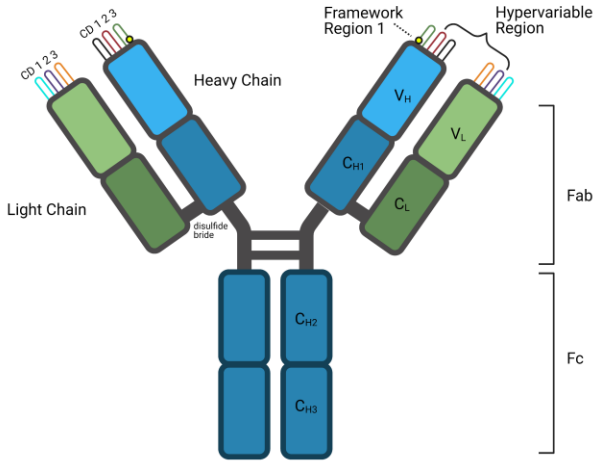


BCR/TCR ANALYSIS PIPELINE

Developed by Rachael J. M. Bashford-Rogers

Wellcome Trust Centre for Human Genetics, Oxford

Rachael Bashford-Rogers
rbr1@well.ox.ac.uk

Program developed by Rachael Bashford-Rogers (2020)
At the Wellcome Centre for Human Genetics, University of Oxford
(rbr1@well.ox.ac.uk)

Table of Contents

Part 1

1. Introduction

This manual provides an outline of the **BCR/TCR processing pipeline for NGS data** based on the BCR and TCR wet lab and computational methods developed in the Bashford-Rogers Lab. For analysing sequencing data we provide three solutions:

a) Python based wrapper for job submission using bsub:
Processing_sequences_large_scale.py
b) Qsub standard bash job submission bash script.
BCR_TCR_Wrapper_Cluster.sh. Preferable for running on the **BMRC cluster.**
c) Bash pipeline manager for running the full pipeline.

A summary of the pipeline is described below:

Stage 1:

1. QC sequences

Stage 2:

1. Join forward and reverse reads (merging)
2. Split sequences according to sample barcode
3. Identify RNA barcode and collapse/error correct based on groups of sequences sharing same barcode
4. Check isotype against reference
5. Check matches to IGHV/J reference sequences
6. Check open reading frame present

Stage 3:

1. Network generation: Here, each vertex represents a different sequence, and the number of identical BCR sequences defines the vertex size. Edges are created between vertices that differ by one nucleotide. Clusters are groups of interconnected vertices (*1, 2*). The program described here calculates edges between unique sequences and determines vertex sizes, creating output files in formats that can directly be used in network analysis programs such as networkx (python) or igraph (R or python).

Stage 4:

1. Sequence annotation
2. Network Analysis
3. Generation of broad repertoire statistics

Stage 5 (optional):

1. Run the optional R script to analyse and visualise summary metrics from the results of Stages 1-4. This will also concatenate files for all samples.
2. Check the percentage of reads which pass Open-Read-Frame filtering.
    a. If this is abnormally low (potentially due to large clonal expansion) consider running the analysis with the ORF column in the sample sheet set to anything other than TRUE. To prevent reads being removed.

Stage 6:

1. Concatenate filtered fastq files into a smaller number of multi-individual large files.
2. Upload large files to **IMGT** for annotation.

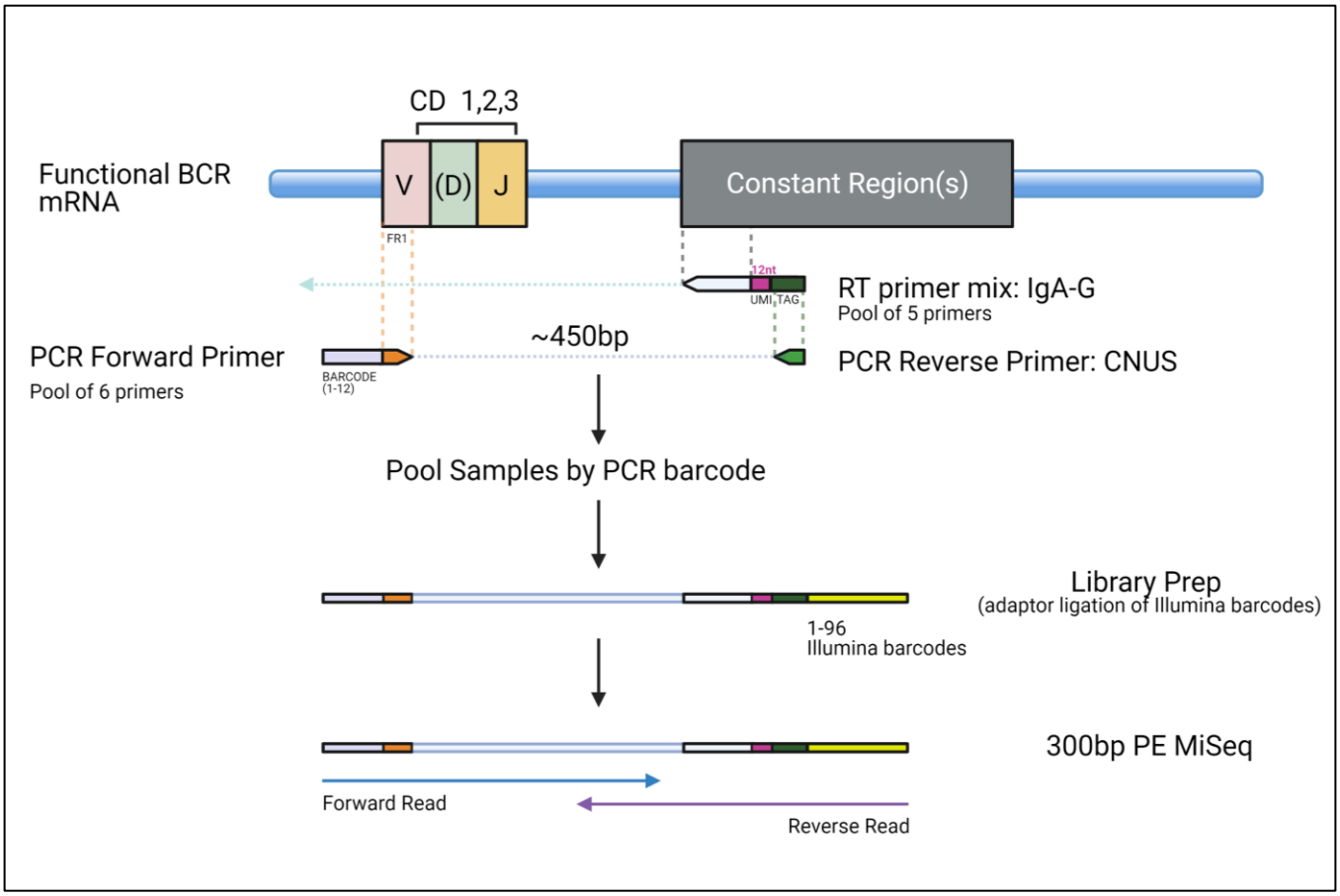
Stage 7:

