Finding Paralog Targets for Neglected Diseases

By

Jeremy Singer

# Abstract

This paper describes a method that can be used to discover and repurpose existing drugs and drug targets by discovering cross species genomic sequence similarities. It uses public domain databases (ChEMBL, EupathDB) and open source software to find measures of sequence similarity with existing targets.

This method can be applied to pathogens with at least a medium sized genome (several thousand genes.) *Neglected tropical diseases* caused by pathogenic protists are good subjects for this approach because many have genomes of sufficient size and because many have genomic features in common with organisms for which there are known targets.

The genome of the apicomplexan parasite *Plasmodium falciparum*, which is responsible for the most virulent form of malaria, was chosen to validate a method that identifies paralogs to existing disease targets because it has known cross-species targets.

ChEMBL provides a PostgreSQL database that contains a list of thousands of targets and target protein sequences as well as ligands for those targets. Using this database and open source software, this paper identified *[<number>]* distinct drugs and *[<number>]* targets validating this approach.

Seven other pathogens (*Plasmodium vivax, Toxoplasma gondii, Trypanosoma brucei, Trypanosoma cruzi, Leishmania major, and Entamoeba )* were also downloaded and run through the same pipeline, identifying potential targets and drugs.

# Introduction

*Neglected Tropical Diseases* are those diseases that affect tropical areas underserved for health care due to the poverty of those areas. These diseases affect over a billion people in over 149 countries, and damage the economies of these areas at a cost of many billions of dollars[1, p. 1].

Repurposing drugs and generating leads for finding new drugs by repurposing targets could be a cost -effective way for combating these diseases. Finding new targets can be difficult, as it requires understanding many specific details for each pathogen. A systematic method of discovering new targets that does not require this specific understanding can reduce the cost and effort of finding these targets.

This paper describes a method for *Drug Repurposing* and *Target Repurposing* based on discovering similarities between existing targets and pathogen genomes.

ChEMBL provides a downloadable database that includes drug targets and drug information for those targets, as well as amino acid sequences of the protein targets[[1]](#footnote-1). Drug targets tend to be proteins that are important enough to the organism to which they belong that they tend to be conserved. If we can find a protein sequence in a disease organism that is sufficiently similar to a known target, the protein may be a promising target in that organism, and drugs used against that target may be successfully used in that organism.

The analysis pipeline uses **BLASTP** [[2]](#footnote-2) or **jackhmmer** [4] to produce similarity reports, parse the results, and upload to supplementary tables in the PostgreSQL database.

This analysis pipeline was first applied to the genome of *Plasmodium falciparum* using both BLASTP and HMMER to generate similarity statistics, and custom scripts included in the Appendix. The scores returned from these two different programs were compared to evaluate which could provide better discrimination criteria of useful targets and drugs.

*Plasmodium falciparum* was chosen for this evaluation because it is the most significant of these neglected diseases. In 2018, there were over 228 million cases of malaria worldwide, causing over 408 thousand deaths.[5] Emerging drug resistance to existing drugs such as choloroquin and sulfadoxine-pyrimethamine, as well as quinine increase demand for new drugs that are more effective.[6],[[3]](#footnote-3)

Database queries identify promising targets and drugs according to criteria developed and implemented in R.

In addition to *p. falciparum*, we processed the following additional pathogens using [*preferred method]*:

[*pathogen list, see abstract].* The statistics were loaded into supplementary tables in the PostgreSQL database.

Queries using the existing ChEMBL\_25 database, in combination with these similarity statistics were used to identify candidate targets and drugs for each of these pathogens.

# Materials and Methods



Figure : Processing overview

PlasmoDB and Chembl provide the data that are analyzed in this process flow.

Amino acid sequences of putative *Open Reading Frames* (ORFs) for *Plasmodium falciparum 3D7* were downloaded from *PlasmoDB.org* as file**PlasmoDB-46\_Pfalciparum3D7\_ORFs\_AA.fasta**.[8],[9]

The FASTA formatted dataset consists of all ORFs in a single file. Each ORF consists of a header line followed by a number of lines containing multiple characters of single letter codes representing an amino acid.

Header lines are formatted according to two different patterns. The first pattern encodes the ORF id that is comprised of the organism code, chromosome, and identifier. The second pattern contains a type identifier that identifies the record as belonging to the mitochondrion, and contains a unique identifier for the ORF id. A script fans out the ORF records into individual files in a directory structure having a separate subdirectory structure for each chromosome. (See script 7.2.1. fan\_out\_fasta.R ).



