amoebae tutorial 2

March 22, 2020

1 Introduction

This tutorial will walk you through a preliminary similarity searching analysis making use of scripts in the AMOEBAE toolkit. While AMOEBAE was not originally written to be used via the command line, Jupyter notebooks provide an easy means of guiding new users through an example analysis with limited need for manual input. The end result of running this code successfully is a spreadsheet summarizing results of similarity searches, as well as a plot to visualize the results.

As a simple example, we will consider the the distribution of orthologues of subunits of the Adaptor Protein (AP) 2 vesicle adaptor complex, and several other membrane-trafficking proteins, in five model eukaryotes: the plant *Arabidopsis thaliana*, the yeast *Saccharomyces cerevisiae*, the fungus *Allomyces macrogynus*, the amoeba *Dictyostelium discoideum*, and the pathogenic protist *Trypanosoma brucei*. AP-2 subunits are homologous to subunits of other AP complexes [6, 3], and published work has traced their evolution among plants [4], Fungi [1], and trypanosomatid parasites [5]. Thus, the protein subunits of the AP-2 complex provide a useful test of similarity searching methods to distinguish between orthologues and paralogues, which can be compared to the results of previous studies. In addition, the membrane trafficking proteins Sec12 (a component of the COPII vesicle coat complex), SNAP33 (a Qbc-SNARE), and Rab2 (a small GTPase) are included to further explore the potential sources of error involved in identification of orthologous proteins.

1.1 Objectives of this tutorial

- Perform similarity searches using the BLASTP, TBLASN, HMMer algorithms simultaneously using AMOBEAE scripts.
- Apply a reciprocal-best-hit search strategy using AMOEBAE code.
- Practice interpreting similarity search results obtained using AMOEBAE.

1.2 Requirements

- Before running this code, you will need to have set up AMOEBAE according to the instructions in the main documentation file here (which you likely have already done): AMOEBAE_documentation.pdf.
- MacOS or Linux operating system (or possibly a work-around on windows, although this has not been tested).
- Approximately 3GB of storage space.
- An internet connection.

- At least an hour of your time (the code in this notebook will take approximately 60 minutes to run).
- Running the code in this notebook is more computationally intensive than webbrowsing for example, so if you are running this on a laptop computer, then make sure it is connected to an electrical outlet.

1.3 Testing

If you wish to simply run all the code in this notebook for testing purposes:

- First, modify the cell in the section labeled "Enter your email to access the NCBI protein database via NCBI Entrez" below such that the value of Entrez.email is hard-coded as your email address.
- Then select "Cell" > "Run All" from the Jupyter menu above.
- Alternatively, close this browser window, navigate to the directory in the container in which this notebook runs, and use the runipy program to run the notebook as follows:

```
runipy -o amoebae_tutorial_2.ipynb
```

2 Preliminary steps

2.1 Check that dependencies are installed

You should have already pulled the amoebae git repository to your computer as described in the main documentation file.

[ ]:
```
%%bash
# This command simply prints the versions of some dependencies which are now␣
 ↪available for use by amoebae.
amoebae check_depend
```

[ ]:
```
%%bash
# This command tests all the import statements in amoebae modules.
amoebae check_imports
```

2.2 Import some basic python modules

[3]:
```
import os
import sys
import platform
import subprocess
from Bio import SeqIO
from Bio import Entrez
import glob
from Bio.Blast import NCBIXML
import pandas as pd
from IPython.display import display, HTML, Image
```

```
sys.path.append('/opt/notebooks')
```

2.3 Record name of this notebook

[4]:
```
%%javascript
// Define relative path to current notebook file.
var nb = IPython.notebook;
var kernel = IPython.notebook.kernel;
var command = "NOTEBOOK_PATH = '" + nb.notebook_path + "'";
kernel.execute(command);
```

```
<IPython.core.display.Javascript object>
```

[5]:
```
# Define path name of current notebook file.
current_notebook = os.path.basename(NOTEBOOK_PATH)
print("Notebook name:\n", current_notebook)
```

```
Notebook name:
 amoebae_tutorial_2.ipynb
```

2.4 Record the specific version of AMOEBAE code used

[6]:
```
# Record git repository version information.
wd = !pwd
script_dir = wd[0]
git_hash = str(subprocess.check_output(["git", "rev-parse", "HEAD"],␣
 ↪cwd=script_dir).strip())
git_branch = str(subprocess.check_output(["git", "rev-parse", "--abbrev-ref",␣
 ↪"HEAD"], cwd=script_dir).strip())
print('\nGit repository (code) version: ' + git_hash + ' (branch name: ' +␣
 ↪git_branch + ')\n')
```

```
Git repository (code) version: b'524a9e3fefea6788b085e102be208ca0ca1eccec'
(branch name: b'master')
```

3 Set up sequence databases for searching

3.1 Download peptide and nucleotide sequences for specific genomes.

Let's download the predicted peptide sequences, genomic assembly (nucleotide sequences of assembled chromosomes), and annotation files (in GFF3 format) for the following eukaryotes from NCBI:

- *Arabidopsis thaliana*
- *Trypanosoma brucei*

- *Dictyostelium discoideum*
- *Allomyces macrogynus*
- *Saccharomyces cerevisiae*

This will take approximately 2 minutes.

