CSCI 4181 / CSCI 6802 Assignment 2

Genome Assembly (February 16, 2023)

This is the second of four assignments in the course that will give you practical experience with biological data. The deadline for this assignment is **Thursday, March 2, 2023, by 23:59**.

Please submit your completed tutorial on **CSCI 4181/6802 Brightspace: Assessments → Assignment 2 - Genome Assembly**. Although the assignment file can be in any format, the most compatible format is docx or PDF. Please try and name your file following this naming scheme: BannerID_LastName_Assignment1.{pdf,docx}.

Questions that you need to answer are numbered and marked in **boldface**. The point value of each question is indicated next to the question. Be sure to read the questions thoroughly and address each part.

The goal of this tutorial is to test your theoretical knowledge of assembly and give you practical experience with genome and metagenome assemblers, quality control parameters and the visualization program Bandage.

This document will help you prepare to run the software pipeline once the tutorial assignment is released.

Part Zero: Running the Programs

The Linux Environment

The server we will use, timberlea.cs.dal.ca, is a Linux server that is accessible to students in Computer Science. The number of Linux commands you will need to run in this assignment is a few and will be outlined at the relevant places in the document. A couple of key ones are:

mkdir - Make a directory

cd - Change to a directory

ls -1 - List directory contents (that's the letter 'l', not the number '1')

ls -lrt - As above, but list in order of time ('t'), reversed ('r'), so the most-recently updated file is at the bottom of the listing. This can be fun if you want to obsessively track the progress of a run.

Very very very useful tip: If you start typing the name of a file or directory, you can autocomplete by hitting the tab key. If there is some ambiguity, you may need to type a couple more letters. This can save a LOT of time. Also, you can cycle through previous commands using the up arrow.

You will need to connect to **timberlea** using the secure shell 'ssh' protocol. You can do this directly from the command line on a Linux or Mac OS system. On Windows I use the 'putty' (https://www.putty.org/) package; if you're unfamiliar with this let me know, and I can help you out.

We cannot easily look at graphical output files (such as those generated using Bandage or Quast) on Timberlea so there are two options. One is to set up an X11 connection: if you know how to do this, great! The other is to use the 'scp' command, either from the command line (again, Linux or Mac OS) or using 'WinSCP'.

If I want to copy the file 'blah.txt' from my home directory on timberlea, I would type from my own computer:

scp finlaym@timberlea.cs.dal.ca:blah.txt .

That dot means 'copy to my current location' on the current computer.

If you're on a Windows machine, I recommend using WinSCP (https://winscp.net/eng/download.php). Getting set up should be straightforward, and it's a graphical interface - again, let us know if you have trouble.

Connecting to the Server and Setting Up Your Environment

We will be using the Computer Science research server 'timberlea.cs.dal.ca' to carry out our analyses. To access this server, you need to have a CS ID. If you don't have the login credentials, please let us know, and we will ensure you get set up.

You can connect to timberlea using the 'ssh' command from a Linux or Mac OS prompt; on Windows, I use the 'putty' software. To connect, use the following command:

```
ssh <your CS ID>@timberlea.cs.dal.ca
```

All software packages you will use have been installed on a conda environment (Bioconda: see https://bioconda.github.io/). The first time you login you will need to create this environment and log out:

- Login to the timberlea server and run `` `conda init bash` ``.
- Logout by running `` `exit` ``
- Login again and now activate the environment by running `` `conda activate bioconda` ``.
- Create a directory for this assignment: `mkdir <assignment_directory_name>`

After you have done this, every time you login to the timberlea server you need to activate the Bioconda environment by running the following **two** commands:

```
source ~/.bashrc
```

```
conda activate bioconda
```

And move to your working directory: `cd <assignment_directory_name>`

MAKE SURE YOU RUN COMMANDS IN THE BIOCONDA ENV - there are broken versions of certain tools on timberlea.

Note: If you use a different shell than bash you will need to modify these commands for that specific shell… if this doesn't make sense to you don't worry about it as it isn't relevant to you!

