immuneSIM Documentation

Release 0.1

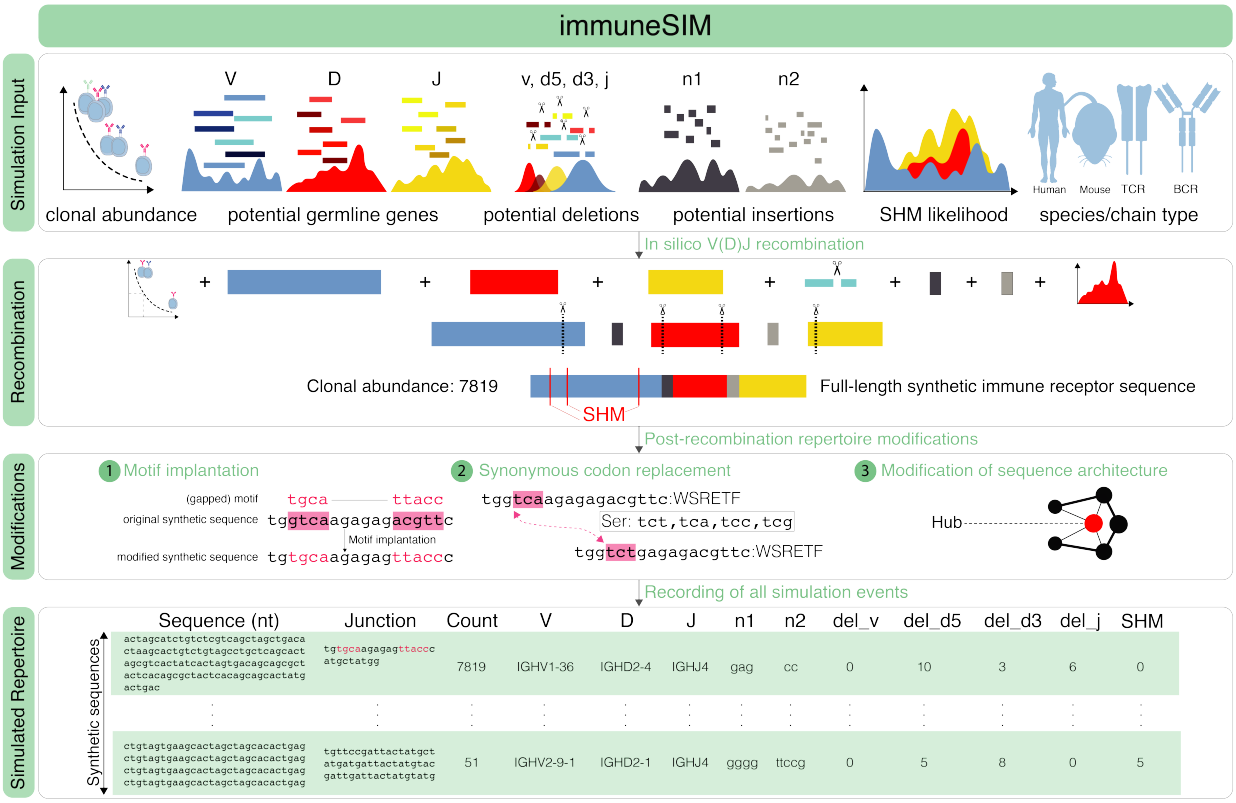
CRW

Mar 05, 2021

Contents:

immuneSIM enables in silico generation of single and paired chain human and mouse B- and T-cell repertoires with user-defined tunable properties to provide the user with experimental-like (or aberrant) data to benchmark their repertoire analysis methods.

The simulation algorithm encompasses an in-silico VDJ recombination process with on-the-go detailed annotation of the generated sequences and if enabled by the user somatic hypermutation (SHM) and sequence motif implantation. The user-definable parameters are: Clone count distribution, Germline Gene Usage, Insertion and Deletion Occurrence, SHM likelihood and Motif Implantation.



CHAPTER 1

Table of Contents

1.1 Quickstart guide

This guide will demonstrate how to use immuneSIM and how to generate a simple simulated immune repertoire.

1.1.1 Overview

The goal of the immuneSIM simulation is to in silico generate human and mouse B- and T-cell repertoires with user-defined properties to provide the user with custom native or aberrant immune receptor sequence repertoires to benchmark their repertoire analysis tools. The simulation algorithm implements an in silico VDJ recombination process with on-the-go annotation of the generated sequences and if enabled by the user somatic hypermutation (SHM) and motif implantation. With a wide range of user-modifiable parameters, a uniquely diverse set of repertoires can be created. The parameters include: Clone count distribution, Germline Gene Usage, Insertion and Deletion Occurrence, SHM likelihood and Motif Implantation.

Prerequisites

To be able to run the code, the following prerequisites are:

1. R >= 3.4.0.
2. Imports: poweRlaw, stringdist, Biostrings, igraph, stringr, data.table, plyr, reshape2, ggplot2, grid, ggthemes, RColorBrewer, Metrics, repmis

Installing immuneSIM

The package can be installed in R (via CRAN or GitHub):

1. Check if all the prerequisites are fulfilled/installed.
2. Execute the following lines in R:

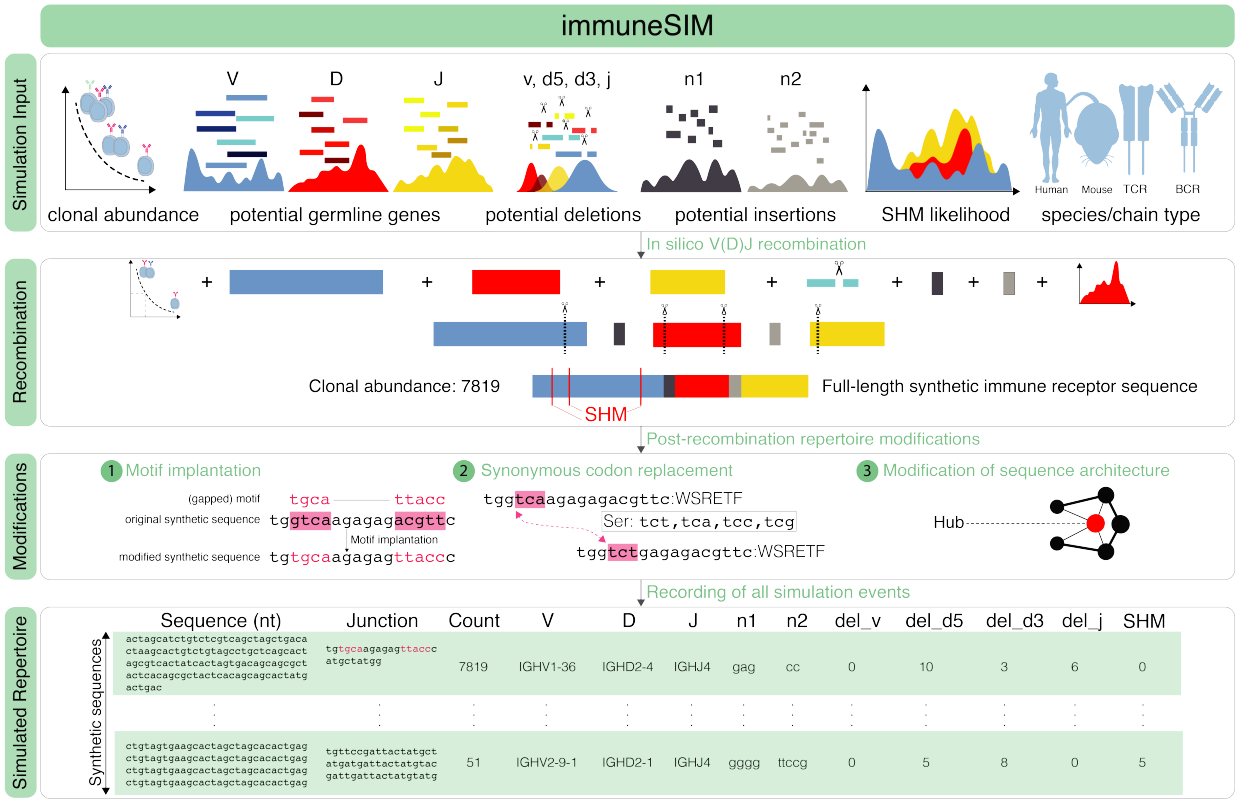


Fig. 1: The ImmuneSIM in silico VDJ recombination is based on our understanding of the VDJ recombination process and includes frequency-based selection of a V, D and J genes and insertions and deletion events in the VD and DJ junctions.

```
#install the devtools package
install.packages("immuneSIM")
```

Alternatively, installation via GitHub is possible in R:

```
#install the devtools package
install.packages("devtools")

#load devtools and install immuneSIM from github
library(devtools)
install_github("GreiffLab/immuneSIM")
```

1.1.2 Workflow of the quickstart simulation

The quickstart simulation using 'immuneSIM' generates a repertoire of a chosen size for a given species and receptor combination. It does not include somatic hypermutation and motif implantation.