1. Results of IMGT analysis are used in Isotyper specific Analysis

Figure 1. Schematic of lab protocol for BCR repertoire amplification.

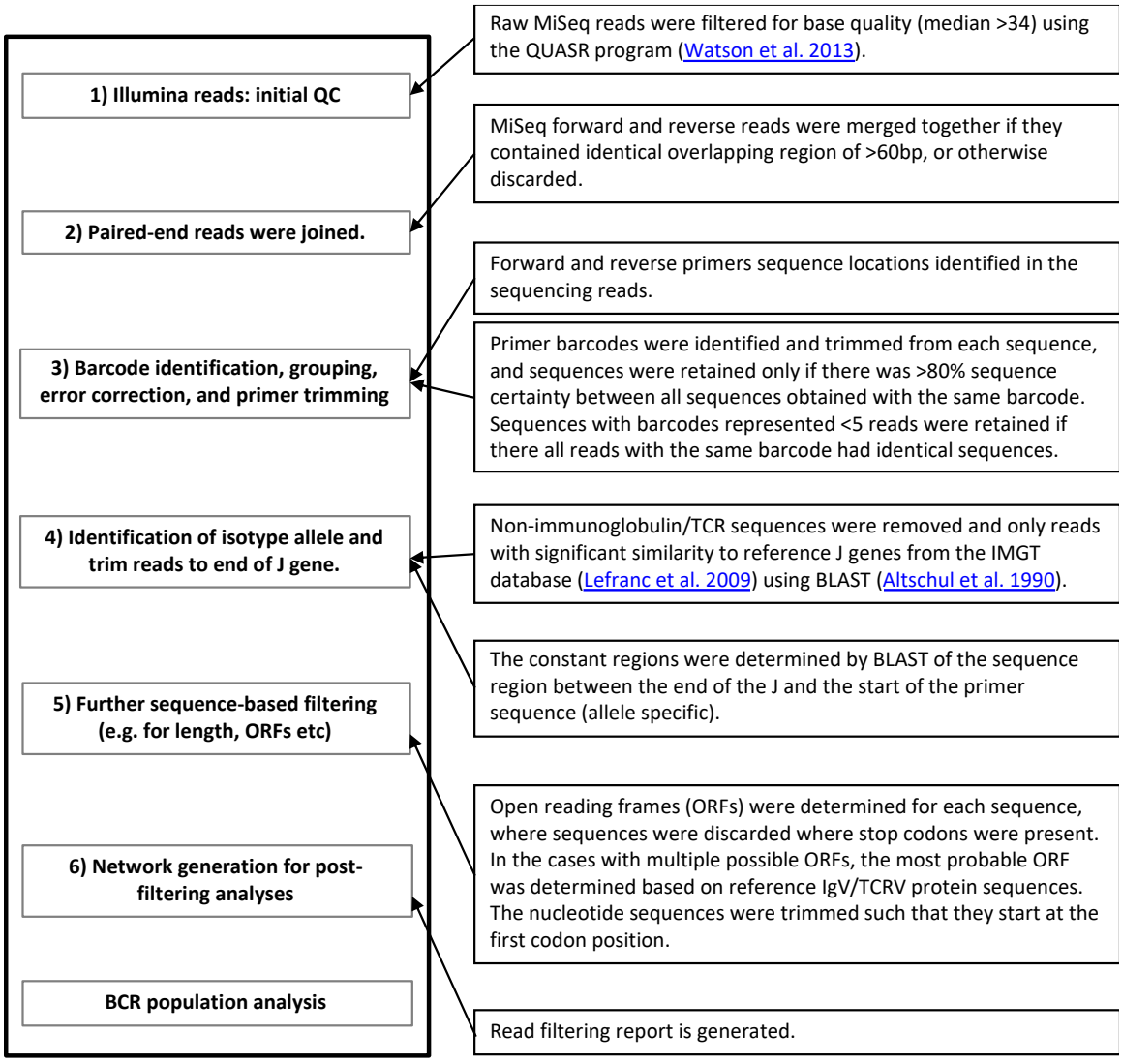
- A reverse transcription (RT) primer pool (5 primers binding to the constant region of each immunoglobulin, incorporating a unique molecular identifier (UMI)) is used to reverse transcribe RNA.
- cDNA is then amplified by PCR using a reverse primer which binds to a tag on the RT primers and a pool of 6 barcoded forward primers which bind to the Framework Region 1 of all known BCR V genes. Forward primers are barcoded with 1 of 12 barcodes to enable sample multiplexing.
- Samples are then pooled according to PCR barcode e.g. 12 samples with barcodes 1-12 could be pooled together but not two samples both barcoded with barcode 12.
- Pooled samples are then library prepped with Kapa Dual Index.
- Libraries with different Illumina barcodes are pooled in a one-to-one ratio.
- Libraries are sequenced on a MiSeq using 300bp Paired End technology with 15-20% PhiX to improving base calling.