Figure : ORF header structure determines fan out destination

The set of target sequences comes from the ChEMBL\_25 PostgreSQL database and downloaded by a *psql* script (See 7.1.1. chembl\_25\_targets.sql) as file **chembl\_targets.txt**.

These targets are converted by a Perl script (See 7.1.2. split\_to\_fasta.pl) to FASTA formatted sequences (See 7.1.2. split\_to\_fasta.pl) creating file **component\_sequences.fa**.

## Gathering BLASTP statistics

The **makeblastdb** utility converts the FASTA formatted file of targets to a **BLASTP** searchable database in the **~/genomes/blast\_targets** directory.

A script applies the **BLASTP** utility to each of the ORFs for the *P. falciparum* organism against the BLAST database (see 7.2.2. do\_all\_blast.sh).

A *bash* script ( 7.2.5. do\_all\_blast\_stats.sh) applies a Perl script (7.2.3. extract\_header.pl) that parses each BLAST report into a *.stats* file for each ORF.

A bash script ( 7.2.4. make\_blast\_statistics.sh ) consolidates all the <ORF>.*blastp.txt.stats* files into a tab delimited text file, **blast\_statistics.txt**.

The **blast\_statistics** table, which has been created previously (7.2.6. create\_blast\_statistics\_tbl.sql), is populated by SQL script (7.2.7. import\_p\_falciparum.sql).

## Gathering HMMER statistics

A script applies the **jackhammer** utility to each of the ORFs of the *p. falciparum* organism, generating a report and a summary file for each (see 7.3.1. do\_all\_jackhmmer.sh).

A *bash* script ( 7.3.3. do\_all\_hmmer\_stats.sh) applies a Perl script ( 7.3.2. extract\_hmm\_summary.pl) that extracts the **jackhmmer** statistics from the reports and summaries and produces a consolidated tab delimited file for all ORFs called **hmm\_stats.txt**.

An import SQL script (7.3.5. import\_hmmer\_statistics.sql) imports these statistics into previously created tables (see 7.3.4. create\_hmmer\_stats\_tbls.sql).

Consolidated statistics records having the same ORF/target were downloaded using this join:

# Discussion and Results

The “targetness” of a protein has to do with how indispensable its function is to the disease organism. This description of the nature of protein targets suggests that paralogous proteins in our organism of interest could also be targets, if they are sufficiently similar to existing targets. The closeness of the match will suggest that the function of the protein has been conserved between the previously identified target organism and our organism of interest. Those sequences in the pathogen organism which are most necessary for its survival are also least likely to change, as mutation would tend to impair functions necessary for survival. At the same time, we are searching exactly for those critically necessary proteins as targets for drugs that can impair them.

To find likely targets in the genome, we need to measure similarity between ORFs from its genome and our target database. When we have computed these similarities, we need to chose threshold criteria for filtering the most promising candidates.

Both **BLASTP** and **jackhmmer** score similarity between amino acid sequences by aligning query and target sequences. Their approach to scoring, however, differs. **BLASTP** detects similarities by scoring the likelihood of successive letters of the amino acid being the same in query and target sequences in comparison to chance. It uses transition matrices based on conserved sequences in different genomes to assign the likelihood that that a particular amino acid will follow another. By contrast, **jackhmmer** uses hidden Markoff models (HMM) that assess patterns by looking for larger domains. We might therefore expect that this approach might be more discriminating, as it takes into account similarities at the level of protein domains. In effect, the **BLASTP** approach uses the limited case of a hidden Markoff model of length two, and then sums the scores, adding for more likely matches, zero for matches for no match, and negative numbers for mismatches. Gaps also result in penalties.



Figure : Understanding BLAST statistics

Each match and alignment contains various scores which BLASTP computed for it.

The “Expect” statistic measures the probability that the matches could occur by chance. For example: if we assume equal probability for the occurrence of a particular amino acid in a sequence (.05), this is the expected value for a random match of this length sequence. This is much too high an expected value to accept this as a non-random match. Results with an expected value >= .001 were filtered out.

The ratio of exact matches to the length of the *subject* gives the *identities* statistics; The ratio of conserved matches (including identities and functionally similar amino acids) to the total number of characters in the *subject* gives the *positives* statistics. In the same way, gaps are tabulated and compared with the total subject length. The score statistic gives BLASTP’s estimate of the query and target sequences similarity by accumulating the positive matches weighted by the match length minus gap penalties.