The repertoires are simulated by *in-silico VDJ recombination*. Each repertoire will consist of a user-predefined number of fully annotated immune receptor sequences.

The user can generate pdfs summarizing the major features of the generated repertoire that includes: VDJ usage, positional amino acid frequency and gapped-k-mer occurrence.

Performing the analysis

In the quickstart.R, we provide a simple example of murine B-cell repertoire generation based on standard (experimental) parameters:

```
library(immuneSIM)

sim_repertoire <- immuneSIM(
        number_of_seqs = 1000,
        species = "mm",
        receptor = "ig",
        chain = "h",
        verbose= TRUE)

save(sim_repertoire,file="sim_repertoire")

#output plots on repertoire (Note: you need to specify output directory)
plot_report_repertoire(sim_repertoire, output_dir="my_directory/")
```

Output plotting function

The above example ends with the `plot_report_repertoire` function which outputs pdfs of the length distribution, amino acid frequency of most abundant length and VDJ usage plots.

1.2 Simulation

1.2.1 The workflow of a standard immuneSIM simulation

The analysis will consist of the following steps:

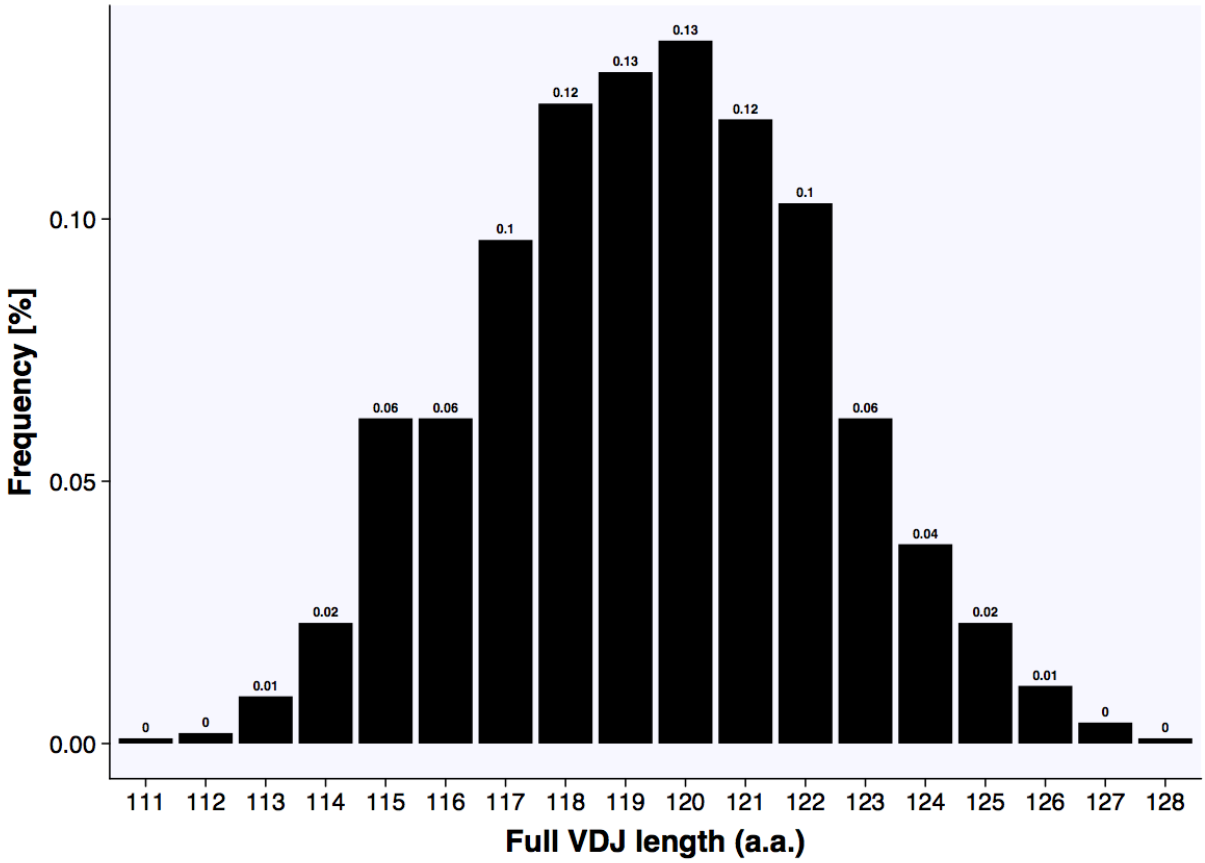


Fig. 2: The length distribution of the full VDJ sequences (a.a).

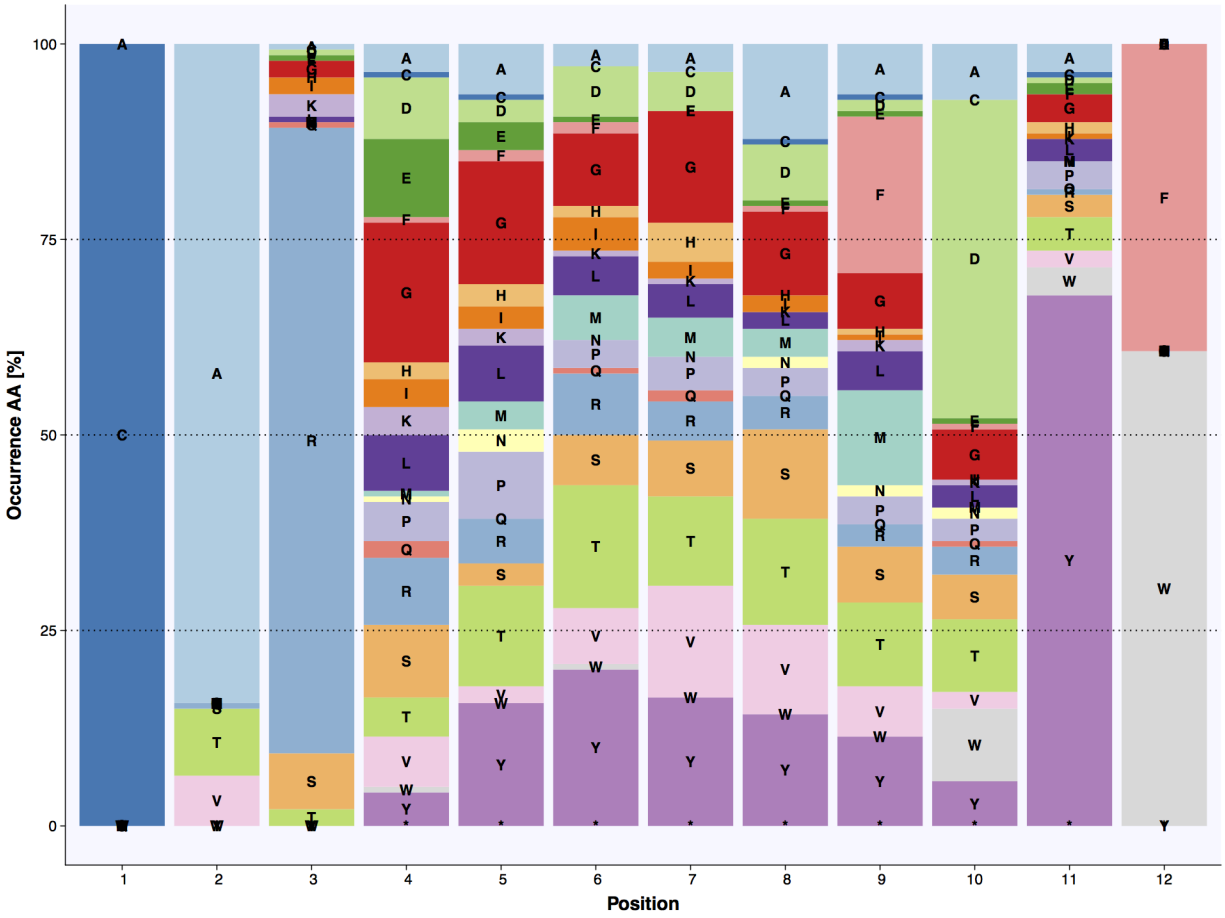


Fig. 3: The positional amino acid frequencies of the most abundant CDR3 length in the simulated dataset.