The protocol for TCR repertoire amplification differs in primer design but is comparable.



Created with BioRender.com

Figure 3. Schematic of QC and filtering pipeline.



2. Installation

Note: Jobs should be from within the pipeline directory relative paths are used.
Note: Ensure access to all required packages and files e.g. chmod 777 files.txt (read-write and execute access).

1. **FLASH:**
   - Download current version from https://ccb.jhu.edu/software/FLASH/
   - Unpack.

2. **CD-HIT(*3*):**
   - Download current CD-HIT from: http://bioinformatics.org/cd-hit/
   - Unpack the file with "tar xvf cd-hit-XXX.tar.gz --gunzip"
   - Change dir by "cd cd-hit-2006"
   - Compile the programs by "make"

3. **Quasr(4):**
   - Download current version from: https://sourceforge.net/projects/quasr/

4. **Blast:**
   - Download current version from: https://blast.ncbi.nlm.nih.gov/Blast.cgi?PAGE_TYPE=BlastDocs&DOC_TYPE=Download
   - **For users of Rescomp (BMRC, Oxford) please not this is already installed so you do not need to perform this stage.**

5. **Python Modules:**
   - Ensure that the following python modules are installed.

Collections	Operator	networkx
Os	Sys	

Note for Rescomp users:	*These modules are already available using the Rescomp 'module load' system and need not be installed locally. To use these modules you MUST run analysis using the **BCR_TCR_Wrapper_Cluster.sh** job submission script **rather than the python wrapper.***

6. **R Modules: Only available with BASH wrapper.**
   - Ensure that the following R modules are installed:

tidyverse	data.table	cowplot
ggplot2	ggforce	gtools
Gviz	foreach	optparse
gridExtra	doParrallel	

Note for Rescomp users:	*These modules are already available using the **Rescomp Bioconductor R module**. To use these modules you MUST run analysis using the **BCR_TCR_Wrapper_Cluster.sh** job submission script **rather than the python wrapper.***

7. Locations of Dependencies:

- To ensure the pipeline can call the dependencies edit: "BCR_TCR_PROCESSING_PIPELINE/ Locations_of_called_programmes.txt" file, providing the full path to correct locations of your versions of:
    - Reference library of genes and primers (already compiled for you)
    - CD-HIT (from above)
    - FLASH (from above)
    - Quasr (*4*) (from above)
    - Blast (from above)

8. Register for IMGT

- Fastq files will be annotated using IMGT/HighV-QUEST – the high throughput gold standard for TCR and BCR repertoire analysis.
- **You must register a user account to use this online tool.**
- All new user accounts require approval by an administrator. Therefore we recommend registering as soon as possible to avoid downstream delays.
    - Register here: http://www.imgt.org/HighV-QUEST/login.action

3. Run

- This pipeline performs the steps described in the introduction, split into several separate stages.
- Each stage can be run **independently** using the python or bash wrappers. Alternatively the pipeline manager can be used to run all the stages of the pipeline, handling job dependencies without additional user input. This is particularly useful when you want to automate the submission of a large number of samples.
- **We provide two wrappers for running independent stages:**
  - **(1)** Python based wrapper for job submission using bsub Processing_sequences_large_scale.py
  - **(2)** Bash standard job submission script using qsub - submits array tasks. BCR_TCR_Wrapper_Cluster.sh.
- **We provide one pipeline manager:** BCR_TCR_PIPELINEMANAGER.sh

Note for Rescomp users:	(2) and the bash pipeline manager are preferable for running on the BMRC cluster. Both utilise the module system and can be easily adapted for another cluster architecture.

3.1 Sample Spreadsheets:

- We provide several example files within the pipeline directory to get started with job submission.
- As samples are double multiplexed (internal PCR barcode and library adapter) you must create two input spreadsheets according to specifications below which enable demultiplexing.
- In most cases you need only adapt the example with the relevant file locations and specify the correct barcodes.