[ ]:
```
%%time

# Initiate a list of file paths for downloaded sequence and annotation files.
datafile_path_list = []

# Define a dictionary of source URLs and new filenames for sequence and␣
 ↪annotation files.
# Note that the filenames (besides extension) are the species name with␣
 ↪underscores instead of spaces.
datafile_dict = {"Arabidopsis_thaliana.faa": "ftp://ftp.ncbi.nlm.nih.gov/
 ↪genomes/all/GCF/000/001/735/GCF_000001735.4_TAIR10.1/GCF_000001735.4_TAIR10.
 ↪1_protein.faa.gz",
                 "Arabidopsis_thaliana.fna": "ftp://ftp.ncbi.nlm.nih.gov/
 ↪genomes/all/GCF/000/001/735/GCF_000001735.4_TAIR10.1/GCF_000001735.4_TAIR10.
 ↪1_genomic.fna.gz",
                 "Arabidopsis_thaliana.gff3": "ftp://ftp.ncbi.nlm.nih.gov/
 ↪genomes/all/GCF/000/001/735/GCF_000001735.4_TAIR10.1/GCF_000001735.4_TAIR10.
 ↪1_genomic.gff.gz",
                 "Saccharomyces_cerevisiae.faa": "ftp://ftp.ncbi.nlm.nih.gov/
 ↪genomes/all/GCF/000/146/045/GCF_000146045.2_R64/GCF_000146045.2_R64_protein.
 ↪faa.gz",
                 "Saccharomyces_cerevisiae.fna": "ftp://ftp.ncbi.nlm.nih.gov/
 ↪genomes/all/GCF/000/146/045/GCF_000146045.2_R64/GCF_000146045.2_R64_genomic.
 ↪fna.gz",
                 "Saccharomyces_cerevisiae.gff3": "ftp://ftp.ncbi.nlm.nih.gov/
 ↪genomes/all/GCF/000/146/045/GCF_000146045.2_R64/GCF_000146045.2_R64_genomic.
 ↪gff.gz",
                 "Trypanosoma_brucei.faa": "ftp://ftp.ncbi.nlm.nih.gov/genomes/
 ↪all/GCF/000/210/295/GCF_000210295.1_ASM21029v1/GCF_000210295.
 ↪1_ASM21029v1_protein.faa.gz",
                 "Trypanosoma_brucei.fna": "ftp://ftp.ncbi.nlm.nih.gov/genomes/
 ↪all/GCF/000/210/295/GCF_000210295.1_ASM21029v1/GCF_000210295.
 ↪1_ASM21029v1_genomic.fna.gz",
                 "Trypanosoma_brucei.gff3": "ftp://ftp.ncbi.nlm.nih.gov/genomes/
 ↪all/GCF/000/210/295/GCF_000210295.1_ASM21029v1/GCF_000210295.
 ↪1_ASM21029v1_genomic.gff.gz",
                 "Dictyostelium_discoideum.faa": "ftp://ftp.ncbi.nlm.nih.gov/
 ↪genomes/all/GCF/000/004/695/GCF_000004695.1_dicty_2.7/GCF_000004695.
 ↪1_dicty_2.7_protein.faa.gz",
```

```
                "Dictyostelium_discoideum.fna": "ftp://ftp.ncbi.nlm.nih.gov/
↪genomes/all/GCF/000/004/695/GCF_000004695.1_dicty_2.7/GCF_000004695.
↪1_dicty_2.7_genomic.fna.gz",
                "Dictyostelium_discoideum.gff3": "ftp://ftp.ncbi.nlm.nih.gov/
↪genomes/all/GCF/000/004/695/GCF_000004695.1_dicty_2.7/GCF_000004695.
↪1_dicty_2.7_genomic.gff.gz",
                "Allomyces_macrogynus.faa": "ftp://ftp.ncbi.nlm.nih.gov/
↪genomes/all/GCA/000/151/295/GCA_000151295.1_A_macrogynus_V3/GCA_000151295.
↪1_A_macrogynus_V3_protein.faa.gz",
                "Allomyces_macrogynus.fna": "ftp://ftp.ncbi.nlm.nih.gov/
↪genomes/all/GCA/000/151/295/GCA_000151295.1_A_macrogynus_V3/GCA_000151295.
↪1_A_macrogynus_V3_genomic.fna.gz",
                "Allomyces_macrogynus.gff3": "ftp://ftp.ncbi.nlm.nih.gov/
↪genomes/all/GCA/000/151/295/GCA_000151295.1_A_macrogynus_V3/GCA_000151295.
↪1_A_macrogynus_V3_genomic.gff.gz"
        }

# Make a new temporary directory to store data files.
temp_db_dir_name = 'temporary_db_dir'
if not os.path.isdir(temp_db_dir_name):
    os.mkdir(temp_db_dir_name)

# Download all the data files via NCBI's FTP server.
for filename in datafile_dict.keys():
    url = datafile_dict[filename]
    filepath = os.path.join(temp_db_dir_name, filename)
    if not os.path.isfile(filepath):
        subprocess.call(['curl', url, '--output', filepath + '.gz'])
        subprocess.call(['gunzip', filepath + '.gz'])
```

3.2 Initiate a data directory structure

To generate a directory structure and spreadsheets for storing formatted sequence files and metadata for each sequence file, use the 'mkdatadir' command (this takes a single argument which is the full path that you want your new directory to be written to):

[ ]:
```
%env DATADIR=AMOEBAE_Data
```

[ ]:
```
%%bash
amoebae mkdatadir $DATADIR
```

This will prompt you to set the 'root_amoebae_data_dir' variable in the settings.py file to this new directory path so that AMOEBAE scripts can locate your files.

This can be done as follows:

```
[ ]: # Check that the path indicated in the settings file is correct.
import settings
print(settings.root_amoebae_data_dir)
assert settings.root_amoebae_data_dir == "AMOEBAE_Data"
```

3.3 Prepare databases for searching

To generate a directory structure and spreadsheets for storing formatted sequence files and metadata for each sequence file, use the 'mkdatadir' command (this takes a single argument which is the full path that you want your new directory to be written to).

This will take at least 11 minutes, because the FASTA files need to be re-written with re-formatted sequence headers and the GFF3 files need to be converted to SQL databases using gffutils.

```
[ ]: %%bash
SECONDS=0

for X in temporary_db_dir/*; do amoebae add_to_dbs $X; done

ELAPSED="Preparing sequence databases for searching took the following amount␣
 ↪of time: $(($SECONDS / 3600))hrs $((($SECONDS / 60) % 60))min $(($SECONDS %␣
 ↪60))sec"
echo $ELAPSED
```

```
[ ]: %%bash
# List the databases now accessible by AMOEBAE.
amoebae list_dbs
```

After adding each file to the AMOEBAE_Data directory, a line is added to the 0_genome_info.csv file with information describing this file. Information from this CSV file is used in downstream analysis steps, so it is important to ensure that it is accurate.

You can view the contents of this CSV file here (empty fields are displayed as "NaN"):

```
[ ]: # Find the path to the CSV file.
csv_file_path = os.path.join(os.path.join(os.environ['DATADIR'], 'Genomes'),␣
 ↪'0_genome_info.csv')
# Load data from the CSV file using the pandas library.
df = pd.read_csv(csv_file_path)
# Display the data in an HTML table.
print("Contents of the file %s:" % csv_file_path)
display(HTML(df.to_html()))
```

I recommended that you add any relevant information to complete this table, although this is not necessary to complete this tutorial. This can serve as a useful record for your own reference, as well as as a supplementary file in publications.

