**Using BLAST+ (BLAST) on Kepler.**

**Case study illustrating how to use BLAST to annotate chloroplast genomes**

**Overview**

BLAST+ superseded the old BLAST suite about 10 years ago, but sometimes you might come across software that still uses the old blast (if you see the program blastall, it's legacy blast). Here BLAST means BLAST+.

BLAST is a suite of programs that align nucleotide or protein sequences to other nucleotide or protein sequences. If in the **alignment** the sequences are so much more similar to each other than you would expect to find by chance, then the better explanation for the similarity is that the two sequences are **homologous**: they are so similar because they share a recent common ancestor; both of the sequences are descendants of different copies of that ancestor. However, sequences *can* be significantly similar by complete random chance and sequences can also be significantly similar because of convergent evolution, but this is seen less often.  
  
With BLAST programs, you can experiment with the similarity thresholds to require more or less similarity to count an alignment as a "**hit**" (also called a HSP, or High Scoring Pair). Hits get reported in the BLAST program output.

**Steps for carrying out this BLAST search:**

* create a directory
* make symbolic links (shortcuts) to a FASTA file
* create a script with the necessary commands
* submit the script to a cluster queue
* after the jobs finish running, use an existing script to summarize the results
* compare results from different blast runs.

Fill in the table for kind of query and kind of database needed for each program: nucleotide or protein. Put nucleotide if the file you give to the program would be a nucleotide FASTA and protein if it would be a protein FASTA. Search Google for "blastn", "blastp", etc. to get to the NCBI page for that program quickly, which has the information needed to fill in the table.

|  |  |  |
| --- | --- | --- |
| **Program** | **Query** | **Database (subjects)** |
| blastn | n | n |
| blastp | p | p |
| blastx | n(translated) | p |
| tblastn | p | n(translated) |
| tblastx | n(translated) | n(translated) |

**Step 1. Try out BLAST**

1. Log on to the cluster with **ssh**
2. BLAST is installed on the cluster. To load it, type:

module load local/blast

1. "**flags**" are the same thing as "**arguments**" or "**options**", and are sometimes called "**parameters**" (which has a different meaning in statistics). They go after the program name and usually start with a single - or a double -- . Flags are settings or inputs for the program being run, and each program can have a different set of them. Flags are analogous to function arguments in Python, so the following two examples could be equivalent:

|  |  |
| --- | --- |
| Python | printFasta(fasta='myseqs.fasta') |
| shell | printFasta -fasta myseqs.fasta |

1. Try the different BLAST programs with the flags **-h** and **-help**. See that the help information shown for each of the different BLAST programs is mostly the identical. There are a pretty large number of options, but these are the five you need to be aware of for most situations.

**The minimal set of two options you need to run BLAST programs**

In the flags shown below, the tags <> indicate a place for your input, the nature of which is defined between the tags. In this first case (-query), 'fasta' indicates the program needs the path to a FASTA file there (the sequences you will be searching for homologous matches in the databases, which are on Kepler, see below for the paths to each).

-query <fasta>

-db <blast\_db\_basename>

<blast\_db\_basename> is the path to a set of files containing sequences that have been formatted specifically for BLAST programs. You could easily make your own database from any FASTA file, but we have some very large BLAST databases on Kepler (from NCBI) called 'nr' (protein sequences) and 'nt' (nucleotide sequences) that we'll be using for this search. 'basename' refers to the common prefix used for all files in the blast database. Because there are many files (take a look at the nr blast db files at /home/derstudent/data/other/nr/ ), by using a common basename, blast programs only need to know the basename and not the full name of every file in the database.

**Some other important options**

-outfmt 7

The standard output has a visual representation of the alignments, which is good to look at but hard to write scripts to parse it (to parse is to manipulate the contents of a file to extract information from it). -outfmt 7 is tabular, with columns and rows.

