CSCI 4181 / CSCI 6802 Assignment 2

Genome Assembly (February 26, 2022)

This is the second of four assignments in the course that will give you practical experience with biological data. The due date for this assignment is **Monday, March 7, 2022**, by midnight. . Please submit your completed tutorial on CSCI 4181/6802 Brightspace: Assessments → Assignment 2. Although the assignment file can be in any format, the most compatible format is .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.

Contents

Part Zero: Running the Programs 2
The Linux Environment 2
Connecting to the Server and Setting Up Your Environment 3
Testing that everything works 3
Longer runs (for Part 3) 5
Part I: Some questions about de Bruijn graphs (5 points total) 6
Part II: Assembling one genome 7
Part III: Assembling several genomes at once 10

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 are 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. Finally, you can search previous commands using Ctrl+R then typing a few characters.

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 beiko@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; the Help Desk folks have created accounts for everyone so you should be able to access your account without difficulty.

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>`

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 pulled 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 directory. You don't need to copy these files from where they are; you can instead refer to them using their full location, for example

```
/data/BioInformatics/a2_files/bacillus_anthracis_reads1.fq
```

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. Please test it out to ensure that everything is working:

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

The run will take a few minutes. If it complains that the output directory already exists, you can use the '--force' option to make it overwrite the existing directory.

Note: If you get a weird message during the shovill runs to the effect of being unable to create new threads; these messages can be resolved by terminating other processes including Jupyter servers (if you're running any): see Jupyter server at https://timberlea.cs.dal.ca:8000/hub/home.

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 [your command] &

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 to the file '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 in https://www.geeksforgeeks.org/screen-command-in-linux-with-examples/. But here are the most common ones:

```
//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
```

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

Q1-1- What is the difference between a Eulerian Cycle and a Hamiltonian cycle? Why is one easier than the other to determine computationally and why? (2 points)

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

ACCGT
GTCAG
CAGAA
AGAAT
GTGAG
GAGAA

a) Construct a de Bruijn graph and compact de Brujn graph for the following reads, with 3-mers as the edges (1 point)
b) Two strains of short genomes are available in this data set, please assemble them. (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

In this section, you will assemble and interrogate the genome of *Bacillus anthracis* str. 'Ames Ancestor' (https://www.ncbi.nlm.nih.gov/genome/181?genome_assembly_id=299887), which is a causal agent of the disease anthrax. To oversimplify a bit, *B. anthracis* is a close relative of another member of genus *Bacillus*, *Bacillus cereus*, which has a very similar genome but is a lot less mean. Most of the time, that is: *B. cereus* does cause food poisoning and the occasional infection in immunocompromised individuals. The genomes are so similar that the usual marker genes we use to differentiate bacteria do not differ. So, how can one be so deadly and the other be (mostly) not deadly? The answer has come up before in class: *B. anthracis* is readily distinguished from *B. cereus* through the presence of toxins, including anthrax toxin, and a capsule; these and other nasty genes are present on a plasmid. Analysis of the chromosome may suggest that *B. cereus* and *B. anthracis* are members of the same species. So the presence of particular plasmids is a key indicator of the species of *Bacillus* you are working with. Let's see if we can find them.

We have supplied you with synthetic reads from *B. anthracis* that were generated using ART (https://www.niehs.nih.gov/research/resources/software/biostatistics/art/index.cfm) and the quality of those read were assessed using FastQC (https://www.bioinformatics.babraham.ac.uk/projects/fastqc/). So, you start off with quality checked reads to make your assemblies. 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), both of which use SPAdes. We will assess the quality of the two assemblies that we generate using QUAST (https://github.com/ablab/quast), assess how many plasmids are present in the genomes using MOB-suite (https://github.com/phac-nml/mob-suite) and visualize the assemblies using Bandage.

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 'AssemblyShovill', type the following command:

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

This will probably take a few minutes to run. Once the run is complete, your 'AssemblyShovill' (or whatever) 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 AssemblyShovill/R1.cor.fq.gz -2 AssemblyShovill/R2.cor.fq.gz -o AssemblyUNI
```

Q2-1- What would be the point of assembling the error corrected and trimmed reads from Shovill with Unicycler, rather than directly in Unicycler? Hint: look at the steps of both Shovill and Unicycler and contrast the preprocessing steps. (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 AssemblyShovill/contigs.fa -o quastshovill
```

```
quast AssemblyUNI/assembly.fasta -o quastunicycler
```

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.

a) Report the N50s, total length, and 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 in then you can run MOB-suite directing it to that folder.

```
mkdir mobsuite
```

```
mob_recon -t --infile AssemblyUNI/assembly.fasta -t --outdir mobsuiteUNI -n 4 -d mobsuite
```

```
mob_recon -t --infile AssemblyShovill/contigs.fa -t --outdir mobsuiteShovill -n 4 -d mobsuite
```

Q2-3- How many plasmids are detected in each of the assemblies? Why might they differ? (1pt)

Q2-4- Visualize (and paste here) both the Unicycler and Shovill assemblies using Bandage. You'll need the load the .gfa files 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 chromosome (1) and plasmids (2)? (1 point)

b) What is an option for resolving (also known as “closing”) the genome? (1 points)

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 bad behaviour. 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 Prokka (https://academic.oup.com/bioinformatics/article/30/14/2068/2390517) 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

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 cast of characters to invade your gut.