By default, AMOEBAE assumes that GFF3 annotation files will be given the same filename (besides the extension) as the nucleotide FASTA file containing the sequences that the annotations are

for. For example, to retrieve annotations for assembled chromosome sequences in a file named "Arabidopsis_thaliana.fna", AMOEBAE will look for a file named "Arabidopsis_thaliana.sql" (which is generated using a file with the name "Arabidopsis_thaliana.gff3). If you wish to use different annotation files, open the CSV file in a spreadsheet program such as Excel or Open Office then copy the name of the .sql file to the row for the corresponding genomic assembly (.fna) file in the column with the header"Annotations file", and do the same for the row describing the corresponding peptide sequence (.faa) file. This allows the correct GFF3 file to be used for the assembly (.fna file) and predicted amino acid sequences (.faa).

Taxonomic information for each genome is arbitrarily divided into four hierarchical categories: "Superbranch", "Supergroup", "Group", and "Species (if applicable)". For files containing sequences or annotations for *Arabidopsis thaliana*, I would enter the values "Diaphoretickes", "Archaeplastida", "Embryophyta", and "Arabidopsis thaliana", respectively. These are arbitrary selected taxonomic groups to which Arabidopsis belongs (Adl et al., 2018). This can be useful when you have many genomes represented in the table.

4 Set up queries

4.1 Enter your email to access the NCBI protein database via NCBI Entrez

[ ]:
```
# Comment out this line and use the line at the bottom of this cell instead, if␣
↪you want to run all cells at once.
Entrez.email = input("Enter your email address here: ")  # Tell NCBI who you␣
↪are.

# Use the line at the top of this cell instead.
#Entrez.email = "yourname@email.com"
```

4.2 Download single-sequence queries

[ ]:
```
%%time

# Define a dictionary with NCBI sequence accessions as keys and filenames to␣
↪write
# the corresponding sequences to as values.
query_dict = {"NP_194077.1": "AP1beta_Athaliana_NP_194077.1_query.faa",
              "NP_851058.1": "AP2alpha_Athaliana_NP_851058.1_query.faa",
              "NP_974895.1": "AP2mu_Athaliana_NP_974895.1_query.faa",
              "NP_175219.1": "AP2sigma_Athaliana_NP_175219.1_query.faa",
              "NP_566961.1": "Sec12_Athaliana_NP_566961.1_query.faa",
              "NP_200929.1": "SNAP33_Athaliana_NP_200929.1_query.faa",
              "NP_193449.1": "Rab2_Athaliana_NP_193449.1_query.faa"
          }

# Make a new temporary directory to store sequence files.
temp_query_dir_name = 'temporary_query_dir'
if not os.path.isdir(temp_query_dir_name):
```

```
    os.mkdir(temp_query_dir_name)

# Loop over keys in the query_dict dictionary.
for accession in query_dict.keys():
    # Retrieve the corresponding filename from the dictionary.
    filename = query_dict[accession]
    filepath = os.path.join(temp_query_dir_name, filename)
    # Only download sequences that have not already been downloaded.
    if not os.path.isfile(filepath):
        # Download the sequence from NCBI via Entrez, using the Biopython␣
 ↪module.
        net_handle = Entrez.efetch(db="protein", id=accession, rettype="fasta",␣
 ↪retmode="text")
        out_handle = open(filepath, "w")
        out_handle.write(net_handle.read())
        out_handle.close()
        net_handle.close()
    # Check that the sequence was actually downloaded.
    assert os.path.isfile(filepath), """The sequence with the following␣
 ↪accession could not be downloaded from NCBI: %s\n
    Try re-running this cell.""" % accession
```

4.3 Prepare single-sequence queries for searching

Queries must be formatted and stored in a similar manner to genomic data files. The query files will include FASTA files containing one sequence and FASTA files containing multiple sequences. Now we are going to generate the query files and add them to your AMOEBAE_Data/ Queries directory, in a similar way to how we added genomic data files to the AMOEBAE_Data/Genomes directory. Since you already downloaded all the peptide sequences for Arabidopsis thaliana, you can retrieve these from your downloaded data using one of the scripts in the amoebae/misc_scripts folder. First, let's generate a query for the A. thaliana AP-1/2 beta subunit(s), which is a component of both the AP-1 and AP-2 complexes, using a representative sequence:

[ ]:
```
%%bash
SECONDS=0

for QUERYFILE in temporary_query_dir/*.faa; do amoebae add_to_queries␣
 ↪$QUERYFILE; done

ELAPSED="Preparing query sequences for searching took the following amount of␣
 ↪time: $(($SECONDS / 3600))hrs $((($SECONDS / 60) % 60))min $(($SECONDS %␣
 ↪60))sec"
echo $ELAPSED
```

[ ]:
```
%%bash
amoebae list_queries
```

4.4 Construct alignments for profile similarity searching

[ ]:
```
%%time

# Define a dictionary of NCBI sequence accessions and filenames to which to␣
 ↪write the corresponding sequences.
query_title_dict = {"AP1beta": "NP_194077.1,CBI34366.3,XP_015631818.
 ↪1,XP_024516549.1,OAE33273.1",
                    "AP2alpha": "NP_851058.1,XP_002270388.1,XP_015631820.
 ↪1,PTQ35247.1,XP_024525508.1",
                    "AP2mu": "NP_974895.1,XP_002281297.1,XP_015627628.
 ↪1,OAE25965.1,XP_002973295.1",
                    "AP2sigma": "NP_175219.1,XP_015618362.1,PTQ50284.
 ↪1,XP_002275803.1,XP_024518676.1",
                    "Sec12": "NP_566961.1,XP_002262948.1,XP_015647566.
 ↪1,OAE21792.1,XP_024530559.1",
                    "SNAP33": "NP_200929.1,XP_002284486.1,AAW82752.1,EFJ31467.
 ↪1,OAE29824.1,XP_006270633.1,XP_006010378.1,XP_006625751.1,NP_001080510.
 ↪1,XP_020370357.1,XP_015181699.1,XP_031769811.1",
                    "Rab2": "NP_193449.1,XP_003635585.2,XP_015626284.
 ↪1,XP_002965710.1,PTQ28228.1"
                   }


# Make a new temporary directory to store sequence files.
temp_alignment_dir_name = 'temporary_alignment_dir'
assert not os.path.isdir(temp_alignment_dir_name), """Directory already exists.
 ↪"""
os.mkdir(temp_alignment_dir_name)

# Download query sequences and write to multiple-sequence FASTA files.
for query_title in query_title_dict.keys():
    accession_list_string = query_title_dict[query_title]
    filepath = os.path.join(temp_alignment_dir_name, query_title + '_hmm1.faa')
    if not os.path.isfile(filepath):
        net_handle = Entrez.efetch(db="protein", id=accession_list_string,␣
 ↪rettype="fasta", retmode="text")
        out_handle = open(filepath, "w")
        out_handle.write(net_handle.read())
        out_handle.close()
        net_handle.close()
```