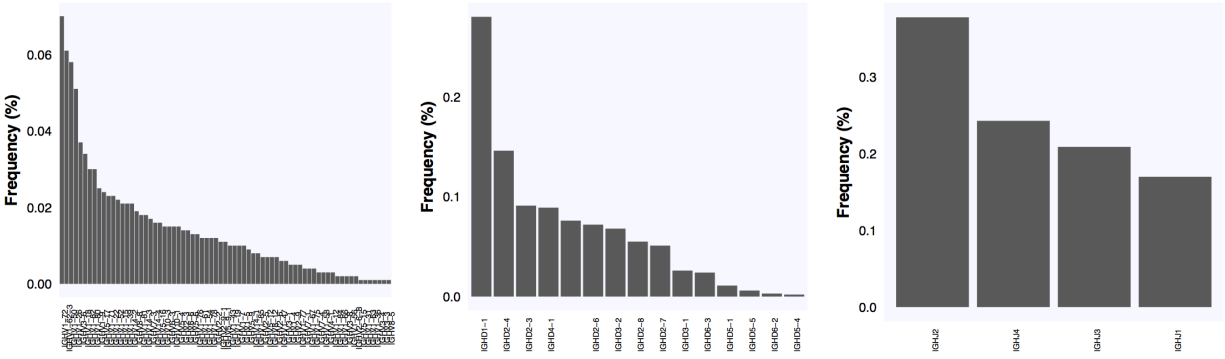


Fig. 4: The V, D, J usage in the simulated repertoire.

1. Simulation of the immune repertoire (including SHM)
2. Post simulation modifications
3. Output of report comparing repertoire to naive experimental repertoire.

The repertoires are simulated by *in-silico VDJ recombination*. Each repertoire will consist of a user-predefined number of fully annotated immune receptor sequences. During the simulation process SHM can be performed based on the previously published AbSIM package[1]. Following the simulation process there are several options to introduce additional biases in the repertoire through *Post simulation modifications*. Finally the user can generate a report about the generated repertoire that includes: VDJ usage, positional amino acid frequency and gapped-k-mer occurence (See: *Report generation*). The entire process is summarized in a flowchart below (See: *Flowchart immuneSIM*)

```
library(immuneSIM)

number_of_sequences <- 1000

mm_igh_sim <- immuneSIM(number_of_seqs = number_of_sequences,
                        vdj_list = list_germline_genes_allele_01,
                        species = "mm",
                        receptor = "ig",
                        chain = "h",
                        insertions_and_deletion_lengths = insertions_and_deletion_
↪lengths_df,
                        user_defined_alpha = 2,
                        name_repertoire = "mm_igh_sim",
                        length_distribution_rand = length_dist_simulation,
                        random = FALSE,
                        shm.mode = 'none',
                        shm.prob = 15/350,
                        vdj_noise = 0,
                        vdj_dropout = c(V=0,D=0,J=0),
                        ins_del_dropout = "",
                        equal_cc = FALSE,
                        freq_update_time = round(0.5*number_of_sequences),
                        max_cdr3_length = 100,
                        min_cdr3_length = 6,
                        verbose = TRUE,
                        airr_compliant = TRUE)

save(mm_igh_sim,file="mm_igh_sim")
```

1.2.2 in-silico VDJ recombination

To enable a simulation in which the prediction of the immune status can be performed for synthetic data, immuneSIM introduces an in-silico VDJ recombination algorithm.

VDJ Recombination includes

- sampling clone count (*Parameter 4: Clone count distribution*)
- sampling V, D and J genes (*Parameter 5: V,D,J germline gene frequencies*)
- sampling insertions and deletions (*Parameter 6: Insertion and deletions*)
- sampling SHM profile (*Parameter 7: Somatic hypermutation likelihood*)

[1] Comparison of methods for phylogenetic B-cell lineage inference using time-resolved antibody repertoire simulations (AbSim), Yermanos et al., Bioinformatics, 33(24), 2017, https://academic.oup.com/bioinformatics/article/33/24/3938/4100159

- introducing signals (*Parameter 10: Motif implantation*)

Output format

The immuneSIM function outputs an R dataframe containing 20 columns and rows equal to the number of sequences simulated. Per sequence immuneSIM provides the following information: * Full VDJ sequence (nucleotide and amino acid): sequence, sequence_aa * CDR3 junctional sequence (nt and aa): junction, junction_aa * VDJ genes used in the recombination event: v_call, d_call, j_call * Nucleotide insertions VD and DJ: np1, np2 * Length of deletion in V, D and J genes: del_v, del_d_5, del_d_3, del_j * CDR3 subsequences from V,D and J genes: v_sequence_alignment, d_sequence_alignment, j_sequence_alignment * Clonal frequency/count information: freqs, counts * Summary of SHM event simulated: shm_events * Given name of repertoire: name_repertoire

VDJ pool

The in-silico VDJ recombination process draws from a pool of germline V, D and J genes. Each immune receptor sequence simulation event starts by sampling a V, D and J germline gene sequence based on its assigned frequency, which is either based on experimental data or the user's preference. To facilitate ease of use, the immuneSIM package includes a library of germline gene datasets for mouse and human BCR and TCR germline genes for both heavy and light and beta, alpha chain (based on IMGT database). Apart from the sequence and annotation information, these datasets include frequencies for each gene as observed in experimental datasets[2].

However, the user is free to add additional datasets with different frequency distributions or with data for species not included so far. (See: *Introducing new germline gene frequencies*)

Insertions and deletions

In the recombination process additional diversity is created through nucleotide insertions and deletions. Deletions occur at the 3' end of the V gene, on 5' and 3' end of the D gene as well as on the 5' end of the J gene. Nucleotides are inserted at the junction of the V and D gene and also between the D and J gene. ImmuneSIM randomly samples a vector of deletion lengths c(del_V,del_D5,del_D3,del_J), subsequently it samples insertions of lengths complementing the resulting V,D and J sequences such that an in-frame sequence results. (See: *Parameter 6: Insertion and deletions*).

NOTE: Due to package-size limitations on CRAN, the dataframe `insertion_and_deletion_lengths_df` contained in the immuneSIM package is only a subset of 500'000 entries of a larger dataset with 11'363'603 entries. The full dataet is provided on GitHub and can be retrieved using the `load_insdel_data()` function.

