DraftGenomeMineR

Anthony E. Melton (PostDoctoral Research Associate, Boise State University)

1/6/2021

Introduction

Genome sequencing and genomics are rapidly growing disciplines in biology. As our abilities to sequence and assemble larger, more complicated genomes, tools for analyzing genomes must also be developed. This set of R scripts, dubbed **DraftGenomeMineR**, will help users identify genes of interest, annotate scaffolds, and perform analyses, such as phylogenetic reconstructions and promoter element analyses. The motivation behind this suite of scripts was to create a *reproducible* and *easy-to-use* pipeline to facilitate analyses on draft genomes.

DraftGenomeMineR can be used to:
- Identify scaffolds that contain genes of interest
- Extract scaffolds
- Identify and annotate ORFs of genes of interest
- Assemble amino acid sequences
- Align sequences and perform phylogenetic analyses
- Summarize promoter element content and assess for differentiation between genes

Module 1: Set up the environment for genome mining

The "RequireLibraries.R" script contains a list of packages that will be installed, if needed, and load the libraries. Then, set the working directory to the "project" directory, in which all work shall be conducted.

```
## Loading required package: ape
```

```
## Loading required package: Biostrings
```

```
## Loading required package: BiocGenerics
```

```
## Loading required package: parallel
```

```
##
## Attaching package: 'BiocGenerics'
```

```
## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB
```

```
## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs

## The following objects are masked from 'package:base':
##
##     anyDuplicated, append, as.data.frame, basename, cbind, colnames,
##     dirname, do.call, duplicated, eval, evalq, Filter, Find, get, grep,
##     grepl, intersect, is.unsorted, lapply, Map, mapply, match, mget,
##     order, paste, pmax, pmax.int, pmin, pmin.int, Position, rank,
##     rbind, Reduce, rownames, sapply, setdiff, sort, table, tapply,
##     union, unique, unsplit, which, which.max, which.min

## Loading required package: S4Vectors

## Loading required package: stats4

##
## Attaching package: 'S4Vectors'

## The following object is masked from 'package:base':
##
##     expand.grid

## Loading required package: IRanges

## Loading required package: XVector

##
## Attaching package: 'Biostrings'

## The following object is masked from 'package:ape':
##
##     complement

## The following object is masked from 'package:base':
##
##     strsplit

## Loading required package: dplyr

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:Biostrings':
##
##     collapse, intersect, setdiff, setequal, union

## The following object is masked from 'package:XVector':
##
##     slice
```

```
## The following objects are masked from 'package:IRanges':
##
##     collapse, desc, intersect, setdiff, slice, union

## The following objects are masked from 'package:S4Vectors':
##
##     first, intersect, rename, setdiff, setequal, union

## The following objects are masked from 'package:BiocGenerics':
##
##     combine, intersect, setdiff, union

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## Loading required package: FastaUtils

## Loading required package: ORFik

## Loading required package: GenomicRanges

## Loading required package: GenomeInfoDb

## Loading required package: GenomicAlignments

## Loading required package: SummarizedExperiment

## Loading required package: Biobase

## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.

## Loading required package: DelayedArray

## Loading required package: matrixStats

##
## Attaching package: 'matrixStats'

## The following objects are masked from 'package:Biobase':
##
##     anyMissing, rowMedians
```

```
## The following object is masked from 'package:dplyr':
##
##     count

## Loading required package: BiocParallel

##
## Attaching package: 'DelayedArray'

## The following objects are masked from 'package:matrixStats':
##
##     colMaxs, colMins, colRanges, rowMaxs, rowMins, rowRanges

## The following objects are masked from 'package:base':
##
##     aperm, apply, rowsum

## Loading required package: Rsamtools

##
## Attaching package: 'GenomicAlignments'

## The following object is masked from 'package:dplyr':
##
##     last

## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2

## Loading required package: readr

## Loading required package: tidyr

##
## Attaching package: 'tidyr'

## The following object is masked from 'package:S4Vectors':
##
##     expand

## Loading required package: rBLAST

## Loading required package: seqinr

##
## Attaching package: 'seqinr'

## The following object is masked from 'package:matrixStats':
##
##     count
```

