Mar 21, 2019

Working

SYSB 3036 W07: Pan-genomics

Version 2

Frank Aylward[1]

[1]Virginia Tech

dx.doi.org/10.17504/protocols.io.zekf3cw

Frank Aylward
Virginia Tech

ABSTRACT

PROTOCOL STATUS

Working

We use this protocol in our group and it is working

SAFETY WARNINGS

Get the data

1 First we need to get some data to start with. I have already prepared some starting files and put them on a GitHub repository, so we can download it using the following command:

git clone https://github.com/faylward/pangenomics_tutorial

After this command finishes you should see a new folder called "pangenomics_tutorial", and inside of this folder you should see a file called "micropan-source.R" and a folder with 4 gzipped .faa files (.faa.gz).

You can check with:

cd pangenomics_tutorial

and then:

ls -la

The "micropan-source.R" file contains source code written in the R language that we will use later in the tutorial. No need to do anything with it just yet.

The four .faa.gz files correspond to the proteins encoded in 4 Chlamydia pneumoniae genomes that I downloaded from NCBI RefSeq. I chose these genomes since Chlamydia pneumonieae genomes are relatively small (~1 Mbp) and encoded only ~1,000 genomes, so calculating orthologous groups between 4 genomes should not take too long. Calculating orthologous groups gets quickly becomes computationally intensive as we add more genomes, so using smaller genomes should save us a bit of time. Also, Chlamydia pneumoniae is a fascinating pathogen that causes lung infections, and so it's an interesting bug to study here (for more information see the CDC page: https://www.cdc.gov/pneumonia/atypical/cpneumoniae/index.html)

Unzip the .faa.gz files and run proteinortho to get orthologous groups

2 First we need to unzip the .faa files in the fasta/ folder so we can begin to analyze the protein files.
If you are in the folder pangenomics_tutorial you can run the command:

gunzip fasta/*.faa.gz

and then check with:

ls -la fasta

Run Proteinortho to get orthologous groups

3 Here we will run a similar command to the one we used in the "orthologous groups" tutorial, only this time we will be including 5 genomes instead of 2 or 3. You can modify the "-cpus" flag depending on how many cores you would like to use. Using 2 cores this should take about 2 minutes to complete.

proteinortho5 -project=cp_pangenome -cpus=2 -singles fasta/*.faa

And of course after this we should use the "ls" command to ensure that the appropriate new files were created:

ls -la

Get orthologous group statistics

4 The main output file we want to work with from Proteinortho is the ".proteinortho" file. Use "head" and "tail" to take a look at this file. The lines are a bit long, here is the general format:

- Each row (aside from the header) has information for one protein cluster (or protein family- both terms are equivalent here).

- Each line is tab-delimited, and the first three columns have information about that particular cluster.

- The first column provides the # of species in which a cluster is present. Here a 5 would mean the protein cluster is found in every genome.

- The second column provides the number of proteins that are present. This can be different than the first column since one genome can have multiple members of a given protein cluster.

- The third column has the Algebraic connectedness. We won't worry about this here, but this gives some information about how the proteins are distributed in the different genomes.

- After that, the number of columns depends on the number of genomes that we analyzed. Since we have 5 genomes there will be 5 more columns (for a total of 8). Each of these next columns just has the names of the proteins that belong to a cluster, or a * if no proteins for that cluster were found.

If we want to know how many protein clusters were found, we can just use "wc" and subtract one from the line count (since one line is the header).

wc -l cp_pangenome.proteinortho

I got 1197 total lines, so 1196 total protein clusters.

5 Now we can start looking through the .proteinortho file to get some idea of what the pan-genome looks like. This will tell us how many proteins are shared between different genomes, and how many unique proteins each genome has.

Let's say I want to know how many proteins were found in all genomes exactly one time. For this I can search for "5\5" since that should match only to the first two columns. We need to use the "-P" flag here to make sure the whitespace is matched.

cut -f 1-2 cp_pangenome.proteinortho | grep -P "5\t5" | wc | wc -l

And if we want to find how many proteins were present only once in one genome, we can use:

cut -f 1-2 cp_pangenome.proteinortho | grep -P "1\t1" | wc | wc -l

I got 986 protein clusters shared between 5 genomes and only 117 singleton clusters. This is out of 1197 total. So the majority of the proteins encoded in these genomes are shared between all 5 genomes, indicating a rather restricted pan-genome (i.e., not a huge amount of

variability).

Start an R session and load in the micropan source code

6 For the next few steps we will be operating in the R programming language. To do this we can start R and then continue working in the command line, only this time we will need to use R command rather than Unix commands.

