**Project Report: E.coli Virulence Finder**

My class project yielded the E.coli Virulence Finder program (E.coli\_Virulence\_Finder.sh), a pipeline that turns accession number input into an output showing all hits in reference genomes, data on the strength of the BLASTn sequence identity, and the full sequence of each hit. While the original purpose of this program was to search for the conservation of speculative/known virulence genes among enteropathogenic E.coli, it can also be much more versatile; adding gene queries is as easy as adding accession numbers to the associated help.txt file, and adding reference genomes just involves adding new TaxIDs to one line of the code. The result is a streamlined and focused BLAST search against an adjustable range of reference genomes.

E.coli\_Virulence\_Finder.sh is a Bash script that sorts hits using R so all 3 files uploaded to the repository are all necessary to run the program:

1. **help.txt** file is a list of the currently tested accession numbers (and associated gene information), the TaxIDs of current reference genomes (can also be adjusted), and a description of the interface and output of the program.
2. **E.coli\_Virulence\_Finder.sh**  is the brunt of the pipeline; the accession number is run through command line BLASTn, the BLAST output and Fasta sequences are saved, and the hits found are automatically run through the data generating R script
3. **genes.R** is the data sorting, cleaning, and output R script. Columns are labeled, hits are sorted by bit score, and poor hits, hits to self, or duplicate hits from the same TaxID are removed.

Additionally, entering the program without any accession # queries ($ ./E.coli\_ Virulence\_Finder.sh <blank>) yields a read out of the help.txt file. Next I will run through what each line of the script does.

1. Open script and download necessary databases

$cat E.coli\_Virulence\_Finder.sh

#!/bin/bash

# required v5 nt blastdb ftp://ftp.ncbi.nlm.nih.gov/blast/db/v5

# <ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/2.8.0alpha/>

My pipeline uses accession numbers as queries, and TaxIDs to search against, a task that I found the BLAST module currently on the Biocluster to be incapable of. Instead I had to download and unzip the newest BLAST database (2.8.0+), as well as the V5 NCBI Database, a folder which I then had to individually download and unzip 55 sub-folders from. This code to do so is not in my final program but is listed below:

#access NCBI V5 Database, then download all subfolders  
lftp <ftp://ftp.ncbi.nlm.nih.gov/blast/db/v5>

> get <subfolder.tar.gz>

#unzip and de-archive each subfolder

for h in `seq -f "%02g" 0 55`; do

echo "nt.$h.tar.gz";

tar -vzxf nt.$h.tar.gz

done

#then download, unzip, and de-archive BLAST 2.8.0+

wget <ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/2.8.0alpha/ncbi-blast-2.8.0-alpha+-x64-linux.tar.gz>

tar -vzxf <ftp://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/2.8.0alpha/ncbi-blast-2.8.0-alpha+-x64-linux.tar.gz>

The NCBI and BLAST databases full urls then needed to be referenced in my script (Lines 33 + 28 of the E.coli\_ Virulence\_Finder.sh respectively)

1. Display help.txt file if there is no accession # input

#referencing the Stack Overflow answer that helped me address this particular problem

#http://stackoverflow.com/a/4419971/2146843

exitfunc()

{

exit 1

}

#get first arg from command line

GENE="$1"

if [[ -z $GENE ]]; then

echo

cat help.txt

exitfunc

fi

1. Limit Search to (TaxID list) and access databases

#limit results to these tax ids

TAXIDS\_TO\_SEARCH\_AGAINST="591946,364106,362663"

[^ This line (line 24) of the script is the one that would be adjusted if the user wanted to run the script + accession # against additional genomes. Simply add another TaxID to the comma delimted list, and that TaxID will be included in the pipeline.]

#location of blast executables

BLASTDIR=/bigdata/gen220/gloga001/ncbi-blast-2.8.0+/bin

BLASTN=$BLASTDIR/blastn

BLASTCMD=$BLASTDIR/blastdbcmd

#location of blastdb

NCBIDIR=/bigdata/gen220/gloga001/v5

DB=$NCBIDIR/nt

1. Extract accession # input, and use command line BLAST to run BLASTn

#extract fastas from nt database

$BLASTCMD -db "$DB" -entry "$GENE" -out "$GENE.fasta"

[^ This command line blast is what searches the accession # entry ($GENE) against the BLAST database ($DB) and saves the fasta file of the sequence.]

#blast fastas to nt dabase

OPTS="qseqid qlen saccver pident gaps mismatch length evalue bitscore staxids stitle qseq sseq"

[^ this line matches the R script results read out; these were the parameters of the alignment I chose to address]

QUERY="$GENE.fasta"

SIZE=$(stat --printf="%s" $QUERY)

#run blast

if [[ $SIZE -ne 0 ]]; then

echo $GENE

OUTFILE="$GENE.blastout.txt"

$BLASTN \

-task 'blastn' -db "$DB" \

-query "$QUERY" \

-taxids "$TAXIDS\_TO\_SEARCH\_AGAINST" \

-max\_target\_seqs "100" \

-evalue "0.001" \

-outfmt "7 $OPTS" \

-out "$OUTFILE"

else

echo "$GENE not found in nt"

fi

[^This If/Else format was used because even though some accession #’s were accessible when checked using the online BLASTn interface, not all (only one of the ones tested so far, which was then removed from the help.txt file) were not able to be found from the local databases I downloaded. As a result, the fasta file file size was 0,“$GENE not found in nt” is displayed, no blastout.txt file is generated, and the pipeline does not proceed.]