Clone count

The clone count is simulated to result in a power law distribution of counts, the user can set the alpha parameter that controls the evenness and also has the choice to create a clone count that is equal across all clones.[3] (See: *Parameter 4: Clone count distribution*)

Somatic hypermutations

The **Somatic hypermutations** are simulated based on the previously published AbSIM R package[1]. (See: *Somatic hypermutation*)

[2] Systems Analysis Reveals High Genetic and Antigen-Driven Predetermination of Antibody Repertoires throughout B Cell Development, Greiff et al., Cell Reports, 19(7), 2017, https://www.sciencedirect.com/science/article/pii/S221112471730565X

[3] Fitting Heavy Tailed Distributions: The poweRlaw Package, Gillespie, Journal of Statistical Software, 64(2), 2015, https://www.jstatsoft.org/article/view/v064i02

1.2.3 Post simulation modifications

In this step a variety additional biases can be introduced. This includes:

- Introducing (antigen-specificity simulating) signals. (See: *Motif implantation*)
- Introducing a codon bias. (See: *Parameter 12: Codon bias*)
- Modifying the sequence similarity structure. (See: *Parameter 11: Sequence similarity*)

1.2.4 Report generation

immuneSIM provides the user with a means to output figures in pdf format summarizing the generated repertoire, either by itself or in comparison to a reference repertoire. (See: *Report generation*)

1.2.5 Flowchart immuneSIM

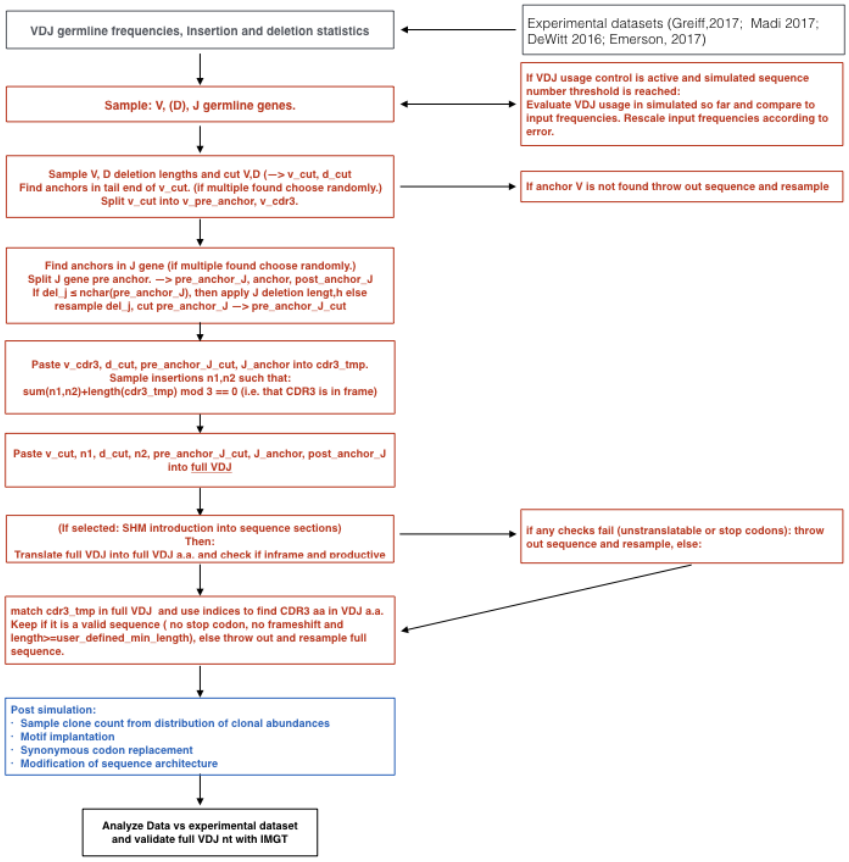


Fig. 5: The step by step breakdown of the in-silico VDJ recombination as performed by immuneSIM.

1.3 Parameters

ImmuneSIM allows the users to control the following parameters:

1.3.1 Parameter 1: Choice of model (species and receptor)

The model organism and receptor type is determined by three main parameters:

- species: Determines the model organism ("mm" = mus musculus, "hs" = homo sapiens) (Default: `species = "mm"`)
- receptor: Determines the receptor type ("ig" = immunoglobulin, "tr" = T-cell receptor) (Default: `receptor = "ig"`)
- chain: Determines the chain type ("h"= heavy, "k" = kappa light, "l" = lambda light, b = "beta", a = "alpha") (Default: `chain = "h"`)

A fourth model parameter *name repertoire* serves to name the repertoire for easier code reference in case many repertoires are simulated. (Default: `name_repertoire = "sim_rep"`)

1.3.2 Parameter 2: Repertoire size (Number of simulated sequences)

Determines the size of the repertoire / number of sequences that are simulated. (Default: `number_of_seqs = 1000`)

1.3.3 Parameter 3: Maximal and minimal amino acid CDR3 length

The parameters *max_cdr3_length* and *min_cdr3_length* control the maximal and minimal threshold for the amino acid CDR3 length (Defaults `max_cdr3_length = 100, min_cdr3_length = 6`).

1.3.4 Parameter 4: Clone count distribution

The clone abundance (clone count) is simulated to result in a power-law distribution of counts using the poweRlaw package[1]. The user can set the alpha parameter via *user_defined_alpha*, which controls the evenness of the distribution (Default: `user_defined_alpha=2`) and also has the choice to create a clone count that is equal across all clones by setting `user_defined_alpha = 1` or by setting `equal_cc = TRUE`.

1.3.5 Parameter 5: V,D,J germline gene frequencies

The sampling of the V, D, J germline genes (V,D,J usage) can be controlled in three ways:

- Choice of frequency distribution for V, D and J genes. (Default: experimental data for chosen species/receptor combination. See: *Table: Reference datasets*)
- Introduction of noise (sampling distribution of values between -1,1, from which noise terms are sampled) modifying the V, D, J frequency distributions.
- Dropout: Option to set frequencies of random or chosen V,D and J genes to 0.

The parameters are:

- *vdj_list*: The list of V, D and J genes and frequencies that provides the input germline gene input for the simulation process. (Default: `vdj_list = list_germline_genes_allele_01`)
- *vdj_noise* (Default: `vdj_noise = 0`): Users can choose a value between 0 (no noise) and 1 (max noise). The value sets the standard deviation for a normal distribution (mean = 0, bounded by -1,1[2]) from which the noise terms introduced in the frequencies are sampled.
- *vdj_dropout* (Default: `vdj_dropout = c(V=0,D=0,J=0)`): Allows the user to drop a specified number of V, D and J genes. The dropped genes are chosen randomly. If the user however desires to drop specific V, D or J genes, they are encouraged to provide a modified vdj_list that sets frequencies of specific genes to 0 (see instructions below).

Introducing new germline gene frequencies

In addition to the germline genes and frequencies provided in the package as *list_germline_genes_allele_01*, the user is free to define a list of germline genes to be used for the simulation process. To do this, it is suggested to extend the *list_germline_genes_allele_01* file or copy its structure. The germline gene dataframes are saved in the list such that they can be identified via a combination of four IDs (species–>receptor–>chain–>gene). For example the murine (mm) immunoglobulin (ig) heavy (h) chain V genes (V) can be accessed using `list_germline_genes_allele_01$mm$tr$b$V`. Below an example of how new germline gene data can be added to the existing `list_germline_genes_allele_01` list:

```
#prepare dataframe containing your V genes including names, nucleotide sequence,␣
↪species and frequency
new_v_gene_df <- data.frame(gene = c("IGHV_synth_1","IGHV_synth_2"),
                                                    allele = c("0X","0X"),
                                                    sequence = c(
↪"aagcagtcaggacctggcctagtgcagccctcacagagcctgtccatcacctgcacagtctctggtttctcattaactagctatggtgtacactgggt
↪",

↪"aagcagtcaggacctggcctagtgcagccctcacagagcctgtccatcacctgcacattctctggtttctgattaaccagctatggtgtacactggga
↪"),
                                                    species = c("synthetic",
↪"synthetic"),
```

(continues on next page)

[1] Gillespie, Colin S., Fitting Heavy Tailed Distributions: The poweRlaw Package. Journal of Statistical Software, 64(2), 1-16 (2015).
[2] https://stackoverflow.com/questions/19343133/setting-upper-and-lower-limits-in-rnorm

(continued from previous page)

```
                                                    frequency = c(0.5,0.5))

#repeat the process for D and J genes
new_d_gene_df <- data.frame(gene = c("IGHD_synth_1","IGHD_synth_2"),
                                                    allele = c("0X","0X"),
                                                    sequence = c(
↪"aggcagcgcagtgccacaacc",
                                                            "gaatacttac"),
                                                    species = c("synthetic",
↪"synthetic"),
                                                    frequency = c(0.8,0.2))

new_j_gene_df <- data.frame(gene = c("IGHJ_synth_1","IGHJ_synth_2"),
                                                    allele = c("0X","0X"),
                                                    sequence = c(
↪"actactttgactactggggccaaggcaccactctcacagtct",

↪"attactatgctatggactactggggtcaaggaacctcag"),
                                                    species = c("synthetic",
↪"synthetic"),
                                                    frequency = c(0.6,0.4))

#extend the existing structure by your genes
list_germline_genes_allele_01_new<-list_germline_genes_allele_01
list_germline_genes_allele_01_new$synth$ig$h$V<- new_v_gene_df
list_germline_genes_allele_01_new$synth$ig$h$D<- new_d_gene_df
list_germline_genes_allele_01_new$synth$ig$h$J<- new_j_gene_df

#simulate repertoire using your germline genes
sim_repertoire <- immuneSIM(
                number_of_seqs = 10,
                vdj_list = list_germline_genes_allele_01_new,
                species = "synth",
                receptor = "ig",
                chain = "h",
                verbose=TRUE)
```

1.3.6 Parameter 6: Insertion and deletions

Two parameters define the insertion and deletion behavior in immuneSIM.