The reason we are doing this is that there is a very nice R package called "micropan" for pan-genomics analysis, and it is written in R.

To start an R session simply type:

R

And you should see an R console open up with a welcome message, and with the cursor now starting with a ">" symbol.

Load in the micropan source code

7 Since we need to use code that is part of the micropan R package, we need to load this code into our console before continuing.
To do this we can use the R command "source" and specify the file "micropan-source.R" that was in the "pangenomics-tutorial" folder. For me this would look like:

source("week7/pangenomics_tutorial/micropan-source.R")

You will need to change this depending on what the PATH to the micorpan-source.R file looks like. Make sure there are no spaces in your folder names- R will not like this.

Load the .proteinortho file into R

8

Now that we are in R and have the micropan code loaded into our session, we will need to load in the results we got from Proteinortho so we can start analyzing them. For this we will create an R DataFrame simply "x" that will contain the data from the "cp_pangenome.proteinortho" file.

x <- read.table("week7/pangenomics_tutorial/cp_pangenome.proteinortho", header=T, sep="\t")

A few notes:
- Here "<-" is essentially a fancy "=" that is used in R.
- "read.table" is the R function for loading data.
- header=T
- sep="\t" specifies that the file is tab-delimited ("\t" is a general expression for tabs).

Now in addition to loading the data into R we also need to transform it into a "pangenome matrix" object that is more easily used by micropan. You will see things like this a lot in R, since different packages need data to be organized in a certain way. We can do this with the simple command:

panmat <- as.panmat(x)

And now we have our data stored as a pangenome matrix in the "panmat" object.

Summarize the data

9

Now we can start using micropan functions to analyze the data.

A simple function to start with is "summary":

summary(panmat)

This should provide statistics that we already received in Step 4. It's always good to verify that they are the same.

Generate a rarefaction curve

10

A standard pan-genomic analysis is to create a rarefaction curve of new proteins identified vs # of genomes used.

The general idea behind a rarefaction curve is to see how many new genes we find every time we add a new genome. The rate at which new genes are identified every time we add a new genome tells us something about how variable the gene content is between the genomes we are analyzing.

Rarefaction curves generally taper off as the x-axis (number of genomes) increases, telling us that sampling a certain number of genomes is sufficient to sample nearly all of the genes in a species. However, sometimes the rarefaction curve continues to increase, telling us that possibly no number of genomes will give us a good enough sampling of all the genes present in that group. The genetic diversity of prokaryotes is pretty amazingly high, so this is not necessarily a surprising result.

To calculate the rarefaction curve let's use the following code:

rarefy <- rarefaction(panmat, n.perm=100)

To get a summary of the rarefaction results we can type:

summary(rarefy)

11 Lastly, it is always nice to get a graph of the rarefaction curve so we can visualize the results.
Generating a plot in R is a 3-step process.

First we need to generate a file that the plot will be written to. Here let's try to create a JPEG file:

jpeg("rarefaction.jpg")

Then we need to plot the graph. Here I will simply use the "plot" function and set the y-axis limits to be from 0 to the max rarefaction value:

plot(rarefy, ylim=c(0, max(rarefy)))

And then we need to close the graphical device so that R knows we are finished plotting to that file:

dev.off()

Depending on your system you could just use the "plot" command above, and a new window with the plot may upen up. However, to save the plot as an image file you will still need to use the "jpeg" and "dev.off()" commands.

12 In addition to rarefaction curves there are a few other ways we can analyze the pan-genome data.

For example we can also get the "Chao" statistic for a pan-genome, which is the estimated number of genes we would have if we could sequence infinitely many genomes (this is like estimating where the rarefaction curve eventually levels off if we follow the rarefaction curve as far to the right as we can).
in R we can calculate this easily with:

chao(panmat)

Here I got 1684, which seems about right given we have just under 1200 genes with only 5 genomes.

13 Lastly, we can calculate a statistics by fitting the data to Heap's Law, which was initially formulated to estimate the number of distinct words in a text document. Here we can use it to get an estimate for how diverse our pan-genome is.

A value of alpha > 1 is used to describe "closed" pan-genomes, or those in which we do not see that many unique genes in genomes. An "open" pan-genome has an alpha < 1 and is used to describe highly variable pan-genomes in which each genome has many unique genes. We can calcualte alpha with:

heap <- heaps(panmat, 1000)

I generally use a large value for the number of permuatations to use for estimating this paramter (~1000). I've noticed the values can be quite variable if a smaller number of permutations is used.

Here I got a value of ~0.75. So the Chlamydia genomes still technically have "open" pangenomes, even though their core genome is much larger than their variable genome.