Note:	*Spreadsheet 1 and spreadsheet 2 will be the same if there is no internal barcoding (see example input file "Sample_example_no_internl_barcodes.txt").*

3.1.1 Spreadsheet A - 'name_pre.txt':

- A spreadsheet detailing the libraries and their barcodes to be used in stage 1.
- It is often helpful to create this file in an excel spreadsheet and then save in the relevant format (see below). Example sheet: Samples_WHG_Trial1_Iso_pre1.txt
  - **If running the bash wrapper (.sh) the sample sheet must be tab separated.**
  - **If running the python wrapper (.py) the sample sheet must be space (" ") separated.**
  - There must be no header-line or empty lines at the end of the txt file.

3.1.2 Spreadsheet B - 'name_post.txt':

- A spreadsheet detailing the individual samples, linking them to their respective library. This will be used from stage 2 onwards.

- It is often helpful to create this file in an excel spreadsheet and then save in the relevant format (see below). Example sheet: Samples_WHG_Trial1_Iso_post1.txt
    - **If running the bash wrapper (.sh) the sample sheet must be tab separated.**
    - **If running the python wrapper (.py) the sample sheet must be space (" ") separated.**
    - There must be no header-line or empty lines at the end of the txt file.

3.2.3 Batch Files

A tab separated txt file detailing the project experimental design. Only required for the bash wrapper and pipeline manager. This will enable the pipeline to compare VDJ sharing across lanes/libraries to assess contamination and also to generate summary plots according to experimental design allowing you to identify potential batch effect.

- In most cases you need only adapt the example with the relevant information.
- **Please leave columns with the specified column name.**

SampleID	Barcode	Lane	Plate	Library	Position	PCRBarcode
	The same as SampleID unless there are multiple samples from the same individual e.g blood and biopsy. Naming convention: "Individual"_"Sample e.g. Volunteer1_1 vs Volunteer1_2				Location on PCR Plate e.g. A1	Internal PCR barcode – doesn't matter how these are named just that the same convention is used for all samples e.g A-L

3.2.3 Technical Files:

A tab separated txt file detailing the project experimental design, specifically which samples are technical replicates and included across multiple lanes and runs. Only required for the bash wrapper and pipeline manager. This will enable the pipeline to summary stats across technical replicates to assess batch effect.

- In most cases you need only adapt the example with the relevant information.
- **Please leave columns with the specified column name.**
- **Use the same table structure as the batch file specified above.**

3.2 Python Wrapper

The python wrapper, Processing_sequences_large_scale.py will manage the job submission for multiple samples using bsub for any **single** stage of the pipeline*. It will not run all the stages at once.*

3.2.1 Python Wrapper: Command Line Arguments

In order to run the wrapper we must provide the following command line arguments in the order shown:

Argument 1:	Sample input file – name_pre.txt or name_post.txt (**space separated format**)
Argument 2:	Stage of pipeline – 1/2/3/ etc.
Argument 3:	Run script in job format - Y/N

Argument 4:	Print commands to screen - Y/N
Argument 5 :	Run pipeline – Y/N

3.2.2 Submission

The python wrapper can be run in the following way (from within the pipeline directory):

```
python Processing_sequences_large_scale.py <sample file list> <commands (comma separated list)> <run as a job: Y/N> <print commands: Y/N> <run commands: Y/N
```

Note: *Unless argument 5 is Y, then no analysis will be performed.*

3.3 BASH Wrapper

The Bash wrapper will manage the job submission for **any single stage** of the pipeline using qsub (BCR_TCR_Wrapper_Cluster.sh)***.*** *Users of other clusters may choose to use this wrapper but will need to adjust the module load commands for equivalent.*

3.3.1 Command Line arguments

In order to run the wrapper we must provide the following command line arguments in the order shown.

Argument 1:	Dependencies – Y/N – whether to check the success of previous jobs before running this analysis.
Argument 2:	Sample input file, either pre.txt or post.txt (**tab separated format**) depending on stage.
Argument 3:	Stage of pipeline – 1/2/3/ etc.
Argument 4:	Run name e.g. TRIAL. **Or if running stage 2 this will be samples file pre (for dependency checking)!**
Argument 5:	Batch File.
Argument 6:	Which Jaccard Task to run: 1: Basic jaccard 2: jaccard for VDJs found at least twice 3: jaccard on UMIs (pre consensus)
Argument 7:	Technical file (enables you to assess batch effect across lanes/batches).

You will need to edit the file adjusting the queue (short or long) and the memory usage according to stage. To do this add the following shebang to the top of the script and edit accordingly.

```
#!/bin/bash
#$ -cwd
#$ -e COMMANDLOGS      (location of error files)
#$ -o COMMANDLOGS      (location of output files)
#$ -N <name_of_job>
#$ -q long.qc                  (long = 10 days, short = 30 hours)
#$ pe shmem <number_of_slots>   (each slot is 16Gb of RAM)
```

3.3.2 Submission

The bash wrapper can be run in the following way (from within the pipeline directory).

- If you are submitting a single sample **–t 1**
- If you are running multiple samples, submit using as an **array task** specifying **–t 1-<number of samples>** which will submit each sample as its own task.
- **Note some stages are run on the combined output of multiple samples therefore this argument will be –t 1.**

qsub -t <1-no. samples> BCR_TCR_Wrapper_Cluster.sh <Arguments 1-5>

3.4 Pipeline Manager

We provide a **bash pipeline manager** to automate job submission on Rescomp. This could be adapted for a similar linux based cluster architecture. The pipeline manager is recommended when running multiple batches/jobs with large numbers of samples as it can be challenging to manually keep track of job ids, errors and completion status when running each stage independently. Additionally, automation decreases run time as there is no user submission delay, for example if jobs complete over a weekend/night then the next job will automatically run.