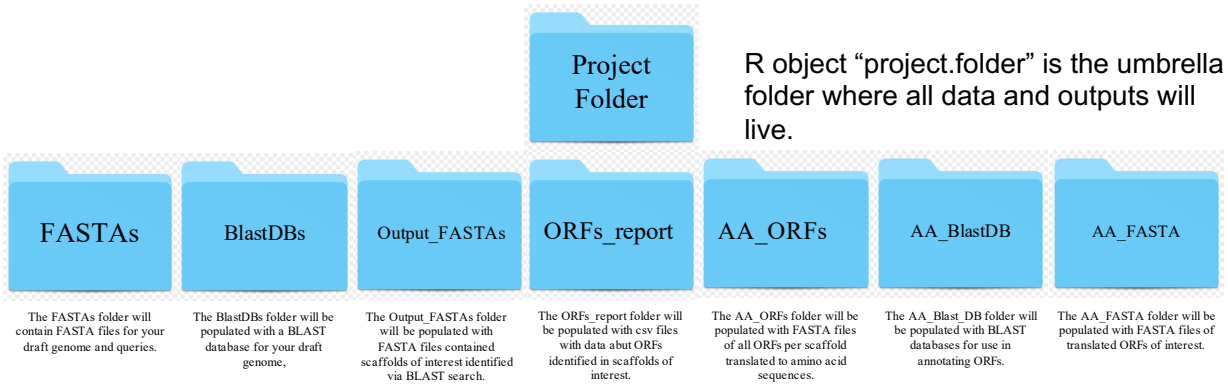
```
## The following object is masked from 'package:dplyr':
##
##     count

## The following object is masked from 'package:Biostrings':
##
##     translate

## The following objects are masked from 'package:ape':
##
##     as.alignment, consensus

## Loading required package: stringr
```

This is a diagram that shows the required folder nesting for running DraftGenomeMineR. The project folder is the highest folder, with the other fodlers nested within. This is the most basic folder requirement for the scripts - others will be added as more functionality is added into the package. Your local folders should be organized and named the same as in the diagram, as the scripts currently have this arrangement coded into them.



Module 2: Load data and perform a blast search

The following are variables that are listed in a function to perform a blast search. Some are specific files, paths to folders, and specific paramters such as blast type and thresholds. The following is an expanded version of the *DoBlastSearch* function in DraftGenomeMineR so everyone can see what's going on under the hood and better understand the process of identifying the scaffolds of interest.

```
query.file.path <- "FASTAs/pip1_4.fa"
genome.file.name <- "Artemesia_tridentata.hipmer.final_assembly.fa"
genome.path <- "FASTAs/Artemesia_tridentata.hipmer.final_assembly.fa"
blast.db.path <- "BlastDBs/Artemesia_tridentata.hipmer.final_assembly.fa"
AA.BlastDB.folder <- "~/Dropbox/Genome_PlayGround/AA_BlastDB/"
AA.ORF.folder <- "~/Dropbox/Genome_PlayGround/AA_ORFs/"
min.e <- 0.00005
perc.ident <- 100.000
query.type <- "AA"
blast.type <- "tblastn"
make.BlastDB <- T
BlastDB.type <- "prot"
```

Next, read in the draft genome to be mined. *readLines* will read in the fasta file line by line. Be aware of the return characters in your text files, as different operating systems may read these differently.

```
genome <- readLines(con = genome.path)
head(genome) # Print the top 6 lines of the fasta. There should be no spaces in what is printed. Each h
```

```
## [1] ">Scaffold0"
## [2] "CTTGAGTAGGCATTAATACTGCTAAAACCACAACGACATCACGCGTGCACCAAGTATTTGGCGCCGCTACCGTGGATATAAAAGAATTAGAGCAA
## [3] ">Scaffold2560"
## [4] "TGGGTCACTTATCCAGCAACACAATGGGATTGTCAATCCAATAGATCTATTCATAATATTCGCGCTCACCCGGAAGTGATTAATGGTTCATGATA
## [5] ">Scaffold5120"
## [6] "GATTTTACCGCAATTAAGGGTATGATTGTCTAATCATTGGGTGTCAGAGTGTCAGAGAAGCACAAAAGCACACGAATGGGCTACGTGGAAGCATG
```