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 *Clostridium* or *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 /data/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 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-** What do '+' 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? (1 point)

The next step is binning. There are many software tools available for binning metagenome assemblies. For this assignment, we will be using MetaBat2 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6662567/). The way in which MetaBat bins contigs together is summarized in Figure 1.

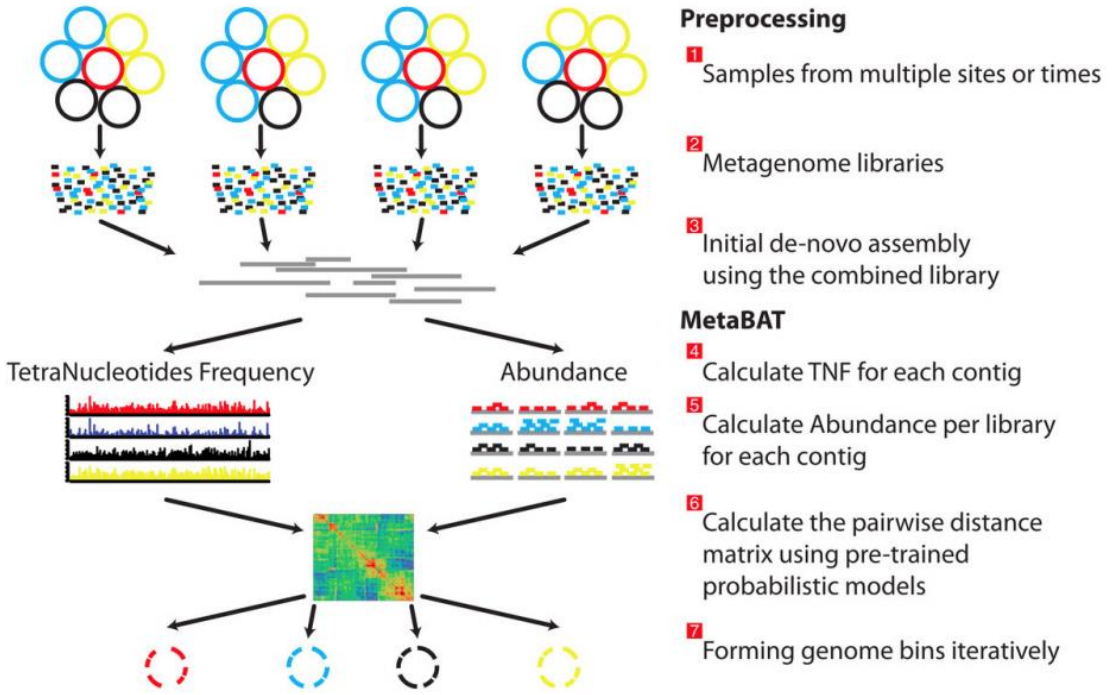

Figure 1 - MetaBat workflow (Kang et al., PeerJ 2015)

Before running MetaBat2, we need to calculate some coverage statistics by mapping reads to the contigs. Contigs are great and all, but we need to back to the reads to determine the relative proportions of different parts of the assembly - an essential part of recovering genomes with different abundance levels! We need to execute several commands to do this; we won't dwell on the details as they're more of a workup to the interesting part of the analysis.

To do this, we use bowtie2 (http://bowtie-bio.sourceforge.net/bowtie2/index.shtml) and samtools (http://www.htslib.org/). Again, this can take some time.

Index the contig file that was produced by metaSPAdes:

```
bowtie2-build metaspades_output/contigs.fasta metaspades_output/contigs.fasta
```

Map the original reads to the contigs and reformat the file with samtools and sort it

```
Note: One command from bowtie2 → assembly.bam

bowtie2 -p 4 -x metaspades_output/contigs.fasta -1
/data/BioInformatics/a2_files/metagenome_reads1.fq.gz -2
/data/BioInformatics/a2_files/metagenome_reads2.fq.gz | samtools view -bS >
assembly.bam

samtools sort -@ 4 -O BAM assembly.bam > assembly_sorted.bam
```

Run the following command to produce a tsv file summarizing the output depth for use with MetaBat2:

```
jgi_summarize_bam_contig_depths --outputDepth assembly.depth.tsv
assembly_sorted.bam
```

Create a directory to store binning results:

```
mkdir metabat_bins
```

Now, we can run MetaBat2:

```
metabat2 -i metaspades_output/contigs.fasta -a assembly.depth.tsv -o
metabat_bins/bins --unbinned -t 4
```

Q3-5-

a) How many bins did metaBat2 produce? Comment on that if the number of bins doesn't match the actual number of organisms we originally had in our sample (i.e., 4 organisms) (1 point)
b) If the number of bins is not equal to the number of organisms in the mix (four), what do you think is the source of the discrepancy? Is this consistent with the results from MetaQuast? (2 points)