Running the pipeline manager is very simple, and will ensure all the relevant stages run in series provided there are no errors. You must supply the following arguments:

Argument 1:	Dependencies – Y/N – whether to check the success of previous jobs before running this analysis.
Argument 2:	Path to Sample File PRE
Argument 3:	Path to Sample File POST
Argument 4: * **	Part of pipeline to run detailed in table below.
Argument 5:	Run name e.g. TRIAL
Argument 6:	Path to batch file
Argument 7:	Path to technical file

***Argument 4:**

		Command
1-5+RS	pre-IMT	**1**
1-5+RS+basic Jaccard	pre-IMT	**2**
1-5+RS+basic and consensus Jaccard	pre-IMT	**3**
User intervention – download fully_reduced.fa files and upload to IMGT HighV-Quest. Upload the IMGT output to output directory See [3.5.6]-[3.5.7] for more details.		
6 + isotyper scripts (productive, unproductive and all)	Post-MGT	**4**
Isotyper scripts (productive, unproductive and all). *Stage 6 can take a while so there is no need to rerun if you just want to rerun isotyper with different sample depths.*	Post-IMGT	**5**

To run the pipeline manager enter the below code from within the code directory.

./BCR_TCR_PIPELINEMANAGER.sh <Dependency check Y/N> <sample file pre> <sample file post> <stage> <run name> <optional batch file> <Technical Samples>

Notes:

- Jobs will be held in the queue until dependant jobs have been run to completion.
- To ensure jobs will **only run** if prior jobs dependencies have been run successfully to completion without error, each successful array task (sample) will output the sample ID of successful samples to a single summary file upon completion, which can be inspected.
    - Only if all samples are found in this file will the subsequent jobs run.
- In the case of a **sample failure** a list of failed samples will be output to a corresponding ***task*FAILED_SAMPLES.txt** file which can be inspected.
    - Usually failures result from a sample having 0 reads post filtering.
    - These samples must be removed from the sample sheets and rerun to avoid error.
    - **Dependency checking is specified using the argument Y (recommended) or turned off using N.**
- All log (e/o) files will be output to a named directory within COMMANDLOGS.
- 'Check files' containing sample id will be output directly to the COMMANDLOGS directory.

WARNING: If you turn off dependency checking (N) you will need to manually check the error files to ensure that all jobs ran successfully to completion for each stage.

3.5 Run Pipeline

3.5.1 Stage 1: QC Sequences

Here, you run the wrapper over the Illumina barcode-split files (from **spreadsheet A – pre.txt**) using the stage command-line argument [1]. This stage will split the Illumina-barcode split files into PCR-barcode split files.

python Processing_sequences_large_scale.py <sample file list pre> 1 N Y Y
qsub -t <**1-no. samples**> BCR_TCR_Wrapper_Cluster.sh <dependencies> <sample file list **pre**> **1**

3.5.2 Stage 2: Read Preparation

Here, you run the wrapper over the PCR-barcode split files (from spreadsheet B – post.txt) using the stage command-line argument [2]. The following steps will be performed:

a) Join forward and reverse reads (merging)
b) Split sequences according to UMI.
c) Identify RNA barcode and collapse/error correct based on groups of sequences sharing same barcode
d) Check isotype against reference
e) Check matches to IGHV/J reference sequences
f) Check open reading frame present

python Processing_sequences_large_scale.py <sample file list post > 2 N Y Y
qsub -t <**1-no. samples**> BCR_TCR_Wrapper_Cluster.sh <dependencies> <sample file list **post**> **2** <samples file **pre**>

3.5.3 Stage 3: Network Generation

Here, you run the wrapper over the pcr-barcode split files (from spreadsheet B – post.txt) using the stage command-line argument [3].

- In summary: each vertex represents a different sequence, and the number of identical BCR sequences defines the vertex size. Edges are created between vertices that differ by one nucleotide. Clusters are groups of interconnected vertices (*1*). The program described here calculates edges between unique sequences and determines vertex sizes, creating output files in formats that can directly be used in network analysis programs such as networkx (python) or igraph (R or python).

python Processing_sequences_large_scale.py <sample file list post > 3 N Y Y
qsub -t <**1-no. samples**> BCR_TCR_Wrapper_Cluster.sh <dependencies> <sample file list **post**> **3**

3.5.4 Stage 4: Annotation

Here, you run the wrapper over the PCR-barcode split files (from spreadsheet B – post.txt) using the stage command-line argument [4]. The following steps will be performed:

a. Sequence annotation
b. Generation of broad repertoire statistics

python Processing_sequences_large_scale.py <sample file list post > 4 N Y Y
qsub -t <**1-no. samples**> BCR_TCR_Wrapper_Cluster.sh <dependencies> <sample file list **post**> **4**

3.5.5 Stage 5: Optional R analysis

At this stage it is strongly recommended that you visualise the results of the first half of the pipeline. To do this we have written several R auxiliary functions that can be run on the files from stages 1-4 using the stage command-line argument [RS]. **This is stage is only available for use with the BASH wrapper.** You must provide a run-name as a 4th command line argument.