- *insertions_and_deletion_lengths*: Determines the reference file, which stores the insertion deletion information. Default is a dataset based on experimental data[3]. (Default: `insertions_and_deletion_lengths = insertions_and_deletion_lengths_df`). The provided `insertions_and_deletion_lengths_df` is a subset of a larger dataset, which can be downloaded from GitHub using the `load_insdel_data()` function which loads the full dataset (Size = 73 MB).
- *ins_del_dropout*: Enables dropping insertions (ie. N1 or N2 = "") and deletions (del_X = 0). (Default: `ins_del_dropout = ""`). Options are the following:
  - no dropout: ""
  - dropout insertions: "no_insertions"

[3] Greiff, Victor, Systems Analysis Reveals High Genetic and Antigen-Driven Predetermination of Antibody Repertoires throughout B Cell Development. Cell Reports, 19(7), 1467-1478 (2017).

  - dropout deletions: “no_deletions”
  - dropout np1 insertions: “no_insertions_n1”
  - dropout np2 insertions: “no_insertions_n2”
  - dropout deletions V gene: “no_deletions_v”,
  - dropout deletions 5’ end D gene: “no_deletions_d_5”,
  - dropout deletions 3’ end D gene: “no_deletions_d_3”
  - dropout deletions J gene: “no_deletions_j”
  - dropout deletions V gene and 5’ end D gene: “no_deletions_vd”

1.3.7 Parameter 7: Somatic hypermutation likelihood

Determines whether SHM is performed (only for BCRs) and for which likelihood distribution. This function is based on the AbSim package[4] (for details see *Somatic hypermutation*)

- *shm.mode* determines which SHM model is applied or whether it SHM performed at all. (Default: `shm = "none"`). Options are as follows:
  - none
  - data
  - poisson
  - naive
  - motif
  - wrc
- *shm.prob* determines how likely an SHM event is to occur. (Default `shm.prob = 15/350`, i.e. 15 mutation events in a 350nt sequence.)

1.3.8 Parameter 8: Frequency update threshold

Because of the restrictive ‘in-frame’ requirement for all simulated sequences and differences in the likelihood of different V,D,J pairings to meet this requirement, there is a need to adjust the V,D,J frequencies during the simulation process. If this is not performed the repertoire V,D,J usage will not match the input V,D,J frequency distribution. The *freq_update_time* determines the moment in the simulation process where the frequencies are adjusted and is set at 50 percent of sequences simulated. (Default: `freq_update_time = round(0.5*number_of_seqs)`)

1.3.9 Parameter 9: Random repertoires

Generates a repertoire with fully random amino acid sequences and corresponding nucleotide sequences. These sequences are V,D,J germline gene agnostic can serve as the most basic “negative control”.

- *random*: Determines whether a random repertoire should be generated or not. Overrides all other parameters.
- *length_distribution_rand*: Determines the length distribution the random repertoire should have. Default distribution is based on experimental CDR3 data[3].

[4] Yermanos, Alexander, Comparison of methods for phylogenetic B-cell lineage inference using time-resolved antibody repertoire simulations (AbSim). Bioinformatics, 33(24), 3938–3946 (2017).

```
random_repertoire <- immuneSIM(number_of_seqs = 10,
                       name_repertoire = "random",
                       random = TRUE,
                           verbose = TRUE
                       )
```

1.3.10 Parameter 10: Motif implantation

Allows the user to implant motifs into simulated repertoires. The motifs are implanted on the amino acid level with effect on the nucleotide sequence (for details see *Motif implantation*).

- The *motif_implantation* option determines what type of motif is implanted. Can be supplied as either a list of parameters (Number of motifs, length of motifs and frequency of motifs). Or as dataframe containing aa,nt sequences and frequencies.
- *fixed_pos*: Determines whether the motif is introduced in a fixed or randomly determined position

Example: Implanting 2 (n) different motifs of length 3 (k), each in 10 percent of sequences at random positions. Here motifs are implanted into a simulated repertoire, however this function can be used on any AIRR-compliant repertoire.

```
#generate a repertoire into which the motifs should be implanted
sim_repertoire <- immuneSIM(number_of_seqs = 10,
                                        species = "mm",
                                        receptor = "ig",
                                        chain = "h")

modified_repertoire <- motif_implantation(
                                   sim_repertoire,
                                   motif = list("n"=2, "k"=3, "freq" = c(0.1,0.1)),
                                   fixed_pos = 0)
```

1.3.11 Parameter 11: Sequence similarity

Constructs a similarity network for the CDR3 amino acid sequences and deletes the top X percent hub sequences (Default: 0.5 percent) thus impacting the network architecture[5]. Here hubs are deleted from a simulated repertoire, however this function can be used on any AIRR-compliant repertoire. If the option `report = TRUE` is set the excluded sequences will be saved as in the current directory as a .csv file

```
#generate a repertoire for which hub sequences should be deleted
repertoire <- immuneSIM(number_of_seqs = 100,
                                   species = "mm",
                                   receptor = "ig",
                                   chain = "h")

#delete top 0.5 percent of hub sequences from repertoire.
modified_repertoire <- hub_seqs_exclusion(repertoire, top_x = 0.005, report = FALSE)
```

1.3.12 Parameter 12: Codon bias

Allows for the directed replacement of codons with a given probability. This allows the creation of repertoires with entries that differ in their nucleotide sequence but are identical in their amino acid sequence (due to synonym codon

[5] Miho, Enkelejda, Large-scale network analysis reveals the sequence space architecture of antibody repertoires. Nature Communications, 10, 1321 (2019).

replacement). To do this the user needs to provide an *exchange_list* which defines which codons are to be replaced and by which codon. The user also needs to define *skip_probability* the probability with which such a replacement event should occur (i.e. how likely a sequence is skipped. Default: 0). Finally, the parameter *mode* allows the user to define whether the replacement should occur in *both* amino acid and nucleotide sequence (Default) or only in one of the two (*aa* or *nt*). Here codons are replaced in a simulated repertoire, however this function can be used on any AIRR-compliant repertoire. Each codon replacement event is recorded in a column named *codon_replacement*. The information therein can be decoded using the provided `codon_replacement_reconstruction` function which outputs a list of dataframes describing all replacment events.

```
#generate a repertoire in which codons should be replaced
repertoire <- immuneSIM(number_of_seqs = 10,
                                species = "mm",
                                receptor = "ig",
                                chain = "h")

#define codons exchange rules.
exchange_list <- list(tat = "tac", agt = "agc", gtt = "gtg")

#modify repertoire sequences
modified_repertoire <- codon_replacement(
        repertoire,
        mode = "both",
        codon_replacement_list = exchange_list,
        skip_probability = 0.5
        )

#recover codon replacement event information
replacement_list<-codon_replacement_reconstruction(modified_repertoire$codon_
↪replacement)
```

1.3.13 Table: Reference datasets

The following table summarizes the reference datasets for the germline gene frequencies currently contained in immuneSIM. As more reliable TCR alpha and BCR light chain data become available the uniform distributions currently used in immuneSIM will be replaced by experimental data as well.

Table 1: Reference datasets used in immuneSIM.

