# Gene Annotation

Julie Boisard, Courtney Stairs, Dag Ahren, Björn Canbäck, Joel Wallenius

2024-01-11

## Contents

# 1 Before you begin

- Do you remember how to copy files from your personal computer to your machine on campus? Or from our biology server to your machine on campus? Still need help? Ask us or Google.
- The last exercise, using eggNOG, is optional. It takes a while to run (30-60 mins), so please start it before the weekend.

# 2 Homology searching with BLAST

**Learning objectives:**

- Install NCBI blast on your local machine
- Download a BLAST database
- Run BLAST from the command-line
- Complete an exercise using BLAST

## 2.1 Download the BLAST executables

We're going to download the BLAST program available from NCBI to your local computer on campus (**not the server**). Select one of the options below (`conda` is **Option C**). I encourage you to install using the 'old school' method to install the pre-compiled binaries if you do not have a lot of experience. Not every software will have a conda package so it is helpful to know! However, if you feel confident in your installing skills, head to **Option C**.

**Option A:**

Go to the BLAST download page at: http://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/ and download the compiled program for Linux machines. `ncbi-blast-2.15.0+-x64-arm-linux.tar.gz`. Transfer this file to the `~/bin/` directory of your computer on campus. And proceed to the tar.gz extraction below (**Extracting & PATH variable**)

**Option B**:

Go to the BLAST download page at: http://ftp.ncbi.nlm.nih.gov/blast/executables/blast+/LATEST/ and **right click** on the executable file you would like to download `ncbi-blast-2.15.0+-x64-linux.tar.gz` and click 'Copy Link Address'.

`cd` into to your `~/bin/` directory and retrieve the compressed BLAST executables on your clipboard directly from NCBI using `wget`.

```
cd ~/bin/
# wget will retrieve the file to your current direction
wget PASTE_URL_HERE
# check to make sure your ncbi-blast-2.15.0+-x64-linux.tar.gz
# file is there with ls
ls
```

Proceed to **Extracting & PATH variable**

**Option C**:

Make a new environment with `conda` called 'blast_env', search the bioconda channel and install:

```
conda create -n blast_env
```

```
conda activate blast_env
conda search -c bioconda blast
```

Proceed to **Database installation**

## 2.2 Extracting & PATH variable

Skip this step and go to **Database installation** if you used `conda` to install the blast program.

Next, we need to extract the binaries from the .tar.gz file and tell our bashrc file where it can find the blast binaries (programs).

Make sure you are in the `~/bin/` directory:

```
tar -xvzf ncbi-blast-2.15.0+-x64-linux.tar.gz
```

Take a look at the blast binaries (programs):

```
ls ncbi-blast-2.15.0+/bin/
```

We want to add this directory to the PATH variable.

Open your `~/.bashrc` file in your home folder (don't forget the dot!) in Atom. Add a new `PATH=`:

```
# Be sure to keep "$PATH:" in your PATH
export PATH="$PATH:/$HOME/bin/ncbi-blast-2.15.0+/bin/"
```

Exit Atom (or your text editor). Now we have to `source` our new `.bashrc` to be able to access the binaries.

```
source ~/.bashrc
```

## 2.3 Database installation

Go to your home directory if you are not already there and create a new directory:

```
cd
mkdir BlastDB
cd BlastDB
```

Look what NCBI offers when it comes to sequence databases (BLAST format). This takes 30 seconds or so to load:

```
update_blastdb.pl --showall
```

Did you get an error unable to find 'update_blastdb.pl'?? Check to make sure you did everything in **Extracting & PATH variable** above.

Download the `pdbaa` database, this database contains all the amino acid sequences of the Protein Data Bank. Read more about the PDB here if you'd like: https://www.rcsb.org/

```
# Should take 1-3 minutes
update_blastdb.pl pdbaa
ls
```

You will see two files. One is the tar archive for the database itself. The other is `pdbaa.tar.gz.md5`. This file contains the so called md5sum (fingerprint) of the tar archive. If the archive file has been successfully downloaded the md5sum of this file should be the same as in `pdbaa.tar.gz.md5`. Test:

```
md5sum -c pdbaa.tar.gz.md5
```

Now we extract the database:

```
tar -xvzf pdbaa.tar.gz
```

The files with .p?? extensions constitute the BLAST database. Notice that there is no fasta file. The sequences are embedded in binary format in the database files. However, they can be retrieved:

```
blastdbcmd -outfmt %f -db pdbaa -entry all -out pdbaa.fas
less pdbaa.fas # Take a look
```

`-outfmt %f` means that output should be in fasta format.

