Alakazam: Ig lineage reconstruction

Jason Anthony Vander Heiden

2016-01-28

Contents

Reconstruction of an Ig lineage requires the following steps:

1. Load a Change-O tab-delimited database file and select a clone
2. Preprocess the clone to remove gap characters and duplicate sequences
3. Run PHYLIP, parse the output, and modify the tree topology

Load Change-O data

A small example Change-O tab-delimited database file is included in the `alakazam` package. Lineage reconstruction requires the following fields (columns) to be present in the Change-O file:

- `SEQUENCE_ID`
- `SEQUENCE_IMGT`
- `CLONE`
- `GERMLINE_IMGT_D_MASK`
- `V_CALL`
- `J_CALL`
- `JUNCTION_LENGTH`

```
library(alakazam)

# Load Change-O file
file <- system.file("extdata", "changeo_demo.gz", package="alakazam")
df <- readChangeoDb(file)

# Select clone
sub_df <- subset(df, CLONE == 164)
```

Preprocess a clone

Before a lineage can be contructed the sequences must first be cleaned of gap (-, .) characters added by IMGT, duplicate sequences must be removed, and annotations must be conbined for each cluster of duplicate sequences. Optionally, "ragged" ends of sequences, such as may occur from primer template switching, may also be cleaned by masking mismatched positions and the leading and trailing ends of each sequence. The function `makeChangeoClone` is a wrapper function which combines these steps and returns a `ChangeoClone` object which may then be passed into the lineage reconstruction function.

Two arguments to `makeChangeoClone` control which annotations are retained following duplicate removal. Unique values appearing within columns given by the `text_fields` arguments will be concatenated into a single string delimited by a "," character. Values appearing within columns given by the `num_fields` arguments will be summed.

```
# This example data set does not have ragged ends
# Preprocess clone without ragged end masking (default)
clone <- makeChangeoClone(sub_df, text_fields=c("SAMPLE", "ISOTYPE"),
                          num_fields="DUPCOUNT")

# Show combined annotations
clone@data[, c("SAMPLE", "ISOTYPE", "DUPCOUNT")]
```

```
##   SAMPLE     ISOTYPE DUPCOUNT
## 1   RL02         IgA        1
## 2   RL02         IgG        1
## 3   RL02         IgG        1
## 4   RL02 IgA,IgG,IgM       61
```

Run PHYLIP

Lineage construction uses the `dnapars` (maximum parsimony) application of the PHYLIP package. The function `buildPhylipLineage` performs a number of steps to execute `dnapars`, parse its output, and modify the tree topology to meet the criteria of an Ig lineage. This function takes as input a `ChangeoClone` object output by `makeChangeoClone` and returns an igraph `graph` object. The igraph `graph` object will contain clone annotations as graph attributes, sequence annotations as vertex attributes, and mutations along edges as edge attributes.

The system call to `dnapars` requires a temporary folder to store input and output. This is created in the system temporary location (according to `base::tempfile`), and is not deleted by default (only because automatically deleting files is somewhat rude). In most cases, you will want to set `rm_temp=TRUE` to delete this folder.

```
library(igraph)
# Run PHYLIP and parse output
dnapars_exec <- "~/apps/phylip-3.69/dnapars"
graph <- buildPhylipLineage(clone, dnapars_exec, rm_temp=TRUE)
```

```
# Clone annotations
data.frame(CLONE=graph$clone,
           JUNCTION_LENGTH=graph$junc_len,
           V_GENE=graph$v_gene,
           J_GENE=graph$j_gene)
```

```
##   CLONE JUNCTION_LENGTH   V_GENE J_GENE
## 1   164              66 IGHV3-48  IGHJ2
```

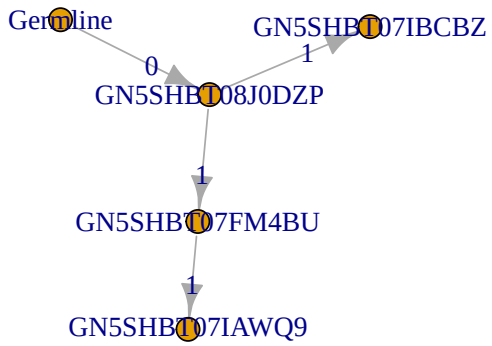
```
# Sequence annotations
data.frame(SEQUENCE_ID=V(graph)$name,
           ISOTYPE=V(graph)$ISOTYPE,
           DUPCOUNT=V(graph)$DUPCOUNT)
```

```
##      SEQUENCE_ID     ISOTYPE DUPCOUNT
## 1 GN5SHBT08J0DZP IgA,IgG,IgM       61
## 2 GN5SHBT07FM4BU         IgG        1
## 3       Germline        <NA>       NA
## 4 GN5SHBT07IAWQ9         IgG        1
## 5 GN5SHBT07IBCBZ         IgA        1
```

Plotting of the lineage tree

Plotting of a lineage tree may be done using the built-in functions of the igraph package. The default edge and vertex labels are edge weights and sequence identifiers, respectively.