Testing that everything works

In the next parts of the assignment, you will practice using some assembly-related tools. The list of tools is quite long, but by the end, you will have performed and evaluated assemblies, and tried to pull out interesting genomes. The list of tools is:

- **Assembly:** SPAdes, Shovill, and Unicycler
- **Quality checking**: Quast
- **Plasmid prediction:** Mob-suite
- **Mapping of reads to contigs:** samtools, Bowtie2
- **Recovery of genomes from metagenomes:** MetaBat2
- **Visualization of results:** Bandage

All required files (i.e. read files) for this assignment are available in /data/BioInformatics/a2_files. You don't need to copy these files from where they are; you can instead refer to them using their full address. You can see the full content of the folder with the command (copy and paste the below as is):

```
ls /data/BioInformatics/a2_files
```

You'll see an example of this in the command below. Once you generate results, they will be in your own folder, and you will not have to worry about this.

The first package you will run as part of the assignment is the `shovill` package. `shovill` is a convenient tool that automatically does quality control processing of raw read data (removing low-quality nucleotides).

Please test it out to ensure that everything is working:

```
shovill -h
```

Longer runs (for Part 3)

Since some commands you run for this assignment might take a few hours to complete, we suggest you run your commands in one of two ways: either using the 'nohup' command or in `screen` mode. Either of these will allow your process to keep running even if you terminate your connection (or it is terminated for you).

Option 1: /usr/bin/nohup … &

If you preface a command with '/usr/bin/nohup/` and terminate it with '&', then it will run in 'no hangup' mode. It will also run as a background process (that's the '&') so you can keep using your terminal for other tasks. A bonus of this approach is that program output is saved in 'nohup.out'.

A simple example would be this:

/usr/bin/nohup ls &

This would run the command 'ls' which lists directory contents in the background. Since this command generally takes only a fraction of a second to run, 'nohup' is not so crucial, but you get the idea.

Option 2: screen

It is more convenient because your process won't stop running if you exit the server. The details on how to use `screen' can be found at https://www.geeksforgeeks.org/screen-command-in-linux-with-examples/. Some examples:

```
//Start a new screen session:
screen

//List all available screen sessions:
screen -ls

//Detach a screen session without stopping it:
Crtl-a + d

//Reattach to a detached screen session:
screen -r
```

Option 3: tmux

This is basically a more advanced version of screen (that I personally use!). Here is a cheatsheet https://tmuxcheatsheet.com/

Part I: Some questions about de Bruijn graphs (5 points total)

Q1-1- What is an Eulerian walk? How can you quickly determine if a directed connected graph contains a valid Eulerian walk? (2 points)

Q1-2- Here is a set of short reads we would like to assemble:

ACCGT
CGTCA
GTCAG
CAGAA
AGAAT
CCGTG
GTGAG
GAGAA

a) Construct a de Bruijn graph for the above 5-mers (1 point)
b) Using this graph assemble and report the two possible sequences (1 point)

Q1-3- What is the impact of k-mer size on de Bruijn graphs? What is the impact of having a k-mer size that is too big or too small? (1 point)

Part II: Assembling one genome (/8)

In this section, you will assemble and analyse a genome from *Bacillus anthracis* (https://www.ncbi.nlm.nih.gov/genome/181?genome_assembly_id=299887), which is a causal agent of the disease anthrax. *B. cereus* is another member of the genus *Bacillus* and can cause food poisoning but is far less deadly than its very close relative *B. anthracis.* If these are so similar though, how can one be so deadly and not the other? The answer is they have different plasmids that contain different genes (including some of the key toxins). Let's see if we use genomics to assemble and compare these plasmids.

To ensure analyses run quickly, we have supplied you with some realistic-looking reads from *B. anthracis* (simulated using a tool called ART).

In this assignment, we are comparing the assemblies generated by two different programs Shovill (https://github.com/tseemann/shovill) and Unicycler (https://github.com/rrwick/Unicycler). Shovill is a full pipeline that performs read-trimming, error correction, assembly, and post-processing, whereas Unicycler just performs assembly and post-processing. They both use the same underlying de Bruijn graph assembler (SPAdes) but with different parameters and post-processing steps.

We will compare the assemblies visually (using a tool called Bandage - https://rrwick.github.io/Bandage), assess quality statistics using a tool called QUAST (https://github.com/ablab/quast), and analyse the detected plasmids using a specialised homology search tool called MOB-suite (https://github.com/phac-nml/mob-suite).