In practice, you will likely use much larger databases in the future, such as the `nr` database. However, this database contains millions of sequences and cannot be downloaded within the time frame of the course. Keep this in mind when you start to set up you own databases in the future.

## 2.4 Make your own BLAST database

Next, we will give an example of how to create BLAST databases from one nucleotide file and one protein file. While in you BlastDB folder, retrieve the **paxillus.fna** and **paxillus.faa** files from the Course Resources directory (available on the server in ls `/home2/resources/binp28/Data/`) and unzip them if they are compressed.

These files contain protein (**paxillus.faa**) and Illumina reads (**paxillus.fna**) from the Basidiomycete fungus *Paxillus involutus*.

You can create your own database from any fasta file using the BLAST+ command `makeblastdb`. Create a database of the files you copied:

```
makeblastdb -in paxillus.fna -out paxillusNt -dbtype nucl
ls -l paxillusNt.n* # n stands for nucleotide
```

We also do it for the protein file:

```
makeblastdb -in paxillus.faa -out paxillusAa -dbtype prot
ls -l paxillusAa.p* # p stands for protein
```

`-in`: The fasta file to build the database from.

`-out`: The basename of the files that the database will consist of. This will be the database name.

`-dbtype`: The type of fasta file used: `prot` for proteins, `nucl` for nucleotides.

If you want to look for other options, do:

```
makeblastdb -h    # For short help
makeblastdb -help # For extensive help
```

## 2.5 BLAST example usage

Here comes an example of a regular BLAST run.

```
cd .. # Move up to the parent directory
mkdir BlastExercise
cd BlastExercise
```

Download the the file `yeast.faa` from the Course Resources folder. The file contains the proteome of baker's yeast (which is an ascomycete). Are there similar proteins in these two fungi? Let us check:

```
blastp -query yeast.faa -db ../BlastDB/paxillusAa -evalue 1e-10 \
-out yeast_vs_Paxillus.blastp -num_descriptions 10 \
-num_alignments 5 -num_threads 2
```

Here are a description of the options specified in the command. Remember, if you ever forget these flags, you can always use `-h` or `.help`.

`-query`: The file containing the **query** sequences.

`-db`: The database to compare against containing the **'target'** or **'subject'** sequences.

`-evalue`: The e-value threshold, 'hits' that have e-values above this value will not be reported. The default value is 10 (quite high), so we specify 1e-10 to avoid retrieving false positives.

`-out`: Name of BLAST output file.

`-num_descriptions`: Number of hits shown in the list of hits. One per line.

`-num_alignments`: Number of local alignments to show. Restrict this if you don't want a big output file.

`-num_threads`: The BLAST program can be parallelized on multiple processes to speed up computations. This options specifies how many threads that should be used. If you don't know how many threads you can use you can find out in `bash`:

```
nproc
```

The BLAST search will take between 5 and 10 minutes. Go stretch your legs. If you are impatient you can monitor the process of the job in a new terminal.

Open a new terminal with `Ctrl-Shift t` (if **Working From Home** remember to `ssh` in on your new terminal window!) and follow the progress:

```
# Check how many queries there are.

cat yeast.faa | grep -c '>'

# Check processed queries, in the output,
# each query will start with 'Query='

cat yeast_vs_Paxillus.blastp | grep -c 'Query='
```

PRO-TIP: Whenever you do an analysis, compare the input and output files, even when the job appears to have completed successfully. Do your grep counts above from the query files (>) and output files ("Query=") match?

Take a look at the output file:

```
less yeast_vs_Paxillus.blastp
```

## 2.6 Exercises

Take help of the BLAST Command Line Applications User Guide at:

http://www.ncbi.nlm.nih.gov/books/NBK1763/ if needed.