Do you need to make a new blast database? Set *make.BlastDB* to T for yes, and F for no.

```
if(make.BlastDB == TRUE){
setwd("BlastDBs/")
makeblastdb(file = genome.file.name, dbtype = BlastDB.type)
}
```

Read in the query. Specify whether the query is a DNA (or RNA; these will be read the same) sequence. If not, it will assume that the query is an amino acid sequence.

```
if (query.type == "DNA") {
  query <- readDNAStringSet(filepath = query.file.path,
    format = "fasta")
} else {
  query <- readAAStringSet(filepath = query.file.path,
  format = "fasta")
}
```

Set up the blast search.

```
bl <- blast(db = blast.db.path, type = blast.type)
```

Perform the blast search.

```
cl <- predict(bl, query)
head(cl)
```

```
##                  QueryID      SubjectID Perc.Ident Alignment.Length Mismatches
## 1 Scaffold128070_PIP1-4A Scaffold128070     83.643              269          0
## 2 Scaffold128070_PIP1-4A Scaffold128070    100.000               74          0
## 3 Scaffold128070_PIP1-4A Scaffold128070     97.273              110          3
## 4 Scaffold128070_PIP1-4A Scaffold128070     92.063               63          5
## 5 Scaffold128070_PIP1-4A Scaffold128070    100.000              111          0
## 6 Scaffold128070_PIP1-4A Scaffold576976     81.959              194         18
##   Gap.Openings Q.start Q.end S.start S.end         E Bits
## 1            1       1   225    6807  6001 3.28e-161  438
## 2            0     226   299    5948  5727 3.28e-161  155
## 3            0     116   225    1976  1647  4.09e-83  214
## 4            0     223   285    1600  1412  4.09e-83  119
## 5            0       1   111    2515  2183  3.84e-67  233
## 6            1     116   292    1345  1926  3.75e-97  305
```

Filter out hits to just have unique scaffolds to extract from draft genome (no need to extract the same scaffold multiple times if it has multiple hits). There are several ways to filter: percent identity, E-value, scaffold ID...

```
cl.filt <- subset(x = cl, SubjectID == "Scaffold128070")
cl.filt.unique <- cl.filt[!duplicated(cl.filt[,c('SubjectID')]),] # SubjectID is the column that contai
cl.filt.unique
```

```
##                   QueryID      SubjectID Perc.Ident Alignment.Length Mismatches
## 1 Scaffold128070_PIP1-4A Scaffold128070     83.643              269          0
##   Gap.Openings Q.start Q.end S.start S.end         E Bits
## 1            1       1   225    6807  6001 3.28e-161  438
```

```
nrow(cl)
```

```
## [1] 192
```

```
nrow(cl.filt)
```

```
## [1] 5
```

```
nrow(cl.filt.unique)
```

```
## [1] 1
```

```
write.csv(x = cl.filt.unique, file = "Unique_Filtered_Blast_Hit_Info.csv", row.names = F)
```

Module 3: Extract scaffolds of interest identified in blast search

This chunk of code uses the output of the previous chunk, the blast search, to find and extract scaffolds of interest from draft genome and make a fasta file. This is the code for the *GetScaffolds* function.

```
cl.filt.unique <- read.csv(file = "Unique_Filtered_Blast_Hit_Info.csv") # Output of Module 1
header <- NULL # Generate empty objects to store the headers and sequences for the scaffolds of interes
seq <- NULL

for(i in 1:nrow(cl.filt.unique)){

  header[i] <- as.character(cl.filt.unique$SubjectID[i])
  seq[i] <- genome[c(grep(paste(">", cl.filt.unique$SubjectID[i], sep=''), genome)+1)]

}
```

```
## Warning in seq[i] <- genome[c(grep(paste(">", cl.filt.unique$SubjectID[i], :
## number of items to replace is not a multiple of replacement length
```

```
x <- dplyr::tibble(name = header, seq = seq) # This will assemble the headers and sequences into an obj
x
```

```
## # A tibble: 1 x 2
##   name          seq
##   <chr>         <chr>
## 1 Scaffold1280~ AAATATTATCGTGTATGACTAGATTCGAACTCGGGTTCGAGGGATTGCACACTTTACGTATGA~
```

```
writeFasta(data = x, filename = "~/Dropbox/Genome_PlayGround/Output_FASTAs/Scaffold128070.fasta")
```