[ ]:
```
%%bash
SECONDS=0

for X in temporary_alignment_dir/*.faa; do amoebae align_fa $X --output_format␣
 ↪fasta; done
```

```
ELAPSED="Aligning FASTA files took the following amount of time: $(($SECONDS /␣
↪3600))hrs $((($SECONDS / 60) % 60))min $(($SECONDS % 60))sec"
echo $ELAPSED
```

Similar to the AMOEBAE_Data/Genomes/0_genome_info.csv file, the AMOEBAE_Data/Queries/0_query_info.csv file contains information about each query file, which can be manually edited. One of the most important pieces of information is the "Query title". Different query files, such as a single FASTA sequence and an HMM, can have the same Query title if they are to be used to search for homologues or orthologues of the same protein(s).

[ ]:
```
# Find the path to the CSV file.
csv_file_path = os.path.join(os.path.join(os.environ['DATADIR'], 'Queries'),␣
↪'0_query_info.csv')
# Load data from the CSV file using the pandas library.
df = pd.read_csv(csv_file_path)
# Display the data in an HTML table.
print("Contents of the file %s:" % csv_file_path)
display(HTML(df.to_html()))
```

4.5 Visually inspect alignments

Alignments used as queries should be visually inspected to make sure that there are no obvious errors in the alignment.

[ ]:
```
%%bash
for QUERYFILE in temporary_alignment_dir/*.afaa; do amoebae afa_to_nex␣
↪$QUERYFILE; done
echo "Alignments to observe:"
ls temporary_alignment_dir/*.nex
```

4.6 Prepare query alignments for searching

[ ]:
```
%%bash
SECONDS=0

for QUERYFILE in temporary_alignment_dir/*.afaa; do amoebae add_to_queries␣
↪$QUERYFILE; done

ELAPSED="Preparing HMM queries from alignments took the following amount of␣
↪time: $(($SECONDS / 3600))hrs $((($SECONDS / 60) % 60))min $(($SECONDS %␣
↪60))sec"
echo $ELAPSED
```

List queries

[ ]:
```
%%bash
amoebae list_queries
```

4.7 Generate lists of potential redundant sequences among *A. thaliana* peptide sequences

In this tutorial, a reciprocal-best-hit search strategy will be used. If you are using a reciprocal-best-hit search strategy, then your initial round of searches will be performed using your original queries (assembled above) to search your genomes of interest. This initial round of searches will be referred to herein as "forward searches", and subsequent searches using forward search hits as queries into reference genomes will be referred to as "reverse searches".

A slight complication to this search strategy is that the NCBI RefSeq peptide sequences for the *A. thaliana* genome include alternative transcripts and lineage-specific inparalogues (as do other databases), implying that if these were retrieved as the top hits in the reverse searches instead of the original query sequence, then this would still potentially be a positive result. So, to properly interpret reverse search results it will be necessary to determine which sequences in our A. thaliana.faa file are redundant for our purposes. To do this we will use the get_redun_hits command:

[ ]:
```
%%bash
# Optional. Get the help output for the get_redun_hits command.
amoebae get_redun_hits -h
```

[ ]:
```
%env REDUNHITDIR=Redundant_hits
```

This step will take approximately 5 minutes:

[ ]:
```
%%bash
SECONDS=0

# Make a directory to store information about redundant hits.
mkdir $REDUNHITDIR

# Write a file listing names of query files to be used.
amoebae list_queries > $REDUNHITDIR/queries.txt

# Use AMOEBAE to retrieve potential redundant hit sequences.
amoebae get_redun_hits $REDUNHITDIR --query_list_file $REDUNHITDIR/queries.txt␣
↪--db_name Arabidopsis_thaliana.faa

ELAPSED="Retrieving potentially redundant sequences took the following amount␣
↪of time: $(($SECONDS / 3600))hrs $((($SECONDS / 60) % 60))min $(($SECONDS %␣
↪60))sec"
echo $ELAPSED
```

This will output a directory in the Redundant_hits folder with a .csv file. This file contains a summary of BLASTP or HMMer search results for searches with the specified queries into the *A. thaliana* predicted proteins.

It should be apparent upon inspection of the ranking of hits and comparison of the associated E-values which hits are likely redundant with your queries (see cell below). To tell AMOEBAE which hits you want to consider as redundant for the purposes of downstream steps, the values in the column with the header "Positive/redundant (+) or negative (-) hit for queries with query title (edit this column)" must be changed from '-' to '+' for hits that are redundant (this will be done automatically for this tutorial (see below).

[ ]:
```
# Find the path to the CSV file.
csv_file_path = glob.glob(os.path.join('Redundant_hits', os.path.
 ↪join('redun_hits_*', '0_redun_hits_*.csv')))[0]
# Load data from the CSV file using the pandas library.
df = pd.read_csv(csv_file_path)
# Display the data in an HTML table.
print("Contents of the file %s:" % csv_file_path)
display(HTML(df.to_html()))
```