Figure : ORF.FASTA.summary from jackhmmer

The summary for a query may hit multiple targets. Each target record is repeated for each domain that **jackhmmer** matches. We are only interested in the total score for the full sequence. The Perl script (7.3.2. extract\_hmm\_summary.pl) creates a single record per ORF/target by de-duping these values. Score is used in a similar way to BLAST, but is computed differently.

## Choosing appropriate metrics for selecting target candidates

**BLASTP** and **jackhmmer** also compute *expected* values, which can be very small numbers. In contrast, the *scores* values are always integers that are easily comparable.

Other statistics computed by BLASTP are *identities*, which are the percentage of exact matches, and *positives*, which are inexact matches that conserve function because the amino acids involved in the comparison function in a compatible manner to each other in the protein.



Figure : Pairwise comparison of BLAST metrics

An advantage of the *score* statistic is that it is additive. While both statistics reflect the cumulative values for matches, products of probability scores that accumulate in the *expect* statistic can lose resolution during computation. Although *expect* should follow (inversely) to score, the comparison here is uninformative because of loss of resolution at low values. *Positives* trend like *identities* but show *greater than or equal*  relationship to them. *Score* also trends with *positives,*  showing greater detail.

Differences between the comparison of *scores* by *identities* and *expect* by *identities* varies substantially due to the imprecision caused by computations using *expect.*

## Comparison of BLASTP and jackhmmer scores

The default inclusion threshold for **jackhmmer** is much more stringent than the default threshold for **BLASTP**. The **blast\_statistics** table imported 562,039 records, where the **hmmer\_statistics** table contains only 127,306 records.

A query shows that there are 20,178 **blast\_statistics** records that join with the **hmmer\_statistics** records with the same OFR/target. A tab delimited file named **consolidated\_stats.txt** was downloaded with these records (see 6.4.1. consolidated\_orf\_target.sql).



Figure : BLASTP vs HMM scoring comparison

The figure shows that the two kinds of scores generally trend the same, but there are many excursions for *hmm\_score* that are much greater than the *blast score* would lead one to expect. (See 6.4.2. compare\_scores.R) Thismay be attributed to **jackhmmers’**s increased sensitivity to structure over **BLASTP**.

The scale of *hmmer\_score*to *blast\_score* is **0.5523**.We can use this later to estimate the appropriate selection threshold for *hmmer* statistics.

While the **jackhmmer** statistics may provide us with a more accurate scoring of the similarity of parasite ORFs with our target universe, we can use the broader **BLASTP** statistics to provide statistical guidelines for selecting a significance threshold. By pairing these two sets of statistics, *blast statistics* can help us inform useful methods for selection from *jackhmmer statistics*.

## How departures from normality can reveal potential targets



Figure : Histogram of log(score) for P. falciparum 3D7

Histogram of log BLASTP scores, showing superimposed normal plot in red.

Vertical blue lines show thresholds that are 1, 2, 3, 4, and 5 median absolute deviations from the median.

Normal curve was calculated using **R dnorm** function, which expects a standard deviation parameter, corrected here by a factor using **mad** function. Log scale was chosen to improve display of the curve.

Choosing a match metric and threshold is a conundrum. What determines whether a match is sufficiently close to suggest that it is likely to be a target?

While a very close match of the *positives* or *identities* (100% or 99%) would indicate that a match identifies a target by tautology, lower scores are hard to justify. A small sequence with a fairly high percentage may still not be as likely a match as a much larger string with a lower *positives* score and a much higher *score* statistic, as the cumulative score takes into account the greater difficulty of achieving a high score for the larger string.

The distribution of log(*score)* for all the matches in *p. falciparum*  appears to be somewhat normal.

There are several reasons why we should expect these results.

1. The distribution is not symmetric around the mean because BLASTP discards matches where the score is too low, and only returns up to the ten highest scores per query.
2. Normally distributed matches are what we would expect due to random mutation for non-conserved sequences, and account for the normal appearance of the graph.
3. Due to conservation of essential peptides, there may be more high scoring matches than predicted by normally distributed processes. We can look for these likely target candidates in the area of the distribution which normally should be approaching zero asymptotically.
4. In addition, the target universe may be “lumpy” in the sense that there may be more than one centroid that has a family of similarities in the *p. falciparum* genome. This “lumpiness” is not important for purposes of finding unusual target similarity, as it is buried in a normal cohort.