### 2.6.1 BLAST Exercise 1

1. How many of the yeast proteins had hits in the *Paxillus* proteome?

*Note: When the query doesn't have a hit the string "No hits" will appear in the BLAST output.*

### 2.6.2 BLAST Exercise 2

As mentioned, *Paxillus involutus* is a basidiomycete. Another basidiomycete is *Phanerochaete chrysosporium*. Download the protein file **Phrcr2.faa** for this fungus from the course Resources Directory. Using what you've learned above, run the necessary commands to answer the following questions:

2. How many *Phanerochaete* proteins produce significant hits against the *Paxillus* database? Motivate your answer.
3. Which of the two proteomes (yeast and *Phanerochaete*) had most hits against *Paxillus*. (Use an evalue of 1e-10)
4. Is the result expected?

### 2.6.3 BLAST Exercise 3

We now want to use all possible BLAST binaries (**blastn**, **blastx**, **blastp**, **tblastn**, **tblastx**). Start by downloading **cytB.fna** and **cytB.faa** from the course directory (this is a segment of cytochrome B is encoded on the mitochondrial genome 'mtDNA' not the nuclear genome). Create two databases consisting of one sequence each, DNA in one and protein in one:

```
# Nucleotide:
makeblastdb -in cytB.fna -out nt -dbtype nucl

# Protein
makeblastdb -in cytB.faa -out aa -dbtype prot

# Take a look
ll nt* aa*
```

Now try the five BLAST binaries just to get used to it. The first search is conducted as follows, try to run the others on your own:

```
blastn -query cytB.fna -db nt # Prints to stdout which is ok for now
blastp
tblastn
tblastx
blastx
```

5. Explain the result for tblastx above.

One common mistake (because it's still a valid search) is this:

```
blastp -query cytB.fna -db aa
```

6. What is the mistake and why does it work?

# 3 Detecting protein domains using InterPro

## 3.1 Learning objectives:

- Search for InterPro domains in a single protein online
- Search for InterPro domains in a FASTA locally
- Analyze the outputs of the local InterPro search
- Compare InterPro and BLAST outputs for an entire proteome.

There are several ways to annotate proteins. One is to identify conserved domains. Oen of the most widely used tool for this purpose InterProScan. The software uses hidden Markov models (HMM) and can be executed online or locally.