Species	Receptor chain	Sample	Dataset	Number of sequences
mus musculus (mm)	IGH	healthy_2_nfbc_igm	Greiff, 2017[3]	373'993
mus musculus (mm)	TRB	SRR1339480_Untreated	Madi, 2017[6]	17'752
homo sapiens (hs)	IGH	dewitt_plos_D1_Na	DeWitt, 2016[7]	2'557'564
homo sapiens (hs)	TRB	HIP19048	Emerson, 2017[8]	52'200
homo sapiens (hs)	IGK/L		uniform distribution	0
homo sapiens (hs)	TRA		uniform distribution	0
mus musculus (mm)	IGK/L		uniform distribution	0
mus musculus (mm)	TRA		uniform distribution	0

[6] Madi A., T cell receptor repertoires of mice and humans are clustered in similarity networks around conserved public CDR3 sequences, eLife, 6 (2017).

[7] DeWitt, William S., A public database of memory and naive B-cell receptor sequences, PLOS One, 11(8) (2016).

[8] Emerson, Ryan O., Immunosequencing identifies signatures of cytomegalovirus exposure history and HLA-mediated effects on the T cell repertoire, Nature Genetics, 49(5), 659-665 (2017).

1.4 Somatic hypermutation

1.4.1 Somatic hypermutation model

immuneSIM relies on the AbSim package for the simulation of somatic hypermutation[1]. The SHM is defined by the chosen SHM method `shm.mode` and probability of occurrence `shm.prob` (adapted from the AbSIM documentation[1]):

shm.mode

The mode of SHM speciation events. Options are either "none", "poisson", "naive" "data", "motif", "wrc", and "all". The option "none" leads to a skipping of the SHM process. Specifying either "poisson" or "naive" will result in mutations that can occur anywhere in the heavy chain region, with each nucleotide having an equal probability for a mutation event. Specifying "data" focuses mutation events during SHM in the CDR regions (based on IMGT), and there will be an increased probability for transitions (and decreased probability for transversions). Specifying "motif" will cause neighbor-dependent mutations based on a mutational matrix from high throughput sequencing data sets[2] (Yaari et al., Frontiers in Immunology,2013). "wrc" allows for only the WRC mutational hotspots to be included (where W equals A or T and R equals A or G). Specifying "all" will use all four types of mutations during SHM branching events, where the weights for each can be specified in the "SHM.nuc.prob" parameter.

- none
- poisson
- naive
- data
- motif
- wrc

shm.prob

This determines the probability with which a SHM event occurrs at any position. The default is set at 15/350, i.e. 15 mutation events in a 350nt sequence.

Record of SHM events

Each SHM event is recorded during the simulation process and included in the output dataframe in the column `shm_events`. The entries in the column encode the SHM information as a string. The user may decode this information using the following immuneSIM function which outputs a list of dataframes outlining the locations and SHM mutation per sequence:

```
#generate a repertoire that has undergone SHM
sim_repertoire <- immuneSIM(number_of_seqs = 100,
                                        species = "mm",
                                        receptor = "ig",
                                        chain = "h",
                                        shm.mode="data",
```

(continues on next page)

[1] Comparison of methods for phylogenetic B-cell lineage inference using time-resolved antibody repertoire simulations (AbSim), Yermanos et al., Bioinformatics, 33(24), 2017, https://academic.oup.com/bioinformatics/article/33/24/3938/4100159

[2] Models of somatic hypermutation targeting and substitution based on synonymous mutations from high-throughput immunoglobulin sequencing data, Yaari et al., Frontiers in Immunology, 4, 2013, https://www.frontiersin.org/articles/10.3389/fimmu.2013.00358/full

(continued from previous page)

```
                                        shm.prob=15/350)

#input: shm_events column of immuneSIM repertoire
list_of_shm_event_dfs <- shm_event_reconstruction(sim_repertoire$shm_events)
```

Note on SHM simulation

The integration of our previously published method provides an in-package method to create somatically hypermutated repertoires. If the user wishes to model aspects of SHM simulation not taken into account by AbSIM (Sheng et al., 2017[3]; Kirik et al., 2017[4]; Schramm and Douek, 2018[5]; Guo et al., 2019[6]), any immuneSIM repertoire can be used as input for any other SHM simulation methods (Hoehn et al., 2019[7]) that take AIRR standard repertoires as input.

1.5 Motif implantation

This function allows the user to implant predefined or randomly generated motifs into an immune repertoire. The output is a modified repertoire that contains the modified sequences and information on the sequence and position of the introduced motif.

```
    #generate a repertoire into which the motifs should be implanted
    sim_repertoire <- immuneSIM(number_of_seqs = 10,
                                        species = "mm",
                                        receptor = "ig",
                                        chain = "h")

    #define motif and position (detailed explanation below)
    motif <- list("n"=3,"k"=1,"freq"=c(0.25,0.25,0.25))
    fixed_pos <- 2

#generate and implant motifs
sim_repertoire_motif <- motif_implantation(sim_repertoire, motif,fixed_pos)
```

The motifs are introduced in the CDR3 at both the nucleotide and the amino acid level. Maximally one motif sequence is introduced per immune receptor sequence. The user can exert control over the process via two parameters: *motif* and *fixed_pos*.

1.5.1 Parameter: *motif*

The *motif* parameter controls the motif introduced and allows two different formats.

The first one randomly generates n motifs of a user-defined length that will be introduced at chosen frequencies. This requires the input of a list with three entries:

- 'k': The length of the motif (k-mer).

[3] Gene-Specific Substitution Profiles Describe the Types and Frequencies of Amino Acid Changes during Antibody Somatic Hypermutation., Sheng et al., Frontiers in Immunology, 8, 2017, https://www.frontiersin.org/articles/10.3389/fimmu.2017.00537/full

[4] Antibody Heavy Chain Variable Domains of Different Germline Gene Origins Diversify through Different Paths., Kirik et al., Frontiers in Immunology, 8, 2017, https://www.frontiersin.org/articles/10.3389/fimmu.2017.01433/full

[5] Beyond Hot Spots: Biases in Antibody Somatic Hypermutation and Implications for Vaccine Design, Schramm and Douek, Frontiers in Immunology, 9, 2018, https://www.frontiersin.org/articles/10.3389/fimmu.2018.01876/full

[6] cAb-Rep: A Database of Curated Antibody Repertoires for Exploring Antibody Diversity and Predicting Antibody Prevalence, Guo et al., 9, 2019, https://www.frontiersin.org/articles/10.3389/fimmu.2019.02365/full

[7] Repertoire-wide phylogenetic models of B cell molecular evolution reveal evolutionary signatures of aging and vaccination, Hoehn et al., Proc. Natl. Acad. Sci., 116, 2019, https://www.pnas.org/content/116/45/22664

- 'n': The number of different motifs to be introduced (diversity)
- 'freq': A vector with an entry for each motif that determines how often each motif occurs in the repertoire (sum(freq) should be smaller or equal to 1)

```
#define motif parameters in list (alternatively, the motifs can be supplied␣
↪predefined as a dataframe)
motif<-list("n"=3,"k"=1,"freq"=c(0.25,0.25,0.25))
```

Alternatively, the user may choose to supply a dataframe containing predefined amino acid sequences and nucleotide sequences (both mandatory) as well as the motif frequency:

```
#define motifs to be introduced
motif <- data.frame(aa=c("AA","FF"),nt=c("gccgcc","tttttt"),freq=c(0.4,0.4))
```

1.5.2 Parameter: *fixed_pos*

The *fixed_pos* parameter lets the user define a fixed position in the CDR3 amino acid sequences in which the motif should be introduced (default=0, random position). If the motif cannot be implanted into the user-defined position (i.e. the position does not exist in the sequence) the motif will be introduced at a random position in the CDR3.