4.8 Identify redundant sequences

[ ]:
```
%time
# Define a dictionary with query titles as keys and lists of sequence IDs as␣
 ↪values, where the IDs are for A. thaliana sequences that are redundant with␣
 ↪the original A. thaliana query sequence.
redun_seq_dict = {"AP1beta":  ["NP_194077.1",
                               "NP_192877.1",
                               "NP_001328014.1",
                               "NP_001190701.1"
                               ],

                  "AP2alpha": ["NP_851058.1",
                               "NP_851057.1",
                               "NP_197669.1",
                               "NP_001330971.1",
                               "NP_001330970.1",
                               "NP_001330969.1",
                               "NP_197670.1",
                               "NP_001330127.1"
                               ],

                  "AP2mu":    ["NP_974895.1",
                               "NP_199475.1"
                               ],

                  "AP2sigma": ["NP_175219.1"
                               ],

                  "Sec12":    ["NP_566961.1",
                               "NP_568738.1",
```

```
                                "NP_680414.1",
                                "NP_178256.1"
                                ],

                    "SNAP33":   ["NP_200929.1",
                                "NP_001332102.1",
                                "NP_172842.1",
                                "NP_001318998.1",
                                "NP_196405.1",
                                "NP_001318503.1"
                                ],

                    "Rab2":     ["NP_193449.1",
                                "NP_193450.1",
                                "NP_195311.1",
                                "NP_001078499.1"
                                ]
                     }


# Identify path to redundant seqs CSV file.
redundant_seqs_csv = glob.glob(os.path.join('Redundant_hits', os.path.
 ↪join('redun_hits_*', '0_redun_hits_*.csv')))[0]

# Define path for new modified redundant seqs CSV file.
redundant_seqs_csv2 = redundant_seqs_csv.rsplit(".", 1)[0] + '_2.csv'

# Open the redundant seqs CSV file, and a new one.
with open(redundant_seqs_csv) as infh, open(redundant_seqs_csv2, 'w') as o:
    # Loop over lines in the CSV file.
    for i in infh:
        if not i.startswith("Query Title"):
            # Identify query title in line.
            line_query_title = i.split(',')[0].strip()
            # Identify accession/id for sequence hit represented in this row.
            line_accession = i.split(',')[9].strip().strip('\"')
            # Loop over keys (query titles) in the redundant seqs dictionary.
            query_title_in_keys = False
            for query_title in redun_seq_dict.keys():
                if line_query_title == query_title:
                    query_title_in_keys = True
                    #print('YYY')
                    #print(line_accession)
                    #print(redun_seq_dict[line_query_title])
                    # Determine whether the accession is a redundant accession.
                    for acc in redun_seq_dict[line_query_title]:
                        #print(line_accession, acc)
```

```
                        if line_accession == acc:
                            # Change the - to + so that the accession will be␣
↪included in the list of redundant accessions used by AMOEBAE.
                            i = ','.join(i.split(',')[:4]) + ',+,' + ','.join(i.
↪split(',')[5:])
                            break
                    # Break loop if the corresponding query title was found already.
                    if query_title_in_keys:
                        break
                # Check that a query title could be recognized as one that is a key␣
↪in the dictionary.
                assert query_title_in_keys, """Could not find query title %s in␣
↪dictionary.""" % line_query_title
            # Write (modified) line to new CSV file.
            o.write(i)
```

Observe how the file has been modified.

[ ]:
```
# Find the path to the CSV file.
csv_file_path = glob.glob(os.path.join('Redundant_hits', os.path.
 ↪join('redun_hits_*', '0_redun_hits_*2.csv')))[0]
# Load data from the CSV file using the pandas library.
df = pd.read_csv(csv_file_path)
# Display the data in an HTML table.
print("Contents of the file %s:" % csv_file_path)
display(HTML(df.to_html()))
```

5 Run forward searches

To begin searching, make a new folder to contain search results, and write text files listing the names (not full paths) of FASTA files you want to use as queries and those that you want to search in.

[ ]:
```
%env SRCHRESDIR=AMOEBAE_Search_Results_1
```

[ ]:
```
%%bash
# Make a new directory to contain search results.
mkdir $SRCHRESDIR
# Write query and database list files.
amoebae list_queries > $SRCHRESDIR/queries.txt
amoebae list_dbs > $SRCHRESDIR/databases.txt
```

Set up searches using the setup_fwd_srch command:

[ ]:
```
%%bash
# Optional. Get the help output for the setup_fwd_srch command.
amoebae setup_fwd_srch -h
```

[ ]:
```
%env FWDSRCHDIR=fwd_srch_1
```

[ ]:
```
%%bash
# Set up forward searches.
amoebae setup_fwd_srch $SRCHRESDIR\
                       $SRCHRESDIR/queries.txt\
                       $SRCHRESDIR/databases.txt\
                       --outdir $SRCHRESDIR/$FWDSRCHDIR
```

This will output a new sub-directory with a name that starts with "fwd_srch_". Now run the searches with this directory as input via the run_fwd_srch command. Forward search criteria may be selected at this point (view the relevant optional arguments via the -h option).

[ ]:
```
%%bash
tree $SRCHRESDIR
```

[ ]:
```
%%bash
SECONDS=0

# Run forward searches. This could take a while.
amoebae run_fwd_srch $SRCHRESDIR/$FWDSRCHDIR

ELAPSED="Running forward searches took the following amount of time:␣
 ↪$(($SECONDS / 3600))hrs $((($SECONDS / 60) % 60))min $(($SECONDS % 60))sec"
echo $ELAPSED
```

This will run BLASTP or HMMer for searches into the .faa files (depending on whether queries are single- or multi-fasta), or TBLASTN for searches into the .fna files with any single-fasta queries.

6 Summarize forward search results

Now we can generate a summary of the raw output files. Important criteria may be customized here as well. Specifically the forward search E-value threshold, and the maximum number of nucleotide bases allowed between TBLASTN HSPs to be considered part of the same gene (view optional arguments via the -h option).

[ ]:
```
%%bash
amoebae sum_fwd_srch -h
```

[ ]:
```
%%time
# Summarize forward search results in a CSV file.
# ***Note that only the top 5 hits for each individual search will be reported,␣
 ↪as specified here.
# This is simply to save time, and previous analyses have confirmed that the␣
 ↪number of positive hits will not exceed 5 for any of the searches.
!amoebae sum_fwd_srch $SRCHRESDIR/$FWDSRCHDIR\
                      $SRCHRESDIR/$FWDSRCHDIR'_sum.csv'\
```

```
                    --max_hits_to_sum 5
```

Examine the resulting CSV file. Note that maximum E-value cutoffs, and other criteria were applied as specified.