1. Run the R script using Bash

#run the r script

Rscript genes.R

1. Display the results generated by the R script

#display results with less (which are saved in R)

cat $GENE.results.txt | less -S

[^ The results were displayed using cat + less for several reasons. Since I’ve so far been testing genes against 3 genomes, less actually displayed all the rows of the R scripts results; less also displayed the full sequences in a linear fashion, instead of condensing the entire sequence below each result, thus allowing the hits columnar data to be seen more easily.]

The R script is pretty straightforward, though I added several segments of code to clean up the data, each of which is pretty clear from the #commented out titles.

1. Access the blast output generated by the command line BLAST script.

#get file names from current directory

files <- list.files(pattern="blastout.txt")

1. (Begin for loop on files) Read in the accesssed files and organize under defined column names

for(fp in files) {

# fp <- files[2]

#read file

hits1 <- read.delim(fp, header = FALSE, comment.char = '#')

#define column names

colnames(hits1) <- c("qseqid","qlen","saccver","pident","gaps","mismatch","length","evalue","bitscore","staxids","stitle","qseq","sseq")

# dim(hits1)

# hits1[,1:11]

gene <- sub("\\.blastout.txt", "", fp)

1. (still in for loop) hits were sorted and trimmed to personally chosen parameters.

#remove any hits to self

self\_rows <- grep(gene, hits1$saccver)

if(length(self\_rows) > 0) {

hits2 <- hits1[-self\_rows,]

} else {

hits2 <- hits1

}

# hits2[,1:11]

[^ The accession # would sometimes be using a sequence isolated from one of the reference genomes, so it would find have two 100% hits with different accession #s but the same TaxID. I didn’t consider these hits relevant to the programs results, so I removed them.   
Additionally, as I updated the results table in R, you’ll notice changes to the table version (hits 2 <- hits 1, hits3 <- hits2, etc). These were helpful when I needed to check the code after each change, but the command to show the hit tables have since been #commented out, so they are artifacts now.]

#sort descending by bitscore

hits3 <- hits3[order(-hits3$length),]

# hits3[,1:11]

[^This sort ends up being a bit redundant later, but there were occasionally multiple hits from the same TaxID; I wanted to remove these poor duplicate matches, but I wanted to make sure the high bitscored/accurate hit was the one that was kept.]

#remove duplicated taxid rows

dup\_taxids <- duplicated(hits2$staxids)

hits3 <- hits2[!dup\_taxids,]

# hits3[,1:11]

[^ because some of the gene queries had very short sequences, I would occasionally get duplicate hits from the same TaxID, which always had very poor bit scores. I chose to remove these poor hits as it seemed likely that they were off target artifacts.]

#remove poor hits by % of best aligned length

cutoff <- 0.5

denom <- hits3$length[1]

ratios <- hits3$length / denom

bad\_hits <- which(ratios <= cutoff)

if(length(bad\_hits) > 0) {

hits4 <- hits3[-bad\_hits,]

} else {

hits4 <- hits3

}

[^I also occasionally got poor hits due to off target plasmids that carried a version of the gene, which weren’t removed by previous code because it had its own accession #. I added this poor hit cut off so that if relevant plasmid carried versions of the gene were hits they would be included, but poor hits would still be excluded across the board. A 50% sequence identity was chosen arbitrarily and may be a target for future updates to the code.]

The results are a column delimited table as listed in the code above, such that every row is a search hit/genome that carries a copy/version of the gene, and the columns denote information about each hit;

Query sequence Accession# - Query Length – Accession # of the search hit/genome – Percent Identity - # of gaps - # of mismatches – Length of the aligned sequence – E value of the alignment – bitscore of the alignment – search hits TaxID – Title of the search hit – Query sequence – Aligned search sequence

Obviously this pipeline doesn’t really give any new information, it basically the same as BLASTing a sequences against a genome, it simply searches against multiple genomes simultaneously. At least on the hits that I have tested so far however, it is noticeably faster, and it allows you a greater ease with which to narrow down what the sequences are being queried against. This could be due to the simplicity of the searches, or thanks to the V5 NCBI database, but I still think it can be useful in the future. Further development on this code would obviously expand the library of genomes that are being used for comparison, but may also be improved in other ways:

* I considered having the files be removed after the report has been accessed, to stop users’ directories from becoming cluttered with files from previous searches. However it felt as though that step shouldn’t be automated, as the point of the program is to find hits for future research, so desired hits would want to be accessed later.
* Once genomes are added from strains and species that are separated much farther phylogenetically, the poor hit cut off may need to be adjusted.
* The same could be true if the pipeline is purposefully used to find homologs, orthologues, and paralogues; in this case the code suppressing multiple hits from the same accession number might need to be altered or commented out.
* Although only a couple non-virulence genes have been tested, this pipeline is ripe for testing many genes; highly conserved housekeeping genes, strain specific genes, theoretical new virulence genes, etc.