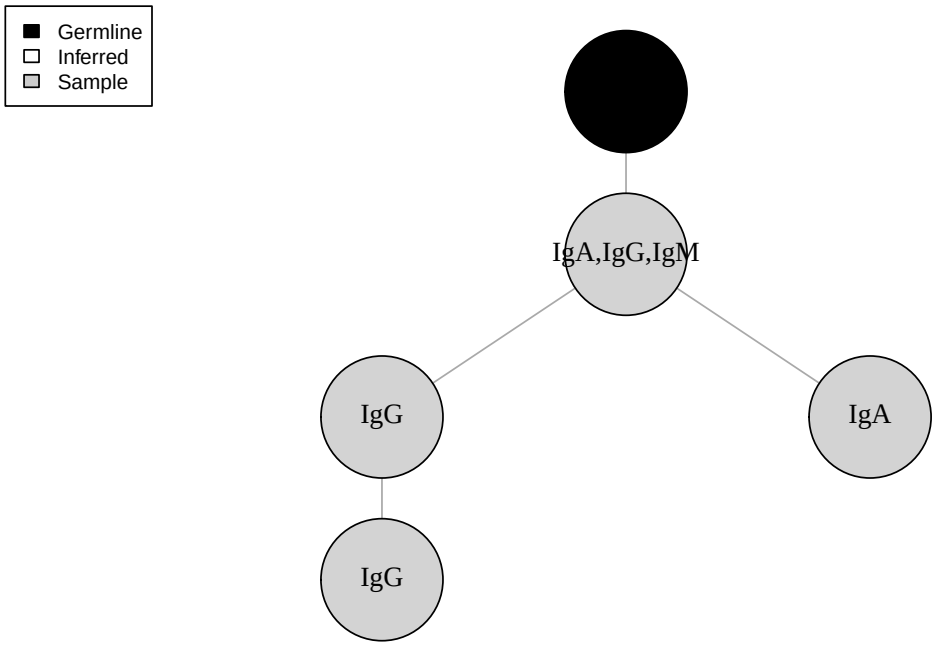
```
# Plot graph with defaults
plot(graph)
```



The default layout and attributes are not very pretty. We can modify the graphical parameter in the usual igraph ways. A tree layout can be built using the `layout_as_tree` layout with assignment of the root position to the germline sequence, which is named "Germline" in the object returned by `buildPhylipLineage`.

```
# Modify graph and plot attributes
V(graph)$color <- "lightgrey"
V(graph)$color[V(graph)$name == "Germline"] <- "black"
V(graph)$color[grepl("Inferred", V(graph)$name)] <- "white"
V(graph)$label <- V(graph)$ISOTYPE
E(graph)$label <- ""

# Remove large default margins
par(mar=c(0, 0, 0, 0) + 0.1)
# Define a tree layout with the Germline at the top
ly <- layout_as_tree(graph, root="Germline", circular=F, flip.y=T)
# Plot graph
plot(graph, layout=ly, edge.arrow.mode=0, vertex.frame.color="black",
     vertex.label.color="black", vertex.size=50)
# Add legend
legend("topleft", c("Germline", "Inferred", "Sample"),
       fill=c("black", "white", "grey80"), cex=0.75)
```



Which is much better.

Batch processing lineage trees

Multiple lineage trees may be generated at once, by splitting the Change-O data.frame on the clone column.

```
library(dplyr)

# Preprocess clones
clones <- df %>%
    group_by(CLONE) %>%
    do(CHANGEO=makeChangeoClone(., text_fields=c("SAMPLE", "ISOTYPE"),
                                num_fields="DUPCOUNT"))

# Build lineages
dnapars_exec <- "~/apps/phylip-3.69/dnapars"
graphs <- lapply(clones$CHANGEO, buildPhylipLineage,
                 dnapars_exec=dnapars_exec, rm_temp=TRUE)

# Note, clones with only a single sequence will not be processed.
# A warning will be generated and NULL will be returned by buildPhylipLineage
# These entries may be removed for clarity
graphs[sapply(graphs, is.null)] <- NULL

# Leaving a subset of clones
nrow(clones)

## [1] 204

length(graphs)

## [1] 17
```