[ ]:
```
# Load data from the CSV file using the pandas library.
df = pd.read_csv(os.path.join(os.environ['SRCHRESDIR'],os.
 ↪environ['FWDSRCHDIR']) + '_sum.csv_out.csv')
# Display the data in an HTML table.
display(HTML(df.to_html()))
```

Subsequent steps in this tutorial will result in copies of this spreadsheet with appended columns to describe further results relevant to each forward search hit.

7 Run reverse searches

Now, to determine which of the "forward hits" in these search results are really specific to our original A. thaliana queries, let's search with these hits as queries back into the A. thaliana genome (i.e., perform "reverse" searches).

Similar to the forward searches, we need to first set up the reverse search directory:

[ ]:
```
%env REVSRCHDIR=rev_srch_1
```

[ ]:
```
%%bash
amoebae setup_rev_srch -h
```

[ ]:
```
%%bash
# Important: the --aasubseq option is used here.
amoebae setup_rev_srch $SRCHRESDIR\
                       $SRCHRESDIR/$FWDSRCHDIR'_sum.csv_out.csv'\
                       Arabidopsis_thaliana.faa\
                       --outdir $SRCHRESDIR/$REVSRCHDIR\
                       --aasubseq
```

This will output a new directory with "rev_srch_" and a timestamp in the name. Run reverse searches using the path to this directory as an input:

[ ]:
```
%%bash
# View reverse search directory contents.
tree $SRCHRESDIR/$REVSRCHDIR
```

Running reverse searches will take approximately 5 minutes.

[ ]:
```
%%time
!amoebae run_rev_srch $SRCHRESDIR/$REVSRCHDIR
```

8 Summarize reverse search results

Now append columns summarizing the results of these reverse searches to our CSV file. This is where the file listing redundant hits for each query title is used. Also, a criterion is applied here based on the order of magnitude difference in E-value between the original query (or redundant hits) in the reverse search results compared to other hits (if present), and this can be optionally modified (view optional arguments via the -h option).

This could take a while.

[ ]:
```
%%bash
amoebae sum_rev_srch -h
```

Summarizing reverse search results should take approximately 5 minutes.

[ ]:
```
%%bash
SECONDS=0

# Important: The --aasubseq option is used here, because it was used when the␣
 ↪setup_rev_srch command was run above.
CSVLIST=($REDUNHITDIR/redun_hits_*/0_redun_hits_*_2.csv)
amoebae sum_rev_srch $SRCHRESDIR/$FWDSRCHDIR'_sum.csv_out.csv'\
                     $SRCHRESDIR/$REVSRCHDIR\
                     --redun_hit_csv ${CSVLIST[-1]}\
                     --aasubseq\
                     --min_evaldiff 2


ELAPSED="Summarizing these results took the following amount of time:␣
 ↪$(($SECONDS / 3600))hrs $((($SECONDS / 60) % 60))min $(($SECONDS % 60))sec"
echo $ELAPSED
```

By default, this will output a CSV file with the same path as the forward search summary CSV file, but with a "_1" added before the filename extension. Examine the resulting CSV file. You could run additional reverse searches into different files, appending columns to the same summary spreadsheet. Reverse searches into the A. thaliana peptide sequences is all that is necessary for this tutorial.

Next run the interp_srchs command to do an additional interpretation of the results (if reverse searches into multiple reference databases were performed then this would be done following summarization of all the reverse searches). Again, customized criteria may be applied at this point using the optional arguments.

[ ]:
```
%%bash
amoebae interp_srchs $SRCHRESDIR/$FWDSRCHDIR'_sum.csv_out_1.csv'
```

Again, examine the resulting CSV file to see whether the results match your expectations. You will notice that the results in this file do not account for the fact that the HMMer, BLASTP, and TBLASTN hits are redundant in many cases as might be expected if each of these search algorithms were effective.

9 Determine which positive hits are redundant

We need to determine which hits likely correspond to the same loci based on having identical accessions or being associated with the same locus in the GFF3 annotation file, or likely represent distinct paralogous gene loci based on sequence similarity in a multiple sequence alignment (see Larson et al. (2019) for explanation of how these are identified). To do this, first we will append a column listing what alignment to use (by default it will be the alignments that are used as queries for the corresponding query title):

[ ]:
```
%%bash
amoebae find_redun_seqs -h
```

[ ]:
```
%%bash
CSVLIST=( $SRCHRESDIR/${FWDSRCHDIR}_sum.csv_out_1_interp_*.csv )
amoebae find_redun_seqs ${CSVLIST[-1]} --add_ali_col
```

Now identify distinct paralogues. This should take about 30 minutes.

[ ]:
```
%%bash
SECONDS=0

CSVLIST=( $SRCHRESDIR/${FWDSRCHDIR}_sum.csv_out_1_interp_*_with_ali_col.csv )

amoebae find_redun_seqs ${CSVLIST[-1]}

ELAPSED="Finding redundant sequences took the following amount of time:␣
↪$(($SECONDS / 3600))hrs $((($SECONDS / 60) % 60))min $(($SECONDS % 60))sec"
echo $ELAPSED
```

This will output another copy of the CSV file with additional columns. Take some time to decide whether you agree with the exclusion of some of the hits, as indicated in the appended columns. You may wish to include an abridged version of this type of final output table as a supplementary file for publications.

[ ]:
```
# Load data from the CSV file using the pandas library.
csv_file = glob.glob(os.path.join(os.environ['SRCHRESDIR'],'*_paralogue_count_*.
↪csv'))[0]
df = pd.read_csv(csv_file)
# Display the data in an HTML table.
display(HTML(df.to_html()))
```

10 Plot the final search results

Finally, we can plot the results of the searches. To customize the organization of the output coulson plot, an additional input CSV file may be optionally provided here. This file simply contains the names of protein complexes in the first column and query titles for proteins that you want to include in each complex in the second column (see example file provided with this tutorial).