First, assemble the reads using shovill. The reads are available in the `/data/BioInformatics/a2_files` directory. For example, if you would like to generate the results in a directory called `shovill_assembly`, type the following command:

```
shovill --R1 /data/BioInformatics/a2_files/bacillus_anthracis_reads1.fq --R2
/data/BioInformatics/a2_files/bacillus_anthracis_reads2.fq  --outdir
shovill_assembly --trim --keepfiles
```

This will probably take a few minutes to run. Once the run is complete, your `shovill_assembly` (or whatever name you used) directory will contain a bunch of files and a subdirectory called 'spades'. We will use only a couple of files from these results, but the directory contains a bunch of information about the constructed graphs for different values of 'k' (by default, 31, 51, 71, 91, and 111), the error-corrected reads, and other fun information too.

Then assemble the error-corrected reads from shovill using Unicycler. For example,

```
unicycler -1 shovill_assembly/R1.cor.fq.gz -2 shovill_assembly/R2.cor.fq.gz
-o unicycler_assembly
```

Q2-1- Notice we are using the error corrected and trimmed reads from Shovill with Unicycler. Why specifically is error correction important in assembly? (1 point)

Then run Quast on both the Shovill and Unicycler assemblies. For Shovill, you will want to use the contigs.fa assembly file and for Unicycler you will want to use assembly.fasta. For example,

```
quast shovill_assembly/contigs.fa -o shovill_quast

quast unicycler_assembly/assembly.fasta -o shovill_unicycler
```

Q2-2- Compare the results from the Quast assessment of the Shovill and Unicycler assemblies. This will be in the report.pdf file in the quast folders (this can be downloaded and opened locally on your computer using `scp` as explained above).

a) Report the N50s, total length, and the number of contigs generated by Shovill and Unicycler (1 point)
b) What does the N50 mean and why is it sometimes problematic? (1 point)
c) Which assembly is better than the other and why? (1 point)

Run MOB-suite on the two assemblies using the assembly.fasta (Unicycler) and contigs.fa (Shovill) files and the scripts below. First, you need to make a directory to save the databases, and then you can run MOB-suite directing it to that folder. This will take ~30-60 minutes to download the databases.

```
mkdir mobsuite

mob_recon -t --infile shovill_assembly/contigs.fa -t --outdir
shovill_mobsuite -n 4 -d mobsuite

mob_recon -t --infile unicycler_assembly/assembly.fasta -t --outdir
unicycler_mobsuite -n 4 -d mobsuite
```

Q2-3- How many plasmids are detected in each of the assemblies? Look at the documentation for Illumina-only assembly with Unicycler (https://github.com/rrwick/Unicycler#method-illumina-only-assembly) ? (1 point)

Q2-4- Visualize (and paste here) both the Unicycler and Shovill assemblies using Bandage. You'll need to install Bandage locally on your laptop (see http://rrwick.github.io/Bandage/ for instructions) download the .gfa files, open them, and click on "Draw Graph" (1 point)

Q2-5-

a) Why does the assembly not create a graph that is just three contigs that represent the complete chromosome (1) and plasmids (2)? (1 point)
b) What is an option for trying to resolve the higher-order structure between contigs? (1 point)

That's the end of Part 2. You have identified plasmids in our friend the anthrax bacterium, but one unanswered question (unanswered because you've done plenty of work on this already!) is whether you have found the "smoking gun" plasmid(s) that are responsible for the bacterium's more extreme virulence. There are a few things you could do from here to establish guilt, including:

- Pulling the reference plasmid accessions from the MOB-suite output (for example, NC_018501 from 'mobtyper_results.txt') and looking them up at NCBI. Doing this may give you a surprising and possibly unhelpful result
- Using a tool such as Bakta (https://pubmed.ncbi.nlm.nih.gov/34739369/) to annotate protein functions, and see if there are any obvious bad actors
- BLAST some of the plasmid-associated contigs against the NCBI RefSeq database and see if any of the matches are suggestive (spoiler: yes they are!)

Part III: Assembling several genomes at once (/7)

The human microbiome is a hot topic because of its role and potential role in infectious disease (obviously), and (less obviously) its potential role in chronic conditions including inflammatory bowel disease and (possibly reaching too far) autism. The microbiome of the human gut is heavily studied because of its central importance to human health. However, the human microbiome is not a great starting point for metagenomic analysis, as a typical sample can contain hundreds of species and subspecies.