## 3.2 InterPro on the web

Go to the `interpro` web site at https://www.ebi.ac.uk/interpro/. Search `interpro` using the following sequence from *Paxillus involutus*.

```
>gnl|BL_ORD_ID|9jgi|Paxin1|164838|fgenesh1_kg.1_#_6_#_isotig02043
MLGHQFAYSAAPIRKVKEVQFGILSPEEIKAYSVAKIEHPEVMDETTHKPKLGGLMDPRM
GTIDRNFKCQTCGEGMSECPGHFGHIELARPVFHPGFIVKVKKILECICVNCGKLKADIS
DPNFADKIRHVRDPKARMAVVWSHCKTKMVCEADEPKEEGAEGDVDEPKKGHGGCGHIQP
LVRKEGLKLFVQYKRPKDEDEDVKSMQPDKRLITPSEVYTTFKKMSDSDLHLIGLSDEYA
RPEWMILTVMPVPPPPVRPSIAVDGGAMRSEDDLTYKLGDIIKASANVRRCEQEGAPAHV
ITEFEQLLQFHVATYMDNDIAGIPQALQKSGRPVKAIRARLKGKEGRLRGNLMGKRVDFS
ARTVITGDPNLELDEVGVPRSIAMNLTYPERVTPYNIAYLQELVRNGPTTYPGARYVVRD
TGERIDLRYNKRADAFLQYGWIVERHLKDGDFVLFNRQPSLHKMSMMSHRVRLMPYSTFR
LNLSVTPPYNADFDGDEMNMHIPQSEETRAELSQIAWVPRQIISPQANKPVMGIVQDTLC
GIRKFTLRDTFLDWNHVQNILLWIPDWDGTVPTPTIIRPKPLWTGKQILSMVIPRGINIH
RAPDPKSSHPVFDDGMMIDNGEIMFGIVDKKTVGATQGGLIHVVFREKGPEATRQLFTGI
QTVVNFWLFHNGFSIGIGDTIAGPKVMNYIAQRIGDKKQQVAEIIEDAYHDRLKPMPGMT
IRESFESKVEGELNRARDDSGQYAQKNLKEDNNVKQMVTAGSKGSYINISQMSVCVGQQS
VEGRRIPFGFRHRTLPHFTKDDFSPEARGFVENSYLRGLTPQEFFFHAMAGREGLIDTAV
KTAETGYIQRRLVKALEDVMVCYDGTVRNSLGDLIQFIYGEDGMDGAFIERQKIDTFGLS
DKEFEHNYRVDVTDKEGGFLPGVLQIGIDDSSLELQAKLDEEYGRLVQDRHELRNFVFPR
ADGLTPHYLPVNLQRIIQNAVQIFHIDRRKPSDLEPAYIVEAVQQLADRLVAIPGNDEMS
KEVQANASLTFRMHVRATLATRRVLEQYHLNRQAFEWVLGEIEAKFNQSLVNPGEMCGTL
AAQSIGEPATQMTLNTFHYAGVSSKNVTLGVPRLKEIINVATNIKTPSLSVYLEPELAEK
NTLAKNVQQELAYTSLRTVTSAVEIWYDPDPTSTIIEEDSVFVESFFAIPDDEIESKLHL
QSPWLLRLELDRARMIDRKLTMAYVAGQIAENFKTDLFVIWSEDNSEKLIIRCRVLGGGD
KEDDGMGSVEEDIFLRQLENTMLNSVSLRGVPGINRVFLQEHDKVYVNEEGSIKSKKQWM
LETDGINLKTVMCIDGVDFTQTYSNSGVEIFNVLGIEAARAAIMRELRGVIEFDGSYVNY
RHLALLCDLMTHHGTLMAITRHGINRADTGALMRCSFEETVEILMEAAAVGEKDDCHGIA
ENVMFGQLAPMGTGAFDVALDIDMLKDAIVDHRLPVQNMLAAHADGGMTPGQVAMTPYDT
NSPAWSETAFKGESAAFSPLAVQGGEDAASFSFLGYGQSPLGAGGMSPAGPGYSPSSPNA
YSPTSPYVPQSPFGGATSPFGTSPYATSPFYDRGRGPTSPTYSPTSPALNLTSPGYSPTS
PRYSPTSPSFSPTSPRYSPQSPSFSPTSPRYSPTSPSFSPASPRSFLIAPAQMSPSSPKY
```

```
SPTSPASPSSPKYSPTSPTYSPASPAYSPASPAYSPTSPQWSPSSPAQNGAAARGHSYST
SPSWE*
```

### 3.2.1 InterPro Exercise 1

1. Which domains did you find?
2. What is the function of the domains?
3. Click on the first domain to go to its details page.

## 3.3 InterPro from the shell using InterProScan

You can also search all databases integrated in InterPro locally from the command-line.

Log in to the server and create a directory **Annotation** in you home folder. We will annotate domains for ten proteins from the *Paxillus involutus* proteome. Copy the `paxillus.faa.gz` file from the course resources folder into your new Annotation folder. Uncompress it. In the interest of time, we will look at just the first 10 sequences (first 78 lines):

```
head -78 paxillus.faa > 10.faa
# Interproscan does not like * (stop codons)
# so you have to get rid of them before
cat 10.faa | sed 's/*//g' > 10_nostop.faa
```

```
# Launch the InterPro annotation analysis
interproscan.sh -i 10_nostop.faa -cpu 2 \
-d paxillus_interpro \
-goterms
```

Take a look at the results, especially for `Paxin1_164838` which is the sequence you used on the web.

### 3.3.1 InterPro Exercise 2

We are going to compare InterPro and BLAST outputs to answer the following questions. Retrieve a pre-computed BLAST and InterPro (especially PFAM domains annotations) outputs for the entire proteome of *Paxillus involutus* from the course resource folder (`pi.blastp` and `pi.pfam`).

4. Looking at the beginning of the blastp file, which database was searched?

Sometimes, gene predictors can over-predict and identify ORFs that are NOT in fact protein-coding genes. If a protein/gene does not have a significant hit using BLAST or PFAM, it might be a false positive protein/gene prediction.