[ ]:
```
%%bash
# Write a file indicating how columns in the output coulson plot should be␣
 ↪constructed.
echo \
"AP-2,AP1beta
AP-2,AP2alpha
AP-2,AP2mu
AP-2,AP2sigma
COPII,Sec12
SNAREs,SNAP33
Rabs,Rab2" > $SRCHRESDIR/complex_info_1.csv
```

[ ]:
```
# Write a file indicating the order in which results should be displayed.
with open(os.path.join(os.environ['SRCHRESDIR'], "databases.txt")) as infh,\
open(os.path.join(os.environ['SRCHRESDIR'], "coulson_row_order.txt"), 'w') as o:
    lines = infh.readlines()
    order_list = ["Arabidopsis",
                  "Saccharomyces",
                  "Allomyces",
                  "Dictyostelium",
                  "Trypanosoma"
                 ]
    for genus in order_list:
        for line in lines:
            if line.startswith(genus):
                o.write(line)
```

[ ]:
```
%%bash
amoebae plot -h
```

[ ]:
```
%%bash
SECONDS=0

CSVLIST=( $SRCHRESDIR/${FWDSRCHDIR}_sum.
 ↪csv_out_1_interp_*_with_ali_col_paralogue_count_*.csv )
amoebae plot ${CSVLIST[-1]}\
             --complex_info $SRCHRESDIR/complex_info_1.csv\
             --row_order $SRCHRESDIR/coulson_row_order.txt\
             --out_pdf $SRCHRESDIR/plot.pdf

ELAPSED="Plotting these results took the following amount of time: $(($SECONDS /
 ↪ 3600))hrs $((($SECONDS / 60) % 60))min $(($SECONDS % 60))sec"
echo $ELAPSED
```

Examine the resulting PDF files. Your coulson plot should look something like that in Figure 1. Compare with the results of searches for AP-2 subunits published by Manna et al. (2013), Barlow et al. (2014), and Larson et al. (2019). You will need to customize formatting of coulson plots

output by the 'plot' command using software such as Adobe Illustrator.

```
[ ]: Image(filename=os.path.join(os.environ['SRCHRESDIR'], "plot_coulson_both.png"),␣
     ↪width="600px")
```

Figure 1: A coulson plot summarizing similarity search results for AP-2 complex subunits in *Trypanosoma brucei gambiense* and *Saccharomyces cerevisiae* peptide and nucleotide sequences using *Arabidopsis thaliana* queries and Hidden Markov Models generated from alignments of embryophyte orthologues. BLASTP and TBLASTN were used to search peptide and nucleotide sequences, respectively, with single sequence queries, and the HMMer3 pack- age was used to perform profile searches. Subplot sectors with blue fill indicate that one or more sequences were found to meet the search criteria applied (with the number being indicated within each subplot sector). Note that the ancestral eukaryotic AP-1 and AP-2 complexes shared a single beta subunit [2]. This is why identified "AP1beta" orthologues are shown as a component of the AP-2 complex here, even though *T. brucei* lacks an AP-2 complex [5]. These results are comparable to the relevant results published by [5], [1], and [4].

11 Interpretation and re-analysis

It should be clear that AMOEBAE identifies "positive" and "negative" results simply by applying criteria that the user specifies. So, it is entirely the users responsibility to select appropriate criteria and interpret the results critically.

Points to consider regarding interpretation of the results of the analysis in this tutorial include the following:

- The BLASTP and HMMer searches (both followed by reverse BLASTP searches) yielded the similar results in this analysis.
- The TBLASTN searches were able to identify most of the genes represented by the peptide sequences identified by BLASTP and HMMer searches.
- A TBLASTN hit in the *A. thaliana* chromosome 5 (NC_003076.8) met the forward and reverse search criteria, but was excluded because the translation of the region that aligned to the query was only 50 amino acids long (this sequence also contained stop codons). If you look on the NCBI genome browser for *A. thaliana* you will see that this region on chromosome 5 (as indicated in the summary CSV file) corresponds to a pseudogene for AP-2 sigma with the gene ID AT5G42568.
- The two *A. thaliana* AP-1/2 beta paralogues and the two *S. cerevisiae* paralogues are brassicalid and fungal inparalogues, respectively, which arose from independent gene duplications. Phylogenetic analysis would be required to determine this (see Larson et al. (2019) and Barlow et al. (2014)).
- An *Arabidopsis thaliana* AP-2 mu splice variant was excluded after running the 'find_redun_seqs' command, because it was found to be encoded by the same gene as the other splice variant based on information in the GFF3 annotation file.
- An *A. thaliana* AP-2 alpha gene was excluded after running the 'find_redun_seqs' command, because it shows over 98% identity with the other AP-2 alpha gene. The summary CSV file

indicates which file contains an alignment of these two sequences (see Larson et al. (2019) for relevant discussion).

- The results for Rab2 illustrate a limitation of the type of reciprocal-best-hit search strategy employed in this tutorial. A positive hit for Rab2 was identified in *Allomyces macrogynus* using these methods. However, Elias et al. (2012) did not identify a Rab2 orthologue *A. macrogynus* in their comprehensive analysis. The positive hit in the analysis herein is a false positive result that occurs due to the absence of Rab4 (a close relative of Rab2) in *A. thaliana*. This example highlights the importance of following up similarity searching with phylogenetic analysis, which compares identified sequences to many homologues simultaneously.

If the analysis in this tutorial were a project you were working on for publication, then upon completing the above analysis steps you work would have only just begun. Careful inspection of the summary CSV file will reveal that minor adjustments to the search criteria would cause the analysis to yield different results. Moreover, there are many different possibilities that would lead to innacurate results based on the criteria applied in the above analysis. A comprehensive discussion of this is beyond the scope of this tutorial. In general, I recommend that you take an iterative approach to analysis involving adjustment of search criteria and re-analysis to include sequences that you know are homologues of interest, but to exclude those that you know are not homologues of interest.

To generate an alignment of homologous sequences identified using AMOEBAE, use the 'csv_to_fasta' command to generate FASTA files for alignment, and then align using your preferred software (e.g., MUSCLE or MAFFT). For visually assessing the sequences for possible issues such as contrasting domain topologies, you may wish to generate FASTA files including all your forward search results for each query title. If you are planning to run a phylognetic analysis, you may wish to generate a FASTA file with only those sequences that match all your search criteria, and with abbreviated sequence headers the csv_to_fasta comand as follows:

[ ]:
```
%%bash
CSVLIST=( $SRCHRESDIR/${FWDSRCHDIR}_sum.
 ↪csv_out_1_interp_*_with_ali_col_paralogue_count_*.csv )

amoebae csv_to_fasta ${CSVLIST[-1]} --abbrev --split_by_query_title
```

12 Delete search output files (optional)

[ ]:
```
%%bash
# Delete temporary files.
#rm -r temporary_alignment_dir
#rm -r temporary_db_dir
#rm -r temporary_query_dir
```

[ ]:
```
%%bash
# Delete all data and results files (WARNING you may want to keep these!).
#rm -r $DATADIR
#rm -r $SRCHRESDIR
#rm -r $REVSRCHDIR
```

13 Retrieve bibliographic information for citations

You can either generate a bibtex (.bib) file with relevant reference information using a citation manager such as Zotero, or generate such a file from a list of digital object identifier (DOI) numbers as in the below two cells:

[1]:
```
# Define a list of DOI numbers.
doi_numbers =\
"""
10.1073/pnas.0707318105
10.4161/cl.28114
10.1242/jcs.101378
10.1371/journal.pbio.1001170
10.1111/tra.12698
10.1016/j.ympev.2013.01.002
10.1016/j.tcb.2004.02.002
""".strip().split('\n')
```

Now retrieve the bibliographical information for these works.

