Tool for Immunoglobulin Genotype Elucidation via Rep-Seq (TIgGER)

Daniel Gadala-Maria

Last modified 2015-07-29

Introduction

Immunoglobulin Repertoire-Sequencing (Rep-Seq) data is currently the subject of much study. A key step in analyzing these data involves assigning the closest known V(D)J germline alleles to the (often somatically mutated) sample sequences using a tool such as IMGT/HighV-QUEST[1]. However, if the sample utilizes alleles not in the germline database used for alignment, this step will fail. Additionally, this alignment has an associated error rate of ~5%[2], notably among sequences carrying a large number of somatic mutations.

Here we provide a **T**ool for **I**mmuno**g**lobulin **G**enotype **E**lucidation via **R**ep-Seq (TIgGER). TIgGER addresses these issues by inferring the set of Ig alleles carried by an individual (including any novel alleles) and then using this set of alleles to correct the initial assignments given to sample sequences by existing tools.

Additional information is available at http://clip.med.yale.edu/tigger/ and in:

Gadala-Maria D, Yaari G, Uduman M, Kleinstein SH (2015) Automated analysis of high-throughput B cell sequencing data reveals a high frequency of novel immunoglobulin V gene segment alleles. *PNAS* 112(8):E862-70.

Input

TIgGER requires two main inputs:

1. Pre-processed Rep-Seq data
2. Database germline sequences

Rep-seq data is input as a data frame where each row represents a unique observation and and columns represent data about that observation. The required names of the required columns are provided below along with a description of each.

Column Name	Description
`SEQUENCE_IMGT`	V(D)J sequence gapped in the IMGT gapped format[3]
`V_CALL`	(Comma separated) name(s) of the nearest V allele(s)
`J_CALL`	(Comma separated) name(s) of the nearest J allele(s)
`JUNCTION_LENGTH`	Length of the junction region of the V(D)J sample

An example dataset is provided with the `tigger` package. It contains unique functional sequences assigned to IGHV1 family genes isolated from individual PGP1 (referenced in Gadala-Maria *et al.* 2015).

The database of germline sequences should be provided in FASTA format with sequences gapped according to the IMGT numbering scheme[3]. IGHV alleles in the IMGT database (build 201408-4) are provided with this package. You may read in your own fasta file using `readGermlineDb`.

```
library(tigger)
# Load example Rep-Seq data and example germline database
data(sample_db, germline_ighv)
```

Running TIgGER

The functions provided by this package can be used to perform any combination of the following:

1. Infer the presence of novel IGHV alleles not in the germline database
2. Infer the individual's IGHV genotype
3. Correct the IGHV allele calls of the samples based on the IGHV genotype

Novel Alleles

Potential novel alleles can be detected by TIgGER. Some of these may be included in the genotype later (see below). `findNovelAlleles` will return a `data.frame` with a row for each allele tested for the presence of polymorphisms. If polymorphisms are found and the novel sequence passes all quality tests (see below), then the novel allele is named and the germline sequence is included in the `data.frame`. Additionally, the result will contain metadata on the parameters used when searching for novel alleles (which can be optionally changed in `findNovelAlleles`).

```
# Detect novel alleles
novel_df = findNovelAlleles(sample_db, germline_ighv)
# Extract and view the rows that contain successful novel allele calls
novel = selectNovel(novel_df)
glimpse(novel)
```

```
## Observations: 1
## Variables:
## $ GERMLINE_CALL       (chr) "IGHV1-8*02"
## $ NOTE                (chr) "Novel allele found!"
## $ POLYMORPHISM_CALL   (chr) "IGHV1-8*02_G234T"
## $ NOVEL_IMGT          (chr) "CAGGTGCAGCTGGTGCAGTCTGGGGCT---GAGGTGAAGAA...
## $ PERFECT_MATCH_COUNT (int) 661
## $ GERMLINE_CALL_COUNT (int) 906
## $ MUT_MIN             (int) 1
## $ MUT_MAX             (int) 10
## $ GERMLINE_IMGT       (chr) "CAGGTGCAGCTGGTGCAGTCTGGGGCT---GAGGTGAAGAA...
## $ POS_MIN             (int) 1
## $ POS_MAX             (int) 312
## $ Y_INTERCEPT         (dbl) 0.125
## $ ALPHA               (dbl) 0.05
## $ MIN_SEQS            (dbl) 50
## $ J_MAX               (dbl) 0.15
## $ MIN_FRAC            (dbl) 0.75
```

The TIgGER procedure for identifying novel alleles (see citation above) involves taking all seqeunces which align to a particular germline allele and, for each position along the aligned sequences, plotting the mutation frequency at that postion as a function of the sequence-wide mutation count. While mutational hot-spots and cold-spots are both expected to have a y-intercept around zero, polymorphic positions will have a y-intercept larger than zero. The required minimum y-intercept may be specified in `findNovelAlleles` by `y_intercept`,