5. How many proteins did not have a significant hit in the BLASTP file? (Remember how to see if a query protein did NOT have a hit? Revisit the BLAST exercise.)

6. Using `bash`, determine how many of the proteins without a BLASTP hit had a PFAM hit.
7. If there were proteins that had hits using PFAM but not BLAST, how can you explain this?
8. (Optional) Based on this analysis, how many genes do you think were false positives? Why?
9. (Optional) What additional data would help you determine if this gene is real?

Tips:

- To get the sequence ID from the blast output file, try playing with `grep -B` and the `pi.blastp` file.

Let's take a break from PFAM and try this bash exercise.

Sometimes, it's easier if each sequence in a fasta file is written in only one line. Eventually you might want to write a simple python or bash process to convert a multiple line fasta file to one-line fasta files.

```
# This is the standard output of many programs where 60 characters are
# written on each line terminated by an
# 'end-of-line' character (\n or \r)

>FastaMultiLine
MLGHQFAYSAAPIRKVKEVQFGILSPEEIKAYSVAKIEHPEVMDETTHKPKLGGLMDPRM
GTIDRNFKCQTCGEGMSECPGHFGHIELARPVFHPGFIVKVKKILECICVNCGKLKADIS
DPNFADKIRHVRDPKARMAVVWSHCKTKMVCEADEPKEEGAEGDVDEPKKGHGGCGHIQP


# Although less pretty, it is useful for some fasta manipulation
# you can do on the command-line

>FastaSingleLine
MLGHQFAYSAAPIRKVKEVQFGILSPEEIKAYSVAKIEHPEVMD........
```

10. Analyze this command.

```
cat pi.faa | tr "\n" @ | sed "s/>/>anyString/g" | tr ">" "\n" | \
sed "s/anyString/>/" | sed "s/@/\n/" | \
tr -d @ | grep . >  pi.oneline.faa
```

# 4 Prediction of sub-cellular localization with SignalP

## 4.1 Learning objectives

- Use signalP 5.0 to detect signal peptides
- Use targetP to predict sub-cellular localization

## 4.2 SignalP

Many proteins in the cell that are destined to organelles or secretion have specific sequence patterns that we can detect with software. First we will look for secretion signals using the tool `signalP`.

We continue to work in the **Annotation** directory on the server. First, extract the first let's say, 143 sequence from the `paxillus.faa` file using `head` (we precomputed the line number for you, you're welcome).

```
head -1000 paxillus.faa > 143.faa
```

Then we run signalP on the 143 proteins:

```
signalp -fasta 143.faa
```

The output is as follows in `143_summary.signalp5`:

Column:

1. Your protein ID
2. The protein prediction, one of: - SP(Sec/SPI); - LIPO(Sec/SPII); - TAT(Tat/SPI); or - OTHER
3. The probability of column 2's prediction
4. The probability of another prediction
5. Cleavage Site (CS) position and associated probability.

For more information consult: https://services.healthtech.dtu.dk/service.php?SignalP-5.0 and references therein.

1. How many proteins are predicted to be secreted (SPI or SPII)?

To do this, we need to count which lines have an 'SP' annotation. BUT BE CAREFUL. You can see that the second line of the file containing the column headers contains a SP, we need to omit this from our result list. We can remove any line that started with a # using the -v ^# line.

```
cat 143_summary.signalp5 | grep -v ^# | grep SP |  \
cut -f 1 > paxillus143.signal.ids
```

## 4.3 TargetP

Still in your *Annotation** directory on the server we will use `targetp` to predict subcellular localization.

```
targetp -fasta 143.faa
```

The output is as follows in `143_summary.targetp5`:

Column:

1. Your protein ID
2. The protein prediction, one of: - SP (secreted); - mTP (mitochondrial); - plant specific: cTP (chloroplastic); - plant specific: luTP (thylakoid luminal transit peptide); - noTP (no prediction)
3. The probability of noTP prediction
4. The probability of SP
5. The probability of mTP prediction
6. Cleavage Site (CS) position and associated probability.

For more information consult: https://services.healthtech.dtu.dk/service.php?TargetP-2.0

```
less 143_summary.targetp2
```

### 4.3.1 SignalP/TargetP Exercise

Try using `bash` and NOT visual inspection to answer the following:

2. From the targetp output, how many proteins do we have for each category? Remember, if you are going to grep, be mindful of the 2nd line of the output file.