So we have simplified things a bit. For this part of the assignment, you will work with a very simple "mock" community we constructed from the genomes of four bacteria:

- *Blautia producta* (https://www.ncbi.nlm.nih.gov/genome/14364?genome_assembly_id=970886),
- *Escherichia coli* O157:H7 strain Sakai (https://www.ncbi.nlm.nih.gov/genome/167?genome_assembly_id=409151),
- *C. difficile* CD 196 (https://www.ncbi.nlm.nih.gov/genome/535?genome_assembly_id=167605),
- *C. difficile* CD21 (https://www.ncbi.nlm.nih.gov/genome/535?genome_assembly_id=257530)

These are all pathogens, with *C. difficile* the nastiest among them. You do not want this microbiome.

As an aside, *Clostridium difficile* went through a bit of a renaming a few years ago to *Peptoclostridium difficile*. But enough people were aghast at the idea of renaming "C.diff" to "P.diff" for semi-obscure taxonomic reasons that a compromise was found: another renaming to *Clostridioides difficile*. But if you see a reference to *Peptoclostridium* in the results below, this is why.

To make things even more interesting, we assigned different levels of abundance to each bacterium in the mix: 20% *E. coli*, 50% CD196, 10% CD21, and 20% *Blautia producta*. Similar to Part 2, the ART simulator (art_illumina) has been used to simulate paired-end reads with read length equal to 150, with an insert size of 500 and 20-fold coverage. The reads have been quality filtered so you can start directly with the metagenomic assembly.

The paired-end reads are stored in the compressed FastQ files 'metagenome_reads1.fq.gz' and 'metagenome_reads2.fq.gz' available at:

`/home/share/BioInformatics/a2_files directory`.

We will use MetaSPAdes (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5411777/) for this assignment.

In the command line, run the following command to see the first 12 lines of the first read file. Note that because it is a compressed file, we use ‘zcat’ instead of ‘cat’:

```
zcat /data/BioInformatics/a2_files/metagenome_reads1.fq.gz | head -12
```

You will see that each read is indeed 150 nt in length, and you will also see the quality scores associated with base in each read.

Run the following command to find out the number of lines available in each read file:

```
zcat /data/BioInformatics/a2_files/metagenome_reads1.fq.gz | wc -l
```

Q3-1- How many reads are present in each file? Remember that not every line in the fastq file contains a read! (1 point)

Let’s move on to the assembly. Be aware that this assembly process can take up to a couple of hours.

Run the following command to perform the assembly:

```
spades.py -1 /data/BioInformatics/a2_files/metagenome_reads1.fq.gz -2
/data/BioInformatics/a2_files/metagenome_reads2.fq.gz --meta -o
metaspades_output
```

[this is where you might want to consider using ‘/usr/bin/nohup’ or ‘screen’]

Files will start to appear almost immediately in the ‘metaspades_output’ directory. Once the run has been completed, you will find the assembly results including the contigs (contigs.fasta and contigs.paths) and the assembly graph (assembly_graph_after_simplification.gfa). Many of the files will look similar to those you generated in Part 2.

Run the following command to see the first 10 lines of contigs.fasta:

```
cat metaspades_output/contigs.fasta | head -10
```

Q3-2- What is the length of the largest contig? Hint: Contigs in contigs.fasta are sorted in descending order of their length. Similarly, find the length of the longest scaffold. Comment on the difference in the length of the longest contig and the longest scaffold produced here. (1 point)

Run the following command to see the list of all links (edges) in the assembly graph.

```
cat metaspades_output/assembly_graph_after_simplification.gfa | grep ^L
```

Q3-3- Using the file specification information (https://github.com/GFA-spec/GFA-spec/blob/master/GFA1.md) what does '+' and '-' signs mean here? How much is the overlap between the two nodes (segments) in each link? (1 point)

To evaluate the performance of our assembler, let's run MetaQuast (https://github.com/ablab/quast):

```
metaquast.py metaspades_output/contigs.fasta -o quast_report
```

Once MetaQuast has finished, you can go into the *quast_report* folder and view the evaluation results. You can view the quast report by opening the file report.html in your web browser (you may need to copy the file to your system first).

Q3-4- Which reference genomes did MetaQuast find after aligning the contigs? Does the ratio of contigs assigned to each of these references make sense given the respective genome sizes? (1 point)

The next step is binning. There are many software tools available for binning metagenome assemblies, one common example is MetaBat2 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6662567/).

Q3-5-

a) What 2 pieces of data do we need about the metagenome assembly to doing MAG binning with a tool like metabat? (1 point)
b) Explain why you may end up with more bins than there were genomes in the metagenome? (1 point)
c) Explain why you may end up with fewer bins than there were genomes in the metagenome? (1 point)