1.6 Report generation

ImmuneSIM includes a pdf plot generation tool for quick analysis of the output including comparison to a reference repertoire with respect to:

- Length distribution
- VDJ usage
- Positional amino acid frequency
- Gapped k-mer frequency

Following the simulation of a repertoire as discussed in *Quickstart guide* the in-silico data can thus be analyzed easily. There are two modes 'single repertoire' or 'comparative'. These functions generate and output plots using the ggplot2[1] and other R packages for aesthetics including: ggthemes[2], ComplexHeatmap[3], circlize[4], RColorBrewer[5] and theme.akbar[6]

1.6.1 Single repertoire

Generates pdf plots of full VDJ length distribution, positional amino acid frequency of most common lengths and VDJ usage plots of generated repertoire. (See: *Output plotting function*)

```
    sim_repertoire <- immuneSIM(number_of_seqs = 100,
                                species = "mm",
                                receptor = "ig",
                                chain = "h",
```

(continues on next page)

[1] ggplot2: https://ggplot2.tidyverse.org
[2] ggthemes: https://CRAN.R-project.org/package=ggthemes
[3] ComplexHeatmap: https://bioconductor.org/packages/release/bioc/html/ComplexHeatmap.html
[4] circlize: https://CRAN.R-project.org/package=circlize
[5] RColorBrewer: https://CRAN.R-project.org/package=RColorBrewer
[6] theme.akbar: https://doi.org/10.5281/zenodo.3362026

(continued from previous page)

```
                                            verbose = TRUE)

plot_report_repertoire(sim_repertoire, output_dir = "my_directory/")
```

1.6.2 Comparative

Outputs pdf plots comparing two repertoires chosen by the user with respect to *length distribution*, *vdj usage*, *amino acid frequencies*, *gapped k-mer occurrence*.

```
    repertoire_A <- immuneSIM(number_of_seqs = 100,
                                            species = "mm",
                                            receptor = "ig",
                                            chain = "h",
                                            verbose = TRUE)

    repertoire_B <- immuneSIM(number_of_seqs = 100,
                                        species = "mm",
                                        receptor = "ig",
                                        chain = "h",
                                        verbose = TRUE)

    #Note: you need to specify your output directory.
plot_repertoire_A_vs_B(
    repertoire_A,
    repertoire_B,
    names_repertoires = c("Sim_repertoire_1","Sim_repertoire_2"),
    length_aa_plot = 14,
    output_dir = "my_directory/"
    )
```

Length distribution (amino acid)

The length distribution of amino acid CDR3 sequences is calculated for both repertoires and output as a barplot.

VDJ usage

VDJ usage checks the usage of V, D and J genes (based on IMGT[7]) between the generated and reference repertoires and outputs and XY plot including Pearson and Spearman correlation measure.

Positional amino acid frequency

Amino acid frequency calculates positional amino acid frequency in the CDR3 of the generated sequences. Output stacked bar plots including mean MSE (mmse)[8] versus reference across all positions.

[7] IMGT, the international ImMunoGeneTics database. Lefranc et al., Nucleic Acids Research, 27(1), 1999, https://academic.oup.com/nar/article/27/1/209/1234872

[8] High-throughput antibody engineering in mammalian cells by CRISPR/Cas9-mediated homology-directed mutagenesis, Mason et al., Nucleic Acids Research, 46(14), 2018, https://academic.oup.com/nar/article/46/14/7436/5042025

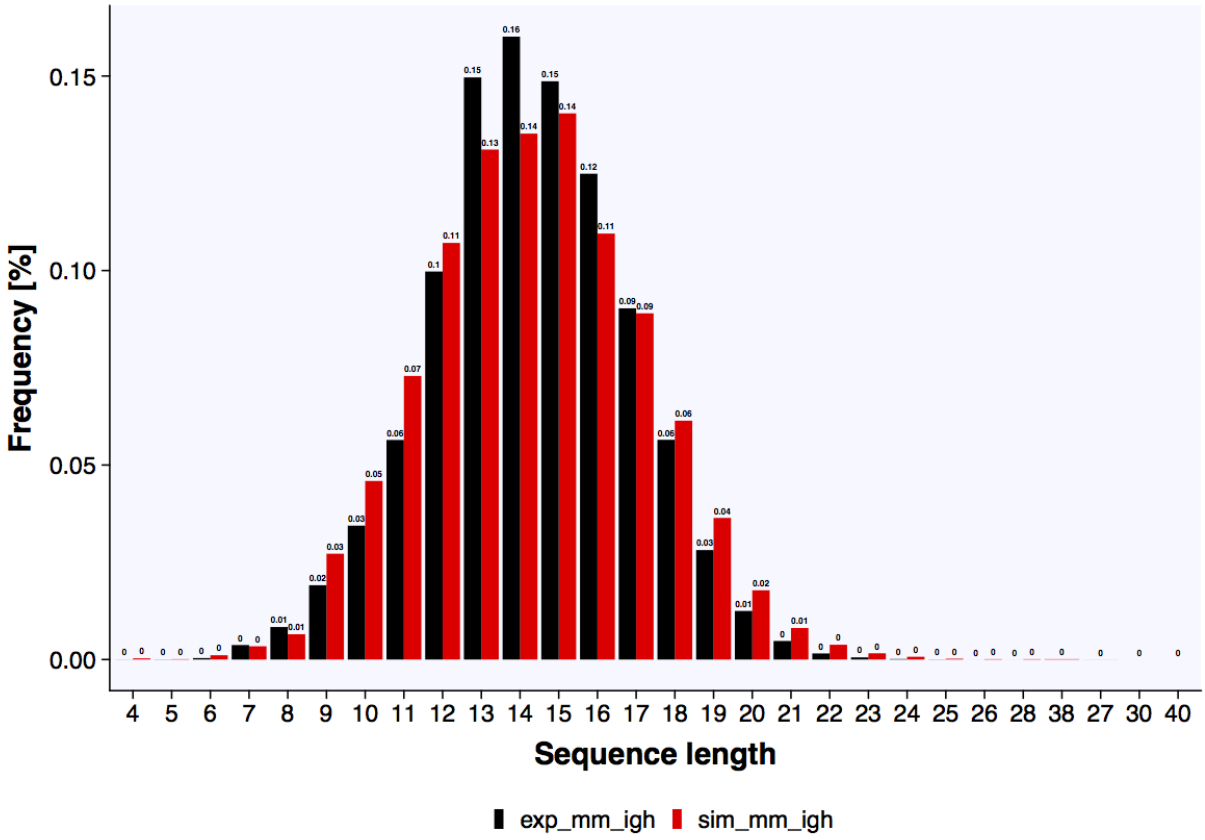


Fig. 6: The length distribution of the reference experimental and simulated datasets.

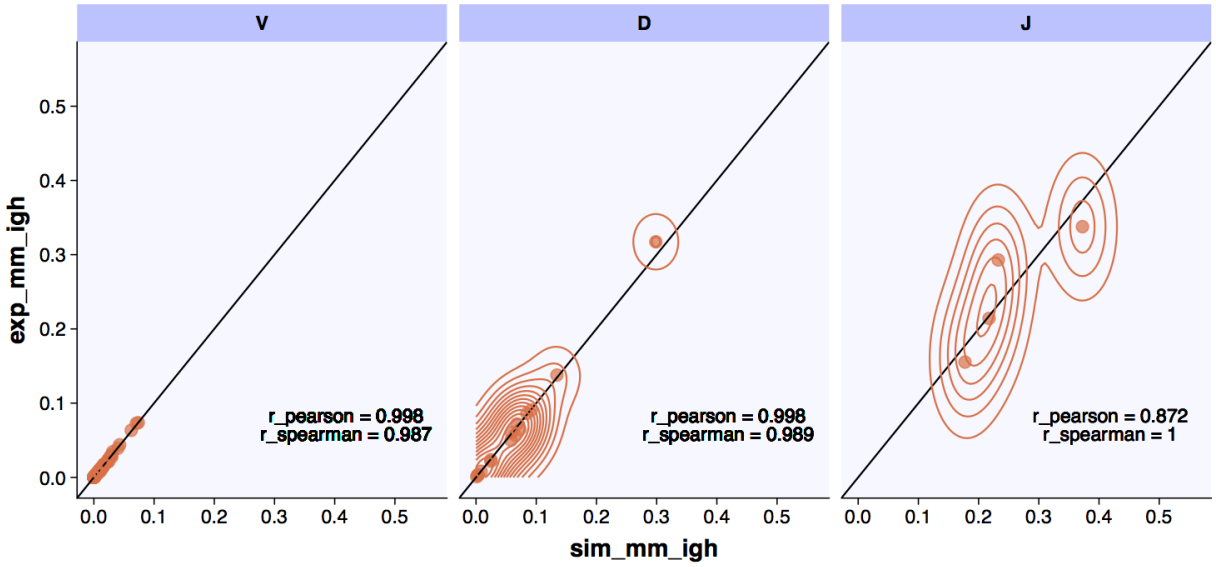


Fig. 7: The VDJ usage comparison between an experimental repertoire and a standard simulation have high correlation..