- Secreted (SP)
- Mitochondrial (mTP)
- Chloroplastic (. . . why?)

3. Compare the confidence scores between those proteins predicted to be secreted by signalP ('SPI' or 'SPII') with their score with targetP. Are there disagreements? Similarities?

# 5 Optional Exercise – eggNOG

THIS EXERCISE IS OPTIONAL

## 5.1 Learning objectives

- Run emapper on the server
- Interpret emapper output
- Understand the power of orthology-based assignments

## 5.2 The eggNOG database & emapper

When you are annotating a genome, it is common to us multiple annotation strategies. So far, we have examined both BLAST and domain-based approaches. Our confidence in these approaches is largely dependent on the quality and contents of the database. The eggNOG database is presently at the forefront of high-quality databases that has been organized based on the relationships among organisms.

> eggNOG (evolutionary genealogy of genes: Non-supervised Orthologous Groups) is a public resource in which thousands of genomes are analyzed at once to establish orthology relationships between all their genes.

For more information see:

- http://eggnog5.embl.de/#/app/methods
- https://academic.oup.com/nar/article/47/D1/D309/5173662
- Refresher on orthologs: https://en.wikipedia.org/wiki/Sequence_homology#Orthology

The developers of this database, have also made an annotation pipeline that allows you to assign a functional annotation to a protein (`emapper`). The software will first match you query sequence to an item in the database to tell you which orthologous group (OG) your protein belongs to, and give you accessory information from other databases based on the annotations known for that OG.

There is incredible flexibility with this tool in terms of taxonomic scope and homology detection methods. To read more check out: https://github.com/eggnogdb/eggnog-mapper/wiki

We're going to annotate the proteome of *Paxillus involutus* in your Annotation folder.

```
# Make a temporary folder in case something goes wrong in your run
mkdir emapper_tmp


# First, we'll need a conda environment to install required librairies
```

```
conda create -n eggnog --file \
/resources/binp28/Programs/eggNOG/newest/requirements.txt \
diamond -y

# don't forget to activate the eggnog env
conda activate eggnog

# then launch emapper
nohup emapper.py -m diamond -i paxillus.faa -o paxillus.emap \
--temp_dir emapper_tmp --cpu 2 2>&1 &

# if you have trouble installing the libs or running the command,
# you can work with pre-ran emapper outputs
cp /resources/binp28/Data/emapper_out/ ./
```

The job will take about 30-60 minutes. The result file `paxillus.emap.annotation` is pretty big, take a look at the file with `less -S`. There is A LOT of information and it might seem overwhelming at first, but there is some very useful information hiding in all this text.

Information about the columns can be found here: https://github.com/eggnogdb/eggnog-mapper/wiki/eggNOG-mapper-v2.1.5-to-v2.1.6#Output_format

For this exercise, we are just going to look at the first 8 columns and column 12; let's `head` the results so we don't flood the screen.

```
head -n 30 paxillus.emap | cut -f 1-8,12
```

Columns information:

- query_name
- seed_eggNOG_ortholog: the best hit against the eggNOG db (taxonID.accession number)
- seed_ortholog_evalue: the evalue of that match
- seed_ortholog_score: the score of that match
- eggNOG OGs: a comma-separated, clade depth-sorted (broadest to narrowest), list of Orthologous Groups (OGs) identified for this query. Note that each OG is represented in the following format: OG@tax_id|tax_name; for any OG, you can take the OG name (before the @) and search for it on http://eggnog5.embl.de/#/app/home
- narr_og_name: OG@tax_id|tax_name for the narrowest OG found for this query.
- narr_og_cat: COG category corresponding to narr_og_name
- narr_og_desc: Description corresponding to narr_og_name
- Preferred_name: preferred name of the protein

Let's look at one particular entry `Paxin1_122960` - remember we'll just need to look at the first 8 columns.

1. What is the tax_name of the most refined OG for this sequence?

2. What is the predicted function of this protein? What is the preferred protein name?
3. Using your results from a previous exercise, where would you expect this protein to function in the cell? Does this align with what is known about this protein (try and find information about it on wikipedia or uniprot)?