It is interesting to note that the smallest *score* value for a comparison that has a 100% positives value is 21.6, which is close to the median of 23.9 ( or log values of 3.07 and 3.17 respectively). Searching the upper tail means missing some targets that are closer to the median.

Median and Median Absolute Deviation were used as measures of centrality and dispersion rather than mean and standard deviation because these measures are more appropriate considering the asymmetry and contamination of the distribution (in the sense that the distribution contains more than a single normal distribution plus other non-normally distributed data.)



Figure :QQ Plot for all P. falciparum BLASTP log(scores)

The red line shows expected normality. This plot shows departure from normality at the right (upper) portion of the graph.



Figure : normal subset of log BLAST stats for p. falciparum

We chose a threshold of 2 *maximum absolute deviations* above the median as a cutoff below which the similarity results are *normal* and therefore *insignificant.* The normal score threshold was 27.16, and the log score threshold was 3.3073. We expect that scores above that will be more than randomly significant.

Applying the Q-Q plot to the *hmmer\_score* values does not provide as neat an answer:



Figure : HMMER scores are less well behaved

The Q-Q plot shows a pronounced *jink* in the middle of the graph. This suggests that the HMMER scores may be easier to stratify into different similarity classes.

## Using kmeans to find a significant cluster

Another approach to find a useful threshold is **kmeans** clustering. We are looking for a region that is less dense, indicating unusually high similarity.



Figure : kmeans clustering of hmmer\_score

The purple line shows the threshold separating less similar insignificant similarity (in red) from highly similar ORFs in black. The similarity threshold found by kmeans is more stringent than the one we found using *median* and *maximum absolute deviation*.

1708 scores out of 20178 belong to the significant cluster.

## Downloading drugs and targets

Having determined a significance threshold, we are ready to select drugs and targets from the *CHEMBL\_25* database. The query specifies tax\_id to find targets for the associated organism.

The score in this query was determined from kmeans, as described (See 6.4.3. Score normality and kmeans analysis).

select distinct h.target, score, td.tax\_id as original\_tax\_id, td.organism as orig\_organism,td.pref\_name, md.pref\_name, md.chembl\_id

from hmmer\_statistics h

join target\_dictionary td

on h.target = td.chembl\_id

join drug\_mechanism dm

ON dm.tid = td.tid

join molecule\_dictionary md

ON dm.molregno = md.molregno

WHERE h.tax\_id = 36329

and score >= 385.3

order by score desc;

# References

[1] “WHO | World Health Organization,” *WHO*. [Online]. Available: http://www.who.int/neglected\_diseases/diseases/en/. [Accessed: 12-Feb-2020].

[2] A. Gaulton *et al.*, “The ChEMBL database in 2017,” *Nucleic Acids Res.*, vol. 45, no. D1, pp. D945–D954, Jan. 2017, doi: 10.1093/nar/gkw1074.

[3] S. F. Altschul *et al.*, “Gapped BLAST and PSI-BLAST: a new generation of protein database search programs,” *Nucleic Acids Res.*, vol. 25, no. 17, pp. 3389–3402, Sep. 1997, doi: 10.1093/nar/25.17.3389.

[4] T. J. Wheeler and S. R. Eddy, “nhmmer: DNA homology search with profile HMMs,” *Bioinformatics*, vol. 29, no. 19, pp. 2487–2489, Oct. 2013, doi: 10.1093/bioinformatics/btt403.

[5] “World malaria report 2019.” [Online]. Available: https://www.who.int/news-room/feature-stories/detail/world-malaria-report-2019. [Accessed: 12-Feb-2020].

[6] M. A. Khan, R. A. Smego, S. T. Razi, and M. A. Beg, “Emerging drug--resistance and guidelines for treatment of malaria,” *J. Coll. Physicians Surg.--Pak. JCPSP*, vol. 14, no. 5, pp. 319–324, May 2004, doi: 05.2004/JCPSP.319324.

[7] “WHO | Responding to antimalarial drug resistance,” *WHO*. [Online]. Available: http://www.who.int/malaria/areas/drug\_resistance/overview/en/. [Accessed: 12-Feb-2020].

[8] C. Aurrecoechea *et al.*, “PlasmoDB: a functional genomic database for malaria parasites,” *Nucleic Acids Res.*, vol. 37, no. Database issue, pp. D539-543, Jan. 2009, doi: 10.1093/nar/gkn814.