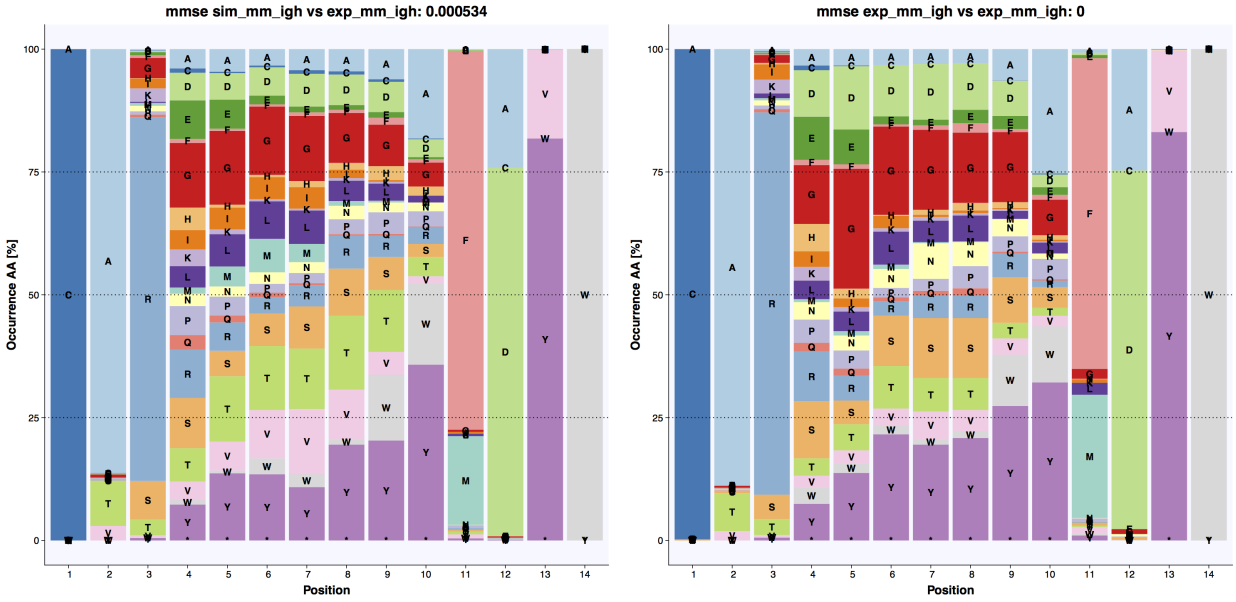


Fig. 8: The positional amino acid frequencies in the experimental and simulated datasets are highly similar.

Gapped k-mer frequency

Gapped k-mer frequency counts the occurrence of gapped k-mers in the CDR.3 of the generated sequences. Gapped k-mers are subsequences of two k-mers separated by a gap of length up to a defined variable m. For the plot generation the gapped k-mers frequencies of the nucleotide sequences for k = 3, m 3 are evaluated.

Unlike the *Positional amino acid frequency* the gapped k-mers contain sequential information allowing the user to assess the similarity of sequence patterns across simulated repertories in an XY-plot.

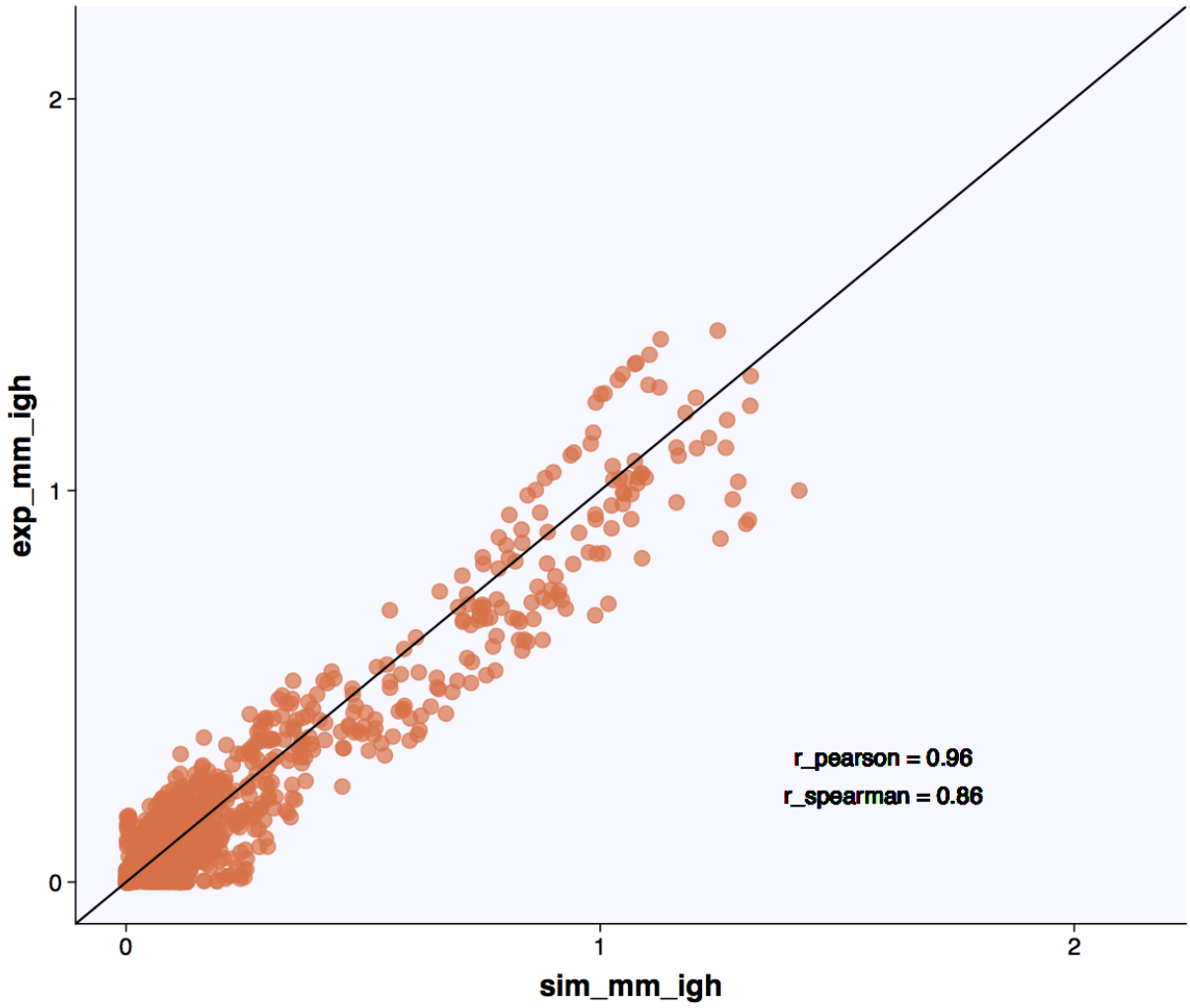


Fig. 9: Experimental (murine naive B-cell repertoire[9]) and standard simulation repertoires have high correlation with respect to the occurrence of gapped k-mer patterns.

[9] Systems Analysis Reveals High Genetic and Antigen-Driven Predetermination of Antibody Repertoires throughout B Cell Development, Greiff et al., Cell Reports, 19(7), 2017, https://www.sciencedirect.com/science/article/pii/S221112471730565X

1.7 Acknowledgments

This work is a close collaboration between the groups of Prof. Sai Reddy at ETH Zurich and the groups of Prof. Victor Greiff and Prof. Geir Kjetil Sandve at University of Oslo. In addition to all co-authors we would like to thank in particular Dr. Philippe Robert, Lonneke Scheffer, Andrei Slabodkin and Maria Chernigovskaya for package testing and feedback on the manuscript.