1
	`IGH`	NG_005838.1
	`IGK`	NG_005612.1
	`IGL`	NG_004051.1

Alignment and mutations encoding

MiXCR outputs alignments in `exportClones` and `exportAlignments` as a list of 7 fields separated by `|` symbol as follows:

```
targetFrom | targetTo | targetLength | queryFrom | queryTo | mutations |
alignmentScore
```

where

- `targetFrom` - position of first aligned nucleotide in **target sequence** (sequence of gene feature from reference V, D, J or C gene used in alignment; e.g. `VRegion` in TRBV12-2); this boundary is inclusive
- `targetTo` - next position after last aligned nucleotide in **target sequence**; this boundary is exclusive
- `targetLength` - length of **target sequence** (e.g. length of `VRegion` in TRBV12-2)
- `queryFrom` - position of first aligned nucleotide in **query sequence** (sequence of sequencing read or clonal sequence); this boundary is inclusive
- `queryTo` - next position after last aligned nucleotide in **query sequence**; this boundary is exclusive
- `mutations` - list of mutations from **target sequence** to **query sequence** (see below)
- `alignmentScore` - score of alignment

all positions are zero-based (i.e. first nucleotide has index 0)

Mutations are encoded as a list of single-nucleotide edits (similar to what is used in definition of Levenshtein distance, i.e. insertions, deletions or substitutions); if one apply these mutations to aligned subsequence of **target sequence**, one will obtain aligned subsequence of **query sequence**.

Each single mutation (single-nucleotide edit) is encoded in the following way (without any spaces; some fields may absent in some cases, see description):

`type` [`fromNucleotide`] `position` [`toNucleotide`]

- type of mutation (one letter):
- `S` for substitution
- `D` for deletion
- `I` for insertion
- fromNucleotide is a nucleotide in **target sequence** affected by mutation (applicable only for substitutions and deletions; absent for insertions)
- position is a zero-based absolute position in **target sequence** affected by mutation; for insertions denotes position in **target sequence** right after inserted nucleotide
- toNucleotide nucleotide after mutation (applicable only for substitutions and insertions; absent for deletions)

Note, that for deletions and substitutions

```
targetSequence[position] == fromNucleotide
```

i.e. target sequence always have fromNucleotide at position position; for insertions fromNucleotide field is absent

Here are several examples of single mutations:

- `SA4T` - substitution of `A` at position `4` to `T`
- `DC12` - deletion of `C` at position `12`
- `I15G` - insertion of `G` before position `15`

Consider the following BLAST-like alignments encoded in MiXCR notation:

- Alignment without mutation

subsequence from `target` (from nucleotide 0 to nucleotide 15) was found to be identical to susequence from `query` (from nucleotide 3 to nucleotide 18).

- Alignment with mutation

so, to obtain subseqeunce from **query sequence** from 3 to 18 we need to apply the following mutations to subsequence of **target sequence** from 2 to 16: - deletion of `G` at position `7` - substitution of `C` at position `9` to `T` - insertion of `C` before at position `13`

CHAPTER 11

Utility actions

Version info

In order to check the current version of MiXCR as usual one can use `-v` option:

```
> mixcr -v
MiXCR v2.1 (built Mon Feb 06 19:56:13 MSK 2017; rev=a9958cd; branch=release/v2.1)
RepSeq.IO v1.2.6 (rev=958e019)
MiLib v1.7.1 (rev=f6ccdbc)
Built-in V/D/J/C library: repseqio.v1.2

Library search path:
- built-in libraries
- /Users/dbolotin/.
- /Users/dbolotin/.mixcr/libraries
```

In order to check which version of MiXCR was used to build some vdjca/clns file:

```
> mixcr versionInfo file.vdjca
MagicBytes = MiXCR.VDJC.V06
MiXCR v1.8-SNAPSHOT (built Fri Jan 29 16:16:40 MSK 2016; rev=327c30c; branch=feature/
↪mixcr_diff); MiLib v1.2 (rev=4f56782; branch=release/v1.2); MiTools v1.2␣
↪(rev=eb91603; branch=release/v1.2)
```

Merge alignments

Allows to merge multiple `.vdjca` files into a single one:

```
> mixcr mergeAlignments file1.vdjca file2.vdjca ... output.vdjca
```

CHAPTER 12

License