[9] “PlasmoDB Download Files.” [Online]. Available: https://plasmodb.org/common/downloads/Current\_Release/Pfalciparum3D7/fasta/data/. [Accessed: 12-Feb-2020].

# Appendix: Scripts

These scripts can be found in the **supplements** directory for this **git** repository.

## BLAST Targets

### chembl\_25\_targets.sql

Run this script at the command line of psql attached as chembl\_25 in the blast\_targets directory.

This script creates **chembl\_targets.txt** file.

\copy (select td.chembl\_id, cs.sequence from target\_dictionary td join target\_components tc on td.tid = tc.tid join component\_sequences cs on tc.targcomp\_id=cs.component\_id) to chembl\_targets.txt

### split\_to\_fasta.pl

Run this script from the *bash* command line in the **blast\_targets** directory: **perl split\_to\_fasta.pl**

This script creates the **component\_sequences.fa**file which can be found in the **supplements**/**blast\_targets** directory.

#######################################

# split\_to\_fasta.pl

# input recs: <key><delim><sequence>

# output : rec1 = ><key>

# rec2 = <sequence>

#######################################

my $infile = 'chembl\_targets.txt';

my $outfile = 'component\_sequences.fa';

my $delim = '\t';

open(IN, $infile) or die("Unable to open $infile\n");

my @lines = <IN>;

close(IN);

open(OUT,">",$outfile) or die ("Unable to open $outfile\n");

foreach my $line(@lines)

{

my @rec = split($delim,$line);

if (scalar(@rec) > 1)

{

print OUT ">$rec[0]\n";

print OUT "$rec[1]\n";

}

}

close(OUT);

exit(0);

## Process FASTA

### fan\_out\_fasta.R

This script is run within RStudio to fan out the single FASTA file from *Plasmodb* into separate directories by organism and chromosome. Each ORF is separated out for ease of obtaining BLAST and HMMER reports for each.

# Fan out AA\_fasta file from plasmodbc

# based on the structure of Plasmodium AA\_orf files.

# FASTA headers come in two varieties:

# 1. >Pf3D7\_01\_v3-1-60871-61059 | organism=Plasmodium\_falciparum\_3D7 | location=Pf3D7\_01\_v3:60871-61059(+) | length=63 | sequence\_SO=chromosome

# ^unique ORF identifier---^ <other stuff> <sequence\_SO=<ORF type> i.e. chromosome, apicoplast, mitochondrial

# ^head indicator

# ^organism

# ^chromosome id

# ^orf\_name

# 2. >Pf\_M76611-5-344-75 | organism=Plasmodium\_falciparum\_3D7 | location=Pf\_M76611:75-344(-) | length=90 | sequence\_SO=mitochondrial\_chromosome

# ^orfname---------^ <other stuff> sequence\_SO=mitochondrial\_chromosome

# parsing strategy is: for non-mitochondrial, parse out chromosome\_name, orf\_name.

# For mitochondrial, orfname is one piece.

library(stringr)

setwd('~/genomes')

aa\_file=file.choose()

aa=read.table(file=aa\_file,header = FALSE, sep='~', stringsAsFactors = FALSE)

aa=aa[!is.na(aa[,1]),] # filter out NA

firstrec=aa[1] # scalar

aa=data.frame(lines=aa, stringsAsFactors = FALSE)

parsed=strsplit(firstrec,'\_')

organism\_pref=substring(parsed[[1]][1],2)

# make a directory for this organism

system(paste('mkdir',organism\_pref))

orf\_headers=aa[substr(aa[,1],1,1)=='>' ,]

mi\_headers=orf\_headers[grep('sequence\_SO=mitochondrial\_chromosome',orf\_headers)]

chrom\_headers = setdiff(orf\_headers, mi\_headers)

parsed=strsplit(chrom\_headers,'\_')

chromosomes=unique(sapply(parsed,function(p){p[2]}))

# make a directory for each chromosome

for (chromosome in chromosomes){

dirname=paste(organism\_pref,chromosome,sep='/')

system(paste('mkdir',dirname))

}

dirname=paste(organism\_pref,'mitochondrion', sep='/')

system(paste('mkdir',dirname))

orf.df = data.frame(line='')

orf\_name=''

orf.df=data.frame(line='')

for(orf\_line in aa[,1]){

print(paste('orf\_line: ',orf\_line))

if (substr(orf\_line,1,1)=='>'){

print('FASTA header line')

if ( is.na(orf\_name) || nchar(orf\_name) > 0){

orf\_name=paste0(orf\_name,'.FASTA')

print("write statement")

write\_dir\_name = paste(organism\_pref,chromosome, orf\_name,sep='/')

write.table(orf.df, file=write\_dir\_name,row.names = FALSE,col.names = FALSE, quote=FALSE)