Module 4: Identify ORFs in the extracted scaffolds

Find ORFs in scaffolds; This is pretty memory intense. It will write out a lot of fasta files - one for each ORF. Larger scafolds may not be able to annotated with this on computers without a lot of free hard drive space. This is the code for the *FindORFs* function.

```
scaffold <- readLines("Output_FASTAs/Scaffold128070.fasta")
scaffoldID <- grep(pattern = "^>", x = scaffold, value = T)
scaffoldID <- gsub(pattern = ">", replacement = "", x = scaffoldID)
#tryCatch(
#  {
#    for(i in 1:length(scaffoldID)){
      findORFsTranslateDNA2AA(scaffold = scaffold, scaffoldID = scaffoldID) #[i]
 #    }
#  })
```

Module 5: Annotate ORFs and write out a fasta containing the amino acid sequence for each scaffold

The next chunk of code will make data base of genes of interest to annotate ORFs. This chunk has not been condensed into function form, yet. :(

```
setwd(AA.ORF.folder)
orf.files <- list.files()
BlastDB.type <- "prot"
blast.type <- "blastp"
#setwd("AA_BlastDB/")
file.copy(orf.files, AA.BlastDB.folder)
```

```
## [1] FALSE
```

```
### All of this will need to be in a loop to loop over each scaffold, generate a db for each, and annot
#genes.seq <- readLines(con = "Scaffold18599_ORFs.fa")
setwd(AA.BlastDB.folder)
for(i in 1:length(orf.files)){
  makeblastdb(file = orf.files[i], dbtype = BlastDB.type)
}
#
```

```
# Annotate ORFs of interest and write them to their own fasta
annotated.genes.file <- "~/Dropbox/Genome_PlayGround/FASTAs/pip1_4.fa"
AA.FASTA.out.folder <- "~/Dropbox/Genome_PlayGround/AA_FASTA/"
BlastDB.type <- "prot"
blast.type <- "blastp"

setwd(AA.ORF.folder)
orf.files <- list.files()

for(i in 1:length(orf.files)){
  setwd(AA.BlastDB.folder)
  blast.db.path <- orf.files[i]
  annotated.fasta <- readAAStringSet(filepath = annotated.genes.file)
  bl <- blast(db = blast.db.path, type = blast.type)
  cl <- predict(bl, annotated.fasta) #annotated.fasta[1,]
  blast.csv.filename <- paste0(orf.files[i], "_BlastOut.csv")
  write.csv(x = cl, file = blast.csv.filename)
}

setwd(AA.ORF.folder)
genes.seq <- readLines(con = orf.files)
header <- NULL
seq <- NULL

setwd(AA.FASTA.out.folder)
for(i in 1:nrow(cl)){
  header <- as.character(cl$QueryID[i])
  seq <- genes.seq[c(grep(paste(">", cl$QueryID[i], sep=''), genes.seq)+1)]
  filename <- paste0(header, "_", as.character(cl$SubjectID[i]), ".fasta")
  x <- dplyr::tibble(name = header, seq = seq)
  writeFasta(data = x, filename = filename)
}
```

```
## Warning: The 'i' argument of ''[.tbl_df'()' must lie in [0, rows] if positive, as of tibble 3.0.0.
## Use 'NA_integer_' as row index to obtain a row full of 'NA' values.
## This warning is displayed once every 8 hours.
## Call 'lifecycle::last_warnings()' to see where this warning was generated.
```

Module 6: Assemble proteins from AA_ORF files

How to automate this? TBD.