1. This will concatenate files for all samples into summary files within the output directory.
2. This will produce some summary plots for visualising data within the output directory.
3. You may need to adjust the width and height of the pdf files within the auxillary functions depending on how many samples you have.

qsub -t **1** BCR_TCR_Wrapper_Cluster.sh <dependencies> <sample file list **post**> **RS** <**Runname** >

You may also run jaccard analysis replacing RS with 'JACCARD'. This looks at the overlap between VDJ repertoires to assess for contamination. You must additionally specify the Jaccard Task (see 3.3.1 Command Line arguments)

3.5.6 Stage 6: File Reduction

Here, you run the wrapper over the PCR-barcode split files (from spreadsheet B – post.txt) using the stage command-line argument [5].

- At this stage the finalised fastaq files from each sample are concatenated into a fewer number of larger files named reduced named in the format which makes them easier to upload to IMGT.
- Download these to your computer (if working on a cluster). Make sure you download **ALL** the files following the below naming convention. **Warning: depending on how your directory is sorted (e.g. by size) they may not appear adjacent to each other so you will need to check you have downloaded them all e.g. sort by 'name' or by 'changed'.**

OUTPUTDIR/ORIENTATED_SEQUENCES/NETWORKS/Fully_reduced_<name of sample file_list>_post<no.X>.txt

python Processing_sequences_large_scale.py <sample file list post > 5 N Y Y
qsub **-t 1** BCR_TCR_Wrapper_Cluster.sh <dependencies> <sample file list **post** > **5**

3.5.7 Stage 7: IMGT Analysis

It is recommended at this point to run the output files from **3.4.5** through IMGT/HighV-QUEST (http://www.imgt.org/HighV-QUEST/). This provides the gold standard annotation of BCR/TCR repertoire sequences. Use **the default submission parameters** and select csv and AIRR as the provided output format, specifying the relevant chain. You will receive an email when analysis has finished. Download the completed files and upload to the cluster. Do not de-compress – this will be performed in the next stage.

To make the IMGT_RAW directory use the following linux command:

mkdir <outputdir>/ ORIENTATED_SEQUENCES/ANNOTATIONS/IMGT_RAW

Save the IMGT files output files in:
OUPUTDIR/ORIENTATED_SEQUENCES/ANNOTATIONS/IMGT_RAW

3.5.8 Stage 8: Extracting IMGT Analysis

In order to extract per sample analysis from IMGT you must run the wrapper over the PCR-barcode split files (from spreadsheet B – post.txt) using the stage command-line argument [6]. Remember to save the IMGT files (compressed) in the IMGT_RAW directory from 3.4.7. The output of this stage will be saved in IMGT_SPLIT.

python Processing_sequences_large_scale.py <sample file list> 6 N Y Y
qsub -t <1 > BCR_TCR_Wrapper_Cluster.sh <sample file list post> 6

3.5.9 Stage 9: Setting Subsample Depth

Prior to running the isotyper (productive/non-productive/combined) analysis you must first adjust the subsample depth by editing the

'/ORIENTATED_SEQUENCES/ANNOTATIONS/Sampling_depth_per_isotype_' files. This file specifies the depth of subsampling for each isotype /chain.

- Adjust the value in the 'min' column for all listed constant regions – this will be used as the subsample depth for each constant region.
- The 5th percentile often offers a good compromise between sample-loss and sufficient read depth to prevent the relationship between sample size and statistic. However this is very much data-set dependant.
- Some of the plots that will be generated at the next stage show the correlation between each measure and the read depth. If you notice that a lot of these are still very significant you may need to edit this file again by increasing the read depth and resubmit.

3.5.10 Stage 10: Running Isotyper Analysis

After setting the subsampling depth you may then proceed with running the isotyper analysis script which will calculate a variety of different metrics from the IMGT analysis, using down-sampling to correct for the correlation between read depth.

- To run all, productive and unproductive reads use the argument **ISO_COMPLETE**
- To run on either unproductive or productive reads only: **ISO1_PRODUCTIVE** and **ISO1_NON_PRODUCTIVE** commands respectively.
- **You can rerun this stage using different user specified read depths, provided that each time you rerun it you rename the old ISOTYPER directory to something unique e.g. ISOTYPER_TRY1 to prevent the directory being overwritten**.
- In this way you can have multiple directories containing metrics calculated at different read depths. This is beneficial as in some cases the user specified read depth is not sufficiently large to negate the effect of read depth and requires some fine-tuning.

python Processing_sequences_large_scale.py <sample file list> ISO1 N Y Y
qsub -t <1 > BCR_TCR_Wrapper_Cluster.sh <dependencies> <sample file list post > **ISO_COMPLETE**

This stage will generate a variety of outputs including plots in the Plot directory and files which can be found in the Summary directory. The key file is a matrix of samples (that have passed the read depth filter) by multiple different metrics which describe the repertoire. This file is named. You can use these measures to compare health/disease etc. in your own statistical analysis.

isotyper_metrics_filtered_FINAL_METRICS_<readdepth><productivity>.txt

---

1. Output

1.1 **Table 1.** Description of network output files.