}

orf.df = data.frame(line=orf\_line)

if ( length(grep('mitochondrial',orf\_line)) > 0){

chromosome='mitochondrion'

print(paste("Chromosome:", chromosome))

parsed=unlist(strsplit(orf\_line,' '))

orf\_name=substr(parsed[1],2,nchar(parsed[1]) -1)

} else {

parsed=unlist(strsplit(orf\_line,' '))

parsed=unlist(strsplit(parsed[1],'\_'))

chromosome=parsed[2]

orf\_name=parsed[3]

}

print(paste('chromosome:',chromosome,', orf\_name:', orf\_name))

} else {

print('rbind FASTA sequence')

orf\_line.df=data.frame(line=orf\_line)

orf.df = rbind(orf.df, orf\_line.df);

}

}

if (is.na(orf\_name) || nchar(orf\_name) > 0){

print("write statement")

write\_dir\_name = paste(organism\_pref,chromosome, orf\_name,sep='/')

write.table(orf.df, file=write\_dir\_name,row.names = FALSE,col.names = FALSE,quote=FALSE)

}

### do\_all\_blast.sh

Run this script in the genome directory.

Specify the *Organism\_dir* on the command line.

#!/bin/bash

if [ -z $1 ]

then

while [ -z $org\_dir ]

do

read -p "Organism directory: " -a org\_dir

done

else

org\_dir=$1

fi

echo $org\_dir

for chrom\_dir in $( ls -d $org\_dir\*/ );do

for orf in $( ls $chrom\_dir\*.FASTA);do

echo "BLASTP " $orf

blastp -db ~/blast\_targets/chembl\_25\_targets -query $orf -num\_alignments 10 -out ${orf}.blastp.txt

done

done

### extract\_header.pl

This Perl script extracts statistics from BLAST reports.

use Switch 'fallthough';

my @lines = <STDIN>;

my $phase = 0;

my @rec = {};

my $rec\_string;

my %recs = ();

my $query;

foreach my $line(@lines)

{

switch($phase){

case 0 {

if ( $line =~ m/Query=\s\*(\S+)/)

{

$query = $1;

}

if ( $line =~ m/\>\s\*(\S+)/) # orf id

{

$phase = 1;

$rec[0] = $1;

}

}

case 1 {

if ( $line =~ m/Length\=(\S+)/ ){

$rec[scalar(@rec)] = $1;

$phase = 2;

}

}

case 2 {

if ( $line =~ m/Score\s\=\s(\S+)/){

$rec[scalar(@rec)] = $1;

$line =~ m/Expect\s\=\s(\S+),/;

$rec[scalar(@rec)] = $1;

$phase = 3;

}

}

case 3 {

if ( $line =~ m/Identities\s\=\s\S+\s\((\S+)\%/){

$rec[scalar(@rec)] = $1;

$line =~ m/Positives\s\=\s\S+\s\((\S+)\%/;

$rec[scalar(@rec)] = $1;

$line =~ m/Gaps\s\=\s\S+\s\((\S+)\%\)/;

$rec[scalar(@rec)] = $1;

$rec\_string = join("\t",@rec);

$recs{$rec\_string} = 1;

$phase = 0;

@rec = {};

}

}

}

}

foreach my $record(keys %recs){

print "$query\t$record\n";

}

### make\_blast\_statistics.sh

Create the blast\_statistics file by concatenating all the \*.blast.stat files.

#!/bin/bash

if [ -z $1 ]

then

while [ -z $org\_dir ]

do

read -p "Organism directory: " -a org\_dir

done

else

org\_dir=$1

fi

echo $org\_dir

echo "orf\_id target query\_length score expect identities positives gaps" > blast\_statistics.txt

for chrom\_dir in $( ls -d $org\_dir\*/ );do

cat $( ls $chrom\_dir\*.blastp.txt.stats) >> blast\_statistics.txt

done

### do\_all\_blast\_stats.sh

Apply the Perl script (**extract\_header.pl**) that extracts statistics to all the BLAST reports.