-evalue <max\_evalue>

The E-value can be thought of as conveying the same basic message as a p-value: The lower the E-value the less likely it is that you would find that alignment/hit by chance. The lower the E-value of a HSP, the more likely the sequence similarity between the query and the subject in the HSP is due to shared common ancestry.

The default E-value is 10, which is too high for a lot of applications. Usually I set -evalue between 0.1 (1e-1 = 1E-5 = ) and 0.00001 (1e-5 = 1E-5 = ). With an E-value threshold of 1 blast programs will find most of the same hits they would with and E-value of 1e-10 – only a great number of additional hits as well (whose homology to the query is less supported statistically).

-perc\_identity <min\_perc\_id>

Minimum % identity in alignment for a hit to be reported. The sequences in the following alignment would have 90% identity:

ATTAGGGCG

| ||| | || ||

ATTAGGGCT

-out <file>

If you don't specify a desired path for the output file BLAST programs will by default output everything to standard out (stdout). These two calls do the same thing:

|  |  |
| --- | --- |
| blastn -query nucleotide\_seqs.fasta \  -db /home/derstudent/data/other/nt/nt \  -out blast\_output.tab \  2>blastn.stderr | blastn -query nucleotide\_seqs.fasta \  -db /home/derstudent/data/other/nt/nt \  > blast\_output.tab \  2>blastn.stderr |

**Step 2. Set up your job submission scripts**

For this exercise, there is one kinds of sequence file:

* nucleotide sequences from chloroplast genomes

And two kinds of BLAST databases:

* chloroplast gene protein sequences
* chloroplast gene nucleotide sequences

**Your mission:**

Using **blastn**, **blastp**, or **tblastx**, blast a Santalales chloroplast genome assbembly against a set of protein or CDS sequences of many chloroplast genes from other plants. You can think of choosing between these three programs as an additional threshold settings, like -evalue and -perc\_identity. In this case, which program out of blastn, blastp, tblastx would you expect to return the most hits? How about the least hits? Assume you use the same -evalue and -perc\_identity thresholds for all three programs.

1. On the cluster, create a new directory for conducting this blast search.

The sequence files to choose from are in these directories:

* + Santalales FastPlast assemblies

/home/derstudent/data/santalales/fast-plast.assemblies/renamed\_scaffolds

* + Chloroplast gene CDS sequences

/home/derstudent/data/santalales/chloroplat\_genes\_fastas/Fast-Plast\_set.osyrisCDS\_v2.viscumCDS\_v2.fna

* + Chloroplast gene protein sequences

/home/derstudent/data/santalales/chloroplat\_genes\_fastas/Fast-PlastSet.osyrisV2.viscumV2.faa

1. The chloroplast gene CDS and protein sequences are in FASTA format and need to be converted into BLAST database format. This can be done using a program called makeblastdb that gets loaded when you use module load local/blas

**Alethea** and **Brittany**, each of you make

a blast database from either the chloroplast gene CDS sequences or the chloroplast protein sequences.

Many hands make light work!

1. makeblastdb can't use multiple processors, but works quickly, so it's not necessary to use qsub to submit a script from the queue.
2. Even though makeblastdb finishes pretty quickly, it's still a good idea to use a compute node to run makeblastdb if one is available, which you can find out by pressing Q (if you're on the derstudent account. If you're not, we can set this alias/script up). I think some of q24's nodes are not working, so even though it may look like there are plenty of processors available on q24 nodes it may be those nodes are actually offline. So use only q40 or performance nodes for this step, and if there isn't a processor available on any of these nodes then just run makeblastdb on the head node.
3. You can log on to the node you chose using ssh:

ssh <node\_name>

where <node\_name> is kepler-0-#, and

# is the # of your node:

performance

q40

1. If you haven't done so, load the blast module:

module load local/blast

1. What are the necessary flags for makeblastdb?

makeblastdb -h

Look at the condensed help output when you use -h. What are the options that are absolutely necessary to run makeblastdb? Hint: the convention for this kind of output is: any options/flags in straight brackets are optional. If an option/flag is not in straight brackets, that flag must be present when you use that program.