File	Description	Format
NETWORKS/Att_SAMPLE.txt	List of unique sequences.	Column 1 = List of unique sequence ids; column 2 = number of reads; column 3 = sequence;
NETWORKS/Cluster_identities_SAMPLE.txt	List of sequences within clusters.	Column 1 =sequence number; column 2 = cluster number; column 3 = sequence ID; column 4 = number of reads;
NETWORKS/Fully_reduced_SAMPLE.fasta	Fasta file of unique sequences.	Sequence ID multiplicity format;
NETWORKS/Plot_ids_SAMPLE.txt	List of sequences for plotting (only sequences that are connected or representing >1 read).	Column 1 =sequence number; column 2 = sequence ID; column 3 = number of reads;

1.2 **Table 2:** Description of annotation output files.

File	Description
ANNOTATIONS/Cluster_statistics_SAMPLE.txt	Cluster statistics
ANNOTATIONS/Constant_region_counts_SAMPLE.txt	Constant region counts
ANNOTATIONS/Distribution_cluster_sizes_SAMPLE.txt	Counts of cluster sizes
ANNOTATIONS/Distribution_vertex_sizes_SAMPLE.txt	Counts of vertiex sizes
ANNOTATIONS/Gene_frequencies_SAMPLE.txt	V/J gene frequencies
ANNOTATIONS/IsoTyper_chain_repertoire_statistics_file_SAMPLE.txt	Network parameters per isotype/constant region
ANNOTATIONS/Network_statistics_SAMPLE.txt	Network parameters (total)
ANNOTATIONS/TMP/Annotation_SAMPLE.txt	Annotation file
ANNOTATIONS/TMP/CDR3_frequencies_SAMPLE.txt	Distribution of CDR3 frequencies
ANNOTATIONS/TMP/CDR3_lengths_SAMPLE.txt	Distribution of CDR3 lengths

1.3 Output Fastq File Format:

To save memory and speed up analysis, output fastqs are stored in the dense multiplicity format described below:

- >IDOFSEQUENCE__X_Y_Z|IGX_IGY_IGZ
- GACGCATGATGCGTAGCAGACGGATATAGC…..

- Where the sequence header provides information of:
  - A unique sequence identifier
  - X, Y and Z correspond to the number of reads mapped to IgX, IgY and IgZ respectively.
- **Note:** the constant region has been trimmed from the sequence and this information is encoded in the header.

2. Author Contribution

Rachael J. M. Bashford-Rogers developed the python based TCR/BCR repertoire analysis pipeline and User Guide.
Lauren E. Overend developed the bash wrappers and pipeline, R functions and module reduction analysis and helped write documentation.

3. References

If you find Immune-Network-Generation useful, please cite reference #2 (*R. J. Bashford-Rogers et al., 2019*).

- 1. R. J. Bashford-Rogers *et al.*, Network properties derived from deep sequencing of human B-cell receptor repertoires delineate B-cell populations. *Genome Res* **23**, 1874-1884 (2013).
- 2. R. J. M. Bashford-Rogers *et al.*, Analysis of the B cell receptor repertoire in six immune-mediated diseases. *Nature*, (2019).
- 3. W. Li, A. Godzik, Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. *Bioinformatics* **22**, 1658-1659 (2006).
- 4. S. J. Watson *et al.*, Viral population analysis and minority-variant detection using short read next-generation sequencing. *Philos Trans R Soc Lond B Biol Sci* **368**, 20120205 (2013).
- 5. T. Magoc, S. L. Salzberg, FLASH: fast length adjustment of short reads to improve genome assemblies. *Bioinformatics* **27**, 2957-2963 (2011).

Part 2: Detailed Methods

1. Flash – Joining Overlapping PE Reads

Parameters:

- [FLASH] Min overlap: 15
- [FLASH] Max overlap: 280
- [FLASH] Max mismatch density: 0.250000
- [FLASH] Allow "outie" pairs: true
- [FLASH] Cap mismatch quals: false
- [FLASH] Combiner threads: 24
- [FLASH] Input format: FASTQ, phred_offset=33
- [FLASH] Output format: FASTQ, phred_offset=33

- **Min overlap set to 15** – based on Figure 6 (*5*). Provides a good balance between detecting correctly merged and incorrectly merged pairs.
- **Max overlap set to 280** – based on Paired End sequencing of 300bp (Total min fragment length would be about 320bp). May need to be adjusted if an alternative sequencing method is used.
- **Function:**
  - Processes each read separately and searches for the correct overlap.
  - If correct overlap is found, reads are merged into ONE extended read which is then used in downstream analysis.
  - Allows only un-gapped alignments as it targets Illumina sequencing platforms.

2. Establishing consensus sequences.

1 read detected with one UMI.	• Taken as consensus
2-15 reads detected with one UMI:	• Sequences aligned using MAFFT. • 80% similarity required in sequence or sequences discarded. • Select most common sequence as consensus.
15+ reads detected with one UMI:	• Cluster the sequences (CD-HIT) and pull out largest cluster. All other sequences discarded. • Sequences aligned using MAFFT. • If cluster bigger than80% of reads take 80% similarity. • 80% confidence required in each base pair to pass filtering. • Select most common sequence

- **Note:** cDNA molecules that originate from different RNAs but with the same UMI will most likely result in the loss of reads from one (or more) RNA molecules. This can happen as despite

using a pool of UMIs of 12nt there can be preferential UMI usage resulting in the incorporation of the same UMI in multiple different RNA molecules.

3. Filtering Raw Reads

Summary of filtering stages from the output of FLASH (joined reads). Note that V and J gene matching is relatively relaxed as annotation will be performed later by IMGT, however this helps identify and filter out garbage sequences.