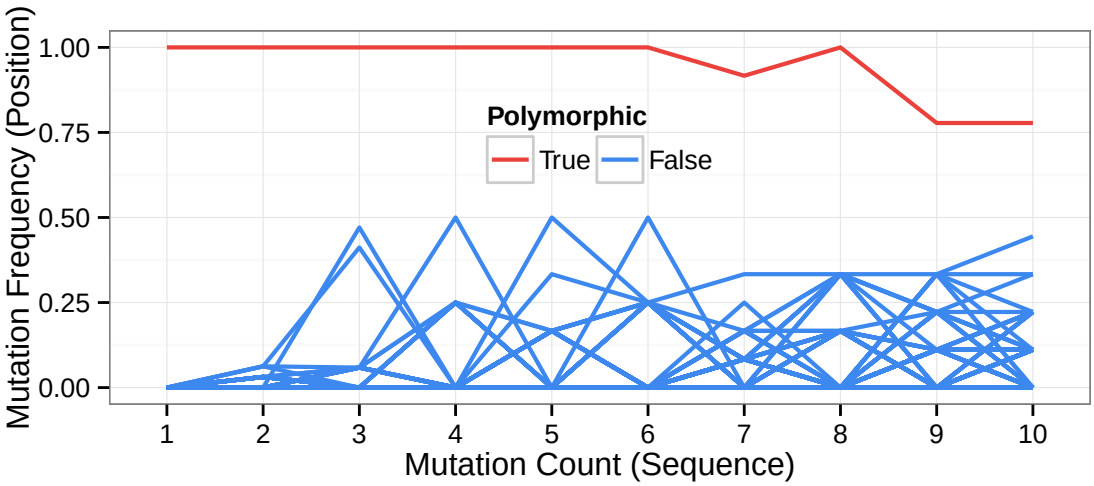
but defaults to `1/8`. Passing this y-intercept threshold is the first of three pieces of evidence that support the novel allele.

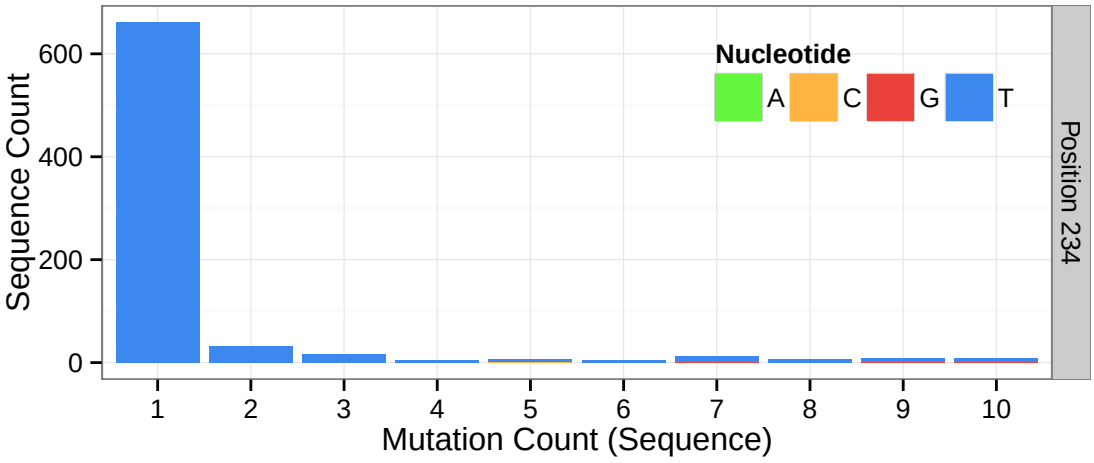
The second piece of evidence supporting novel allele calls is the nucleotide usage at the polymorphic positions as a function of sequence-wide mutation count. We expect the polymorphic allele to be prevalent at all mutation counts, and we expect the mutation count equal to the number of polymorphisms in the novel sequence to be the most prevalent.

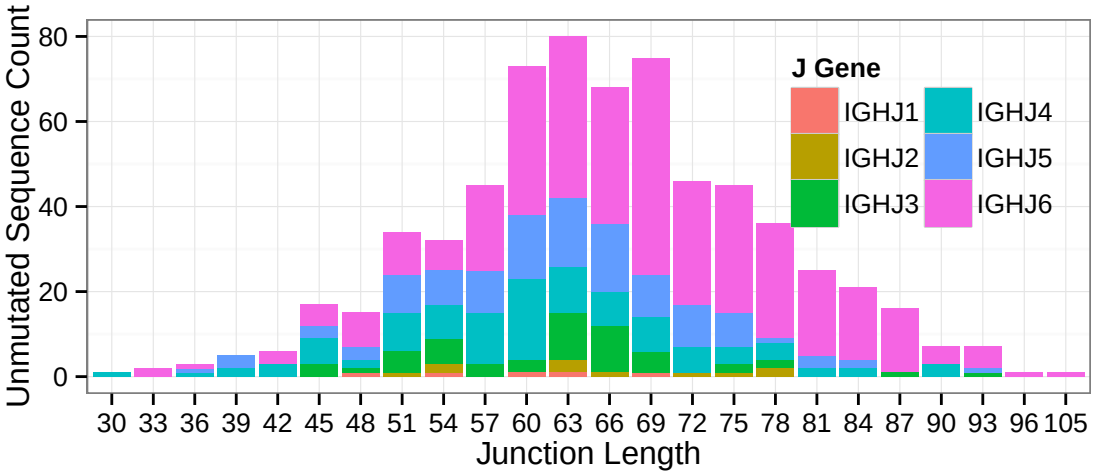
Finally, to avoid cases where a clonal expansion might lead to a false positive, combinations of J gene and junction length are examined among sequences which *perfectly match* the proposed germline allele. A true novel allele is expected to utilize a wide range of J genes, and sequences with different junction length can be ruled out as not being clonally related. The maximum portion of sequences which can consist of a specific combination of J gene and junction length may be specified in `findNovelAlleles` by `j_max`.