#!/bin/bash

if [ -z $1 ]

then

while [ -z $org\_dir ]

do

read -p "Organism directory: " -a org\_dir

done

else

org\_dir=$1

fi

echo $org\_dir

for chrom\_dir in $( ls -d $org\_dir\*/ );do

for orf in $( ls $chrom\_dir\*.blastp.txt);do

echo "BLAST stats " $orf

perl ~/genomes/extract\_header.pl < ${orf} > ${orf}.stats

done

done

### create\_blast\_statistics\_tbl.sql

Enter this at the psql command line:

CREATE TABLE blast\_statistics

(

sk\_blast\_statistics SERIAL -- synthetic primary key

, tax\_id bigint -- NCBI taxonomy id of target

, organism character varying(100) -- convenience name of organism

, chromosome character varying(50)

, orf\_id character varying(50)

, target character varying(50) -- typically, chembl\_id

, query\_length int

, score numeric

, expect numeric

, identities numeric

, positives numeric

, gaps numeric

, import\_date timestamp not null default clock\_timestamp()

);

CREATE TABLE blast\_statistics\_import

(

orf\_id character varying(50)

, target character varying(50)

, query\_length int

, score numeric

, expect numeric

, identities numeric

, positives numeric

, gaps numeric

);

### import\_p\_falciparum.sql

(run this at psql prompt logged in as chembl\_25:

truncate table blast\_statistics\_import;

\copy blast\_statistics\_import from 'blast\_statistics.txt' delimiter E'\t' CSV HEADER

insert into blast\_statistics

( tax\_id, organism, orf\_id, target, query\_length, score, expect, identities, positives, gaps)

SELECT 36329 -- tax\_id

, 'Plasmodium falciparum 3D7'

, orf\_id

, target

, query\_length

, score

, expect

, identities

, positives

, gaps

FROM blast\_statistics\_import;

## HMM targets

### do\_all\_jackhmmer.sh

#!/bin/bash

if [ -z $1 ]

then

while [ -z $org\_dir ]

do

read -p "Organism directory: " -a org\_dir

done

else

org\_dir=$1

fi

echo $org\_dir

for chrom\_dir in $( ls -d $org\_dir\*/ );do

for orf in $( ls $chrom\_dir\*.FASTA);do

echo "jackhmmer " $orf

jackhmmer --domtblout $orf.summary -o $orf.hmm.txt $orf ~/hmmer\_targets/component\_sequences.fa

done

done

### extract\_hmm\_summary.pl

(in the ~/genomes directory.)

#!/bin/perl

use Switch;

if (scalar(@ARGV) < 1) {die "No filename passed.\n";}

my $text\_fn = $ARGV[0];

my $summary\_fn;

# print $text\_fn,"\n";

$summary\_fn = $text\_fn;

$summary\_fn =~ s/.hmm.txt/.summary/;

# print $summary\_fn,"\n";

my @lines;

open($IN, "<", $summary\_fn ) or die "Can't open $summary\_fn\n";

@lines = <$IN>;

close($IN);

# print "Lines: ",scalar(@lines), "\n";

my %target;

foreach my $line(@lines){

if ( $line =~ m/^(CHEMBL\S+)\s+(\S+)\s+(\S+)\s+(\S+)\s+(\S+)\s+(\S+)\s+(\S+)\s+(\S+)\s+/ ) {

if ( ! exists $target{$1} ) { # prevent duplicate line for a target match

print $1,"\t", $3, "\t", $4,"\t",$6, "\t", $7, "\t", $8, "\n";

$target{$1} = 1;

}

}

}

### do\_all\_hmmer\_stats.sh

Run this script in the **~/genomes** directory to create **hmm\_stats.txt** file which gathers all the generated stats.

#!/bin/bash

if [ -z $1 ]

then

while [ -z $org\_dir ]

do

read -p "Organism directory: " -a org\_dir

done

else

org\_dir=$1

fi

echo $org\_dir

echo "target tlen orf qlen evalue score" > hmm\_stats.txt

for chrom\_dir in $( ls -d $org\_dir\*/ );do

for orf in $( grep -L "\[No hits" $chrom\_dir\*hmm.txt ); do

perl ~/genomes/extract\_hmm\_summary.pl $orf >> hmm\_stats.txt

done

done

### create\_hmmer\_stats\_tbls.sql

Import this from the psql command line as chembl\_25 user.