Filtering Stage	Description
N joined reads	After FLASH – number of reads which were joined.
N reads with UMIs	Total number of joined reads that have a UMI.
N uniq UMIs	Number of unique UMIs (i.e. RNA molecules captured). ***Deduplication to account for amplification biases.***
N reads post-V matching	Number of unique UMIs after filtering out sequences without any similarity to a V gene.
N reads post-J matching	Number of unique UMIs after filtering out sequences without any similarity to a J gene.
N BCR filtered (post ORF filtering)	Number of unique UMIs after filtering out sequences without functional ORF. ***This filter can be turned off.***
N unique BCRs	Number of unique sequences from the pool of unique UMIs. ***Account for multiple mRNA BCRs per cell (e.g. identical BCR but different UMI).***

4. IMGT Output

Summary of files produced by IMGT. Table table from: (you will need to log in to view).
http://www.imgt.org/IMGT_vquest/user_guide#Esummary

File number	File name	Number of columns filled	Results content *
#1	**"Summary"**	**33 (or 29)**	- Alignment score and identity percentage with the closest V and J genes and alleles, - D-REGION reading frame, - FR-IMGT and CDR-IMGT lengths, - Amino acid (AA) JUNCTION, - Description of insertions and deletions if any, - User sequence in the direct orientation, - Sequence orientation at the submission, the number of trimmed "n" before analysis if any, sequence length, sequence category.
#2	**"IMGT-gapped-nt-sequences"**	**18**	- Nucleotide (nt) sequences gapped according to the IMGT unique numbering for the labels V-D-J-REGION, V-J-REGION, V-REGION, FR1-IMGT, CDR1-IMGT, FR2-IMGT, CDR2-IMGT, FR3-IMGT, - nt sequences of CDR3-IMGT, JUNCTION, J-REGION and FR4-IMGT.
#3	**"nt-sequences "**	**118 (57 (V-J), 79 (1D), 91 (2D) 103 (3D))**	- nt sequences of all labels that can be automatically described and delimitated by IMGT/Automat (57 columns for IGL, IGK, TRA and TRG sequences, 79 (if one D), 91 (if two D) or 103 (if 3 D) columns for IGH, TRB and TRD sequences). The 3 last columns evaluate the number of missing nt for partial V-(D)-J-REGION and of uncertain nt in V-REGION
#4	**"IMGT-gapped-AA-sequences"**	**18**	- AA sequences gapped according to the IMGT unique numbering for the labels V-D-J-REGION, V-J-REGION, V-REGION, FR1-IMGT, CDR1-IMGT, FR2-IMGT, CDR2-IMGT, FR3-IMGT, - AA sequences of CDR3-IMGT, JUNCTION, J-REGION and FR4-IMGT.
#5	**"AA-sequences"**	**18**	Same columns as "IMGT-gapped-AA-sequences" (#4), but sequences of labels are without IMGT gaps.
#6	**"Junction"**	**84 (36 (V-J), 50 (1D), 62 (2D), 77 (3D)**	- Results of IMGT/JunctionAnalysis (36 columns for IGL, IGK, TRA and TRG sequences, 50 (if one D), 62 (if two D) or 77 (if 3 D) columns for IGH, TRB and TRD sequences).
#7	**"V-REGION-mutation-and-AA-change-table"**	**11**	- List of mutations (nt mutations, AA changes, codon change, hotspot motifs, AA class identity (+) or change (-)) for V-REGION, FR1-IMGT, CDR1-IMGT, FR2-IMGT, CDR2-IMGT, FR3-IMGT and germline CDR3-IMGT.
#8	**"V-REGION-nt-mutation-statistics "**	**130**	- Number (nb) of nt positions including IMGT gaps, nb of nt, nb of identical nt, total nb of mutations, nb of silent mutations, nb of nonsilent mutations, nb of transitions (a>g, g>a, c>t, t>c) and nb of transversions (a>c, c>a, a>t, t>a, g>c, c>g, g>t, t>g) for V-REGION, FR1-IMGT, CDR1-IMGT, FR2-IMGT, CDR2-IMGT, FR3-IMGT and germline CDR3-IMGT.
#9	**"V-REGION-AA-change-statistics "**	**109**	- nb of AA positions including IMGT gaps, nb of AA, nb of identical AA, total nb of AA changes, nb of AA changes according to AAclassChangeType (+++, ++-, +-+, +--, -+-, --+, ---), and nb of AA class changes according to AAclassSimilarityDegree (nb of Very similar, nb of Similar, nb of Dissimilar, nb of Very dissimilar) for V-REGION, FR1-IMGT, CDR1-IMGT, FR2-IMGT, CDR2-IMGT, FR3-IMGT and germline CDR3-IMGT.
#10	**" V-REGION-mutation-hotspots "**	**8**	- Hot spots motifs ((a/t)a, t(a/t), (a/g)g(c/t)(a/t), (a/t)(a/g)c(c/t)) detected in the closest germline V-REGION with positions in FR-IMGT and CDR-IMGT.
#11	**"Parameters "**		- Date of the analysis, - IMGT/V-QUEST programme version, IMGT/V-QUEST reference directory release, - Parameters used for the analysis: species, receptor type or locus, IMGT reference directory set and Advanced parameters.
#12	**"scFv "**	40	Available only for Advanced functionalities, Analysis of single chain Fragment variable (scFv). - positions and length, CDR_length, JUNCTION for the 2 V-DOMAIN of the scFv