The three pieces of evidence desribed above can be viewed for any allele call made by `findNovelAlleles` using the function `plotTigger`.

```
# Plot evidence of the first (and only) novel allele from the example data
plotTigger(sample_db, novel[1,])
```







Genotype

An individual's genotype can be inferred using the function `inferGenotype`. This function will remove from the genotype rare/erroneous allele calls which may result from mutations in allele-differentiating regions.

This is done by determining the fewest alleles that account for nearly all (default is 7/8) of the allele calls made. The user may opt to only use sequences which perfectly match germline alleles, and may opt to include potential novel alleles. The genotype output is designed to be human readable. For each allele, the number of sequences which match the germline are listed in the same order as the alleles are listed. The total number of sequences that match any allele of that gene is also given. To output these alleles as a names vector of nucleotide sequences, the use may use the function `genotypeFasta`.

```
# Infer the individual's genotype, using only unmutated sequences and checking
# for the use of the novel alleles inferred in the earlier step.
geno = inferGenotype(sample_db, find_unmutated = TRUE,
                     germline_db = germline_ighv, novel_df = novel_df)
# Save the genotype sequences to a vector
genotype_seqs = genotypeFasta(geno, germline_ighv, novel_df)
# Visualize the genotype and sequence counts
print(geno)
```

```
##         GENE     ALLELES      COUNTS TOTAL NOTE
## 1    IGHV1-2       02,04     664,302   966
## 2    IGHV1-3          01         226   226
## 3    IGHV1-8 01,02_G234T     467,370   837
## 4   IGHV1-18          01        1005  1005
## 5   IGHV1-24          01         105   105
## 6   IGHV1-46          01         624   624
## 7   IGHV1-58       01,02       23,18    41
## 8   IGHV1-69    01,04,06 515,469,280  1279
## 9 IGHV1-69-2          01          31    31
```

Corrected Allele Calls

Finally, the original V allele calls may be limited to only those within the inferred genotype. this can be done by using the function `reassignAlleles`. By corrected the calls in this manner, the user can greatly reduce the numer of ambiguous allele calls (where a single sample sequences is assigned to multiple V alleles, thus preventing the mutations analysis of allele-differentiating positions). Additionally, assignments to erroneous not-in-genotype alleles (expected to be ~5% [2], as mentioned above, are corrected in this manner.

```
# Use the personlized genotype to determine corrected allele assignments
V_CALL_GENOTYPED = reassignAlleles(sample_db, genotype_seqs)
# Append the corrected calls to the original data.frame
sample_db = bind_cols(sample_db, V_CALL_GENOTYPED)
```

From here, one may proceed with further downstream analyses, but with the advantage of having much-improved allele calls. Besides having discovered alleles not in the IGMT database, the calls made by IMGT have been tailored to the subject's genotype, greatly reducing the number of problematic calls, as can be seen below.

```
# Find the set of alleles in the original calls that were not in the genotype
not_in_genotype = sample_db$V_CALL %>%
  strsplit(",") %>%
  unlist() %>%
  unique() %>%
  setdiff(names(genotype_seqs))
# Determine the fraction of calls that were ambigious before/after correction
```

```r
# and the fraction that contained original calls to non-genotype alleles. Note
# that by design, only genotype alleles are allowed in "after" calls.
data.frame(
  Ambiguous = c(mean(grepl(",",sample_db$V_CALL)),
  mean(grepl(",",sample_db$V_CALL_GENOTYPED))),
  NotInGenotype = c(mean(sample_db$V_CALL %in% not_in_genotype),
  mean(sample_db$V_CALL_GENOTYPED %in% not_in_genotype)),
  row.names = c("Before", "After")
  ) %>% t() %>% round(3)
```

```
##                Before After
## Ambiguous       0.112 0.006
## NotInGenotype   0.057 0.000
```

References

1. Alamyar *et al.* (2010)
2. Munshaw and Kepler (2010)
3. Lefranc *et al.* (2003)