CREATE TABLE hmmer\_statistics

(

hmmer\_statistics\_id SERIAL

, tax\_id numeric

, organism character varying(100)

, chromosome character varying(50)

, target character varying(50)

, tlen int

, orf character varying(50)

, qlen int

, evalue numeric

, score numeric

, import\_date timestamp not null default clock\_timestamp()

);

CREATE TABLE hmmer\_statistics\_import

(

target character varying(30)

, tlen int

, orf character varying(30)

, qlen int

, evalue numeric

, score numeric

);

### import\_hmmer\_statistics.sql

Import this script from the psql command line as user chembl\_25.

This script is in **~/genomes**.

truncate table hmmer\_statistics\_import;

\copy hmmer\_statistics\_import from 'hmm\_stats.txt' delimiter E'\t' CSV HEADER

insert into hmmer\_statistics

( target, tlen, orf, qlen, evalue, score)

select target, tlen, orf, qlen, evalue, score

from hmmer\_statistics\_import;

## Consolidated statistics analysis

### consolidated\_orf\_target.sql

\copy ( select h.orf

, b.target,b.score as blast\_score

, h.score as hmmer\_score

, b.expect as blast\_expect, h.evalue

FROM blast\_statistics b

join hmmer\_statistics h

on b.orf\_id = h.orf and b.target = h.target)

to C:\Users\Jeremy-satellite\Documents\RBIF120\consolidated\_stats.txt CSV delimiter ' ‘

### compare\_scores.R

# Scores comparison

consolidated\_stats=read.csv(file = "consolidated\_stats.txt", sep='\t', stringsAsFactors = FALSE)

attach(consolidated\_stats)

plot(blast\_score,hmmer\_score,main='Comparison of BLASTP vs HMM scores for P. falciparum with targets')

abline(a=0,b=median(hmmer\_score/blast\_score)+mad(hmmer\_score/blast\_score),col='red')

detach()

### Score normality and kmeans analysis

# score normality

stats=read.csv(file='../process\_plasmodium/blast\_statistics.txt', sep="\t", stringsAsFactors = FALSE)

organism="P. falciparum 3D7"

attach(stats)

qqnorm(log(score),main='Q-Q Plot for BLASTP log(scores)')

qqline(log(score),col='red',lwd=3)

detach()

norm\_log\_thresh=median(log(stats$score))+2\*mad(log(stats$score))

norm\_thresh=median(stats$score)+2\*mad(stats$score)

print(paste('norm\_log\_thresh:',norm\_log\_thresh))

print(paste('norm\_thresh:', norm\_thresh))

norm=stats[log(stats$score) < norm\_log\_thresh,]

qqnorm(log(norm$score), main='Q-Q Plot for log normal scores')

qqline(log(norm$score), lwd=3, col='red')

highly\_similar=stats[stats$score > norm\_thresh,]

consolidated\_stats=read.csv(file = "consolidated\_stats.txt", sep='\t', stringsAsFactors = FALSE)

attach(consolidated\_stats)

qqnorm(log(hmmer\_score),main='Q-Q Plot for HMMER log(scores)')

qqline(log(hmmer\_score),lwd=3,col='red')

# kmeans analysis

kh=kmeans(hmmer\_score,2,nstart=25)

k\_threshold = min(hmmer\_score[kh$cluster==1])

plot(hmmer\_score,col=kh$cluster,main='2 kmeans clusters for hmmer\_score',

sub=paste('Significance threshold:',k\_threshold))

print(paste('Kmeans threshold for significance:', k\_threshold))

abline(h=k\_threshold,lwd=3, col='purple')

detach()

## Download p. falciparum drugs and targets

/\*\* this is the copy directive used by the **psql** command line to download \*\*/

\copy (select avg(score), td.tax\_id as original\_tax\_id, td.organism as orig\_organism, md.pref\_name, md.chembl\_id from hmmer\_statistics h join target\_dictionary td on h.target = td.chembl\_id join drug\_mechanism dm ON dm.tid = td.tid join molecule\_dictionary md ON dm.molregno = md.molregno WHERE h.tax\_id = 36329 and score > 385.3 group by td.tax\_id, td.organism, md.pref\_name, md.chembl\_id order by avg(score) desc) to 'C:/Users/Jeremy-satellite/Documents/RBIF120/paralog\_targets/supplements/targets/p\_falciparum\_hmmer\_drugs.txt' CSV HEADER delimiter ' '

1. [2] [↑](#footnote-ref-1)
2. [3] [↑](#footnote-ref-2)
3. [7] [↑](#footnote-ref-3)