Check your answer be looking at the option in the full help output using:

makeblastdb -help

The other two flags you need are -in and -parse\_seqids. Specify the file with your sequences using -in <file>, and just add the -parse\_seqids flag. This allows you to extract sequences from the database by name later.

1. Run makeblastdb.
2. Use this shortcut I often use to get the full path (the full path is the path from the very top of the filesystem, '/', all the way through every directory necessary to get to a particular file:

ls `pwd`/<filename>

Copy the output and paste it somewhere safe. You'll use it in when you use blast. This is equivalent to typing **pwd**, copying the output, pasting it, then adding a slash and the filename.

1. Navigate back to your blast directory and then use **ls** to see the files in the Santalales FastPlast assemblies directory (see step 1), and choose one to annotate (using blast).

ls <path\_to\_directory>

1. Inside the new directory, make a symbolic link to your chosen sequences using **ln**:

ln -s <path\_to\_file>

1. Copy the template qsub script to your blast directory using **cp**. The template is here

/home/derstudent/scripts/template.qsub

**Use nano, emacs, or vim to replace the highlighted items in the script below**. You can also edit it in this document, then copy and paste it into nano, emacs, or vim. That would be faster for now but eventually you need to be able to use a text editor in the terminal.

In order from top to bottom, the things to replace in the script are:

1. **Your email address.** When your job finishes you'll get an email notification.
2. **q40**. This is the queue to use (a queue is a group of nodes in the cluster). For us it's almost always 'q40' or 'performance'. Type: Q and see if there are processors available in the q40 or performance queues, and use the queue that has processors available, if any. If not, use q40.
3. **ppn=40**. This is "processors per node". Set it to the maximum number of processors available for any single node in q40 or performance.
4. **jobname** any name for your job, but no spaces
5. **Your blast output directory**
6. Your commands. This should include module load local/blast and your blast command with flags.

module load local/blast

blastn -query nucleotide\_seqs.fasta \

-db /home/derstudent/data/other/nr/nr \

-evalue 1e-5 \

-perc\_identity 65.5 \

-outfmt 7 \

-out blast\_output.tab \

> blastn.stdout \

2>blastn.stderr

The backslashes make a multi-line call like this possible. The backslash has a special meaning in bash. It means "cancel the normal meaning of whatever is immediately in front of me". In this case, it's cancelling the meaning of the end of the line. So this mult-line call is equivalent to this single line call:

blastn -query nucleotide\_seqs.fasta -db /home/derstudent/data/other/nr/nr -evalue 1e-5 -perc\_identity 65.5 -outfmt 7 -out blast\_output.tab > blastn.stdout 2>blastn.stderr

The first is easier to read.

#!/bin/bash

#PBS -k oe

#PBS -j oe

#PBS -m ea

#PBS -M your@email.address

#PBS -q q40

#PBS -l nodes=1:ppn=40

#PBS -N jobname

# Comments start with #

# On a line, anything after a # is ignored when the script is executed

# Put your working directory here

cd /my/working/directory

# Prints when the job starts

echo "START"

date

# Put things to do here!

# Delete the example below and replace it with your commands

module load local/bedops

gff2bed < imin.gff > imin.bed

# Prints when the job ends

echo "END"

date

Make a copy of the qsub script and get rid of the -outfmt flag. Even though the default output is cumbersome to manipulate and extract information from, looking at it will help you understand BLAST more.

When you have finished editing your script, submit it to the queue system with qsub like this:

qsub <script>

If you have the correct working directory you should see the stdout and stderr files appear. If you also have the right queue, not too many processors, you loaded the blast module, and you have a valid bash call, your job should start (if there are available processors on the cluster).

**Step 3: Interpret the blast results**