[16]:
```
import urllib.request
from urllib.error import HTTPError
import bibtexparser

# Retrieve bibliographical information in bibtex format.
base_url = 'http://dx.doi.org/'
bibtex_string = ''
for doi in doi_numbers:
    url = base_url + doi
    req = urllib.request.Request(url)
    req.add_header('Accept', 'application/x-bibtex')
    try:
        with urllib.request.urlopen(req) as f:
            bibtex = f.read().decode()
        #print(bibtex)
        for i in bibtexparser.loads(bibtex).entries:
            print(i['ID'])
        bibtex_string = bibtex_string + bibtex
    except HTTPError as e:
        print(doi)
        if e.code == 404:
            print('DOI not found.')
        else:
            print('Service unavailable.')

# Write bibtex file.
bibtex_file_path = current_notebook.rsplit('.', 1)[0] + '.bib'
with open(bibtex_file_path, 'w') as o:
```

```
    o.write(bibtex_string)
    print("\nBibliographical information written to file: %s" %␣
 ↪bibtex_file_path)
```

```
Dacks_2008
Barlow_2014
Elias_2012
Hirst_2011
Larson_2019
Manna_2013
Robinson_2004

Bibliographical information written to file: amoebae_tutorial_2.bib
```

14 Print this notebook

This can be done at any stage; you don't need to run any of the above code first.

First, provide a title and author name for this notebook:

[17]:
```
# Define author name for this notebook.
author_name = ""

# Define title of this notebook.
notebook_title = current_notebook.rsplit('.', 1)[0].replace('_', ' ') # Or,␣
 ↪write your own.
```

Save and checkpoint the current notebook:

[18]:
```
%%javascript
// Save and checkpoint the current notebook (same as doing it manually through␣
 ↪the GUI).
require(["base/js/namespace"],function(Jupyter) {
    Jupyter.notebook.save_checkpoint();
});
```

```
<IPython.core.display.Javascript object>
```

Now write this notebook to a PDF file (with formatted citations):

[15]:
```
# Import modules.
import os
from string import Template

# Check that a file containing citation information in bibtex format exists.
bibtex_file_path = current_notebook.rsplit('.', 1)[0] + '.bib'
```

```
assert os.path.isfile(bibtex_file_path), """A file containing citation␣
 ↪information could not be found."""

# Write a latex template file for converting this notebook to latex (as an␣
 ↪intermediate to PDF).
latex_template_string = Template(r"""
((*- extends 'article.tplx' -*))

((* block author *))
\author{$an}
((* endblock author *))

((* block title *))
\title{$nt}
((* endblock title *))

((* block bibliography *))
\bibliographystyle{plain}
\bibliography{$rf}
((* endblock bibliography *))
""")
latex_file_contents =\
latex_template_string.substitute(an=author_name,
                                 nt=notebook_title,
                                 rf=bibtex_file_path
                                )
latex_template_file_path = 'latex_template.tplx'
with open(latex_template_file_path, 'w') as o:
    o.write(latex_file_contents)

# Convert notebook to PDF (with latex as an intermediate to process bibtex␣
 ↪citations, etc.).
!jupyter nbconvert {current_notebook} --to pdf --template␣
 ↪{latex_template_file_path}

# Remove latex template file and bibtex file.
os.remove(latex_template_file_path)
os.remove(bibtex_file_path)
```

```
      ␣
↪---------------------------------------------------------------------------

       AssertionError                                Traceback (most recent call␣
↪last)

       <ipython-input-15-2feaef907b3b> in <module>
```

```
        5 # Check that a file containing citation information in bibtex format␣
↪exists.
        6 bibtex_file_path = current_notebook.rsplit('.', 1)[0] + '.bib'
   ----> 7 assert os.path.isfile(bibtex_file_path), """A file containing␣
↪citation information could not be found."""
        8
        9 # Write a latex template file for converting this notebook to latex␣
↪(as an intermediate to PDF).


      AssertionError: A file containing citation information could not be␣
↪found.
```

15 Where to go from here?

First, try modifying the parameters and observe how that changes the results. Then customize this notebook to search with different queries in different genomes, or run amoebae from the command line.

References

[1] Lael D Barlow, Joel B Dacks, and Jeremy G Wideman. From all to (nearly) none. *Cellular Logistics*, 4(1):e28114, jan 2014.

[2] J. B. Dacks, P. P. Poon, and M. C. Field. Phylogeny of endocytic components yields insight into the process of nonendosymbiotic organelle evolution. *Proceedings of the National Academy of Sciences*, 105(2):588–593, jan 2008.

[3] Jennifer Hirst, Lael D. Barlow, Gabriel Casey Francisco, Daniela A. Sahlender, Matthew N. J. Seaman, Joel B. Dacks, and Margaret S. Robinson. The fifth adaptor protein complex. *PLoS Biology*, 9(10):e1001170, oct 2011.

[4] Raegan T. Larson, Joel B. Dacks, and Lael D. Barlow. Recent gene duplications dominate evolutionary dynamics of adaptor protein complex subunits in embryophytes. *Traffic*, 20(12):961–973, oct 2019.

[5] Paul T. Manna, Steven Kelly, and Mark C. Field. Adaptin evolution in kinetoplastids and emergence of the variant surface glycoprotein coat in african trypanosomatids. *Molecular Phylogenetics and Evolution*, 67(1):123–128, apr 2013.

[6] Margaret S. Robinson. Adaptable adaptors for coated vesicles. *Trends in Cell Biology*, 14(4):167–174, apr 2004.