Module 7: Use the blast results to extract promoter sequences for analyses

This function will need some love. Currently, there is one filtering step and one string modifcation step that are example specific. This is the code for the *GetPromoterSequences* function.

```
orfs.report = "ORFs_report/Scaffold128070_ORFs.csv"
blast.out = "AA_BlastDB/Scaffold128070_ORFs.fa_BlastOut.csv"
scaffold.fasta = "Output_FASTAs/Scaffold128070.fasta"
promoter.csv.file.out = "PROMOTER_OUT_TEST.csv"
promoter.sequence.fasta = "PROMOTER_OUT_TEST.fa"

orf.blast.out <- read.csv(blast.out)
orf.blast.out.filt <- filter(orf.blast.out, orf.blast.out$Perc.Ident == 100.000)
orf.blast.out.filt.keep <- str_remove_all(string = orf.blast.out.filt$SubjectID, pattern = "Scaffold128
#

#
csv.full <- read.csv(file = orfs.report, sep = " ")
#csv <- csv.full[csv.full$ORFID == orf.blast.out.filt.keep]
csvsub <- csv.full %>%
  filter(csv.full$ORFID %in% orf.blast.out.filt.keep)
csv <- csvsub[which(as.numeric(csvsub$start) >= 1500),]

#

#Create data frame and populate it
Promoters <- data.frame(matrix(ncol=6, nrow=nrow(csv)))
colnames(Promoters) <- c("ORFid", "ScaffoldID","Strand", "Start","End", "Sequence")

Promoters$ORFid <- csv$ORFID
Promoters$End <- csv$start
Promoters$ScaffoldID <- as.vector(csv$scaffoldID)
Promoters$Strand <- as.vector(csv$strand)
Promoters$Start <- as.numeric(Promoters$End) - 1500
Promoters$Start[Promoters$start < 0] <- 1

#Read FASTA file (line by line)
scaffold <- readLines(scaffold.fasta)

#Extract sequences to be mined for promoter sequences using PALACE
for(i in 1:nrow(Promoters)){
  #Add DNA sequence adapted to strand
  #Extract seq from FASTA file
  seqRaw <- scaffold[c(grep(paste0(">", Promoters$ScaffoldID[i]), scaffold)+1)]

  #Package sequence and extract start and end
  if(Promoters$Strand[i] == "+"){
    Promoters$Sequence[i] <- paste(strsplit(seqRaw, split='')[[1]][as.numeric(Promoters$Start[i]):as.nu
  }
  if(Promoters$Strand[i] == "-"){
    revComp <- as.vector(reverseComplement(DNAStringSet(seqRaw)))
    Promoters$Sequence[i] <- paste(strsplit(revComp, split='')[[1]][as.numeric(Promoters$Start[i]):as.n
  }
}
#
#

#
```

```r
#Promoters$Start
#

#
write.csv(Promoters, promoter.csv.file.out, row.names = F, quote = F)
#

#
csv <- Promoters
#csv
seqType <- "DNA"
FASTA <- NULL
for(i in 1:nrow(csv)){
  if(seqType == "DNA"){
    DNA <- paste(paste(">", csv$ScaffoldID[i], "_", csv$Gene_hypothesis[i], sep=''), csv$Sequence[i], s
    FASTA <- rbind(FASTA, DNA)
  }
}
#

#
#FASTA
write.table(FASTA, file = promoter.sequence.fasta, col.names = F, row.names = F, quote = F)
#

head(read.csv("~/Dropbox/Genome_PlayGround/PROMOTER_OUT_TEST.csv"))
```

```
##    ORFid     ScaffoldID Strand Start  End
## 1 ORF_12 Scaffold128070      -  3742 5242
## 2 ORF_21 Scaffold128070      -   309 1809
##
## 1 TACCCTTTCATTTTTCATCTTCGTTAAGTTTTTGATTTAAATACTATCTATTCATAATGTATCCTTGAACATAAAATACCGCCAAAGTTTCCGCGTTC
## 2 CACATATATTATAGATTCCATATATCATTTGCAAAACTTACGAGGCATGTGAGTCACCATCACACAATTCCAAAGATAGTCTATCTAACCTTGTTCCT
```