Genome Assembly and Annotation

De Novo Genome Assembly

Genome assembly is a necessary task that has many confounding factors. The basic steps outlined below will give one possible assembly result. However, it is possible that using a different algorithm may yield different results. Therefore, it is incumbent upon the investigator to determine if further assembly attempts are required using different parameters, more/less data, or different algorithms entirely.

Step 1: Retrieve fastq reads

Genomes sequenced by previous students have been collected in this table: 02_Isolates_for_Assembly.xlsx (https://docs.google.com/spreadsheets/d/1DCJxfr_FYVzkxv46yWPnlcD8K90kyBpV/edit?usp=sharing&ouid=105012416008914218444&rtpof=true&sd=true). Choose a sample that interests you and make a new directory to work in. (Replace `my_group` with your group name and `my_example_assembly` with a reasonable folder name.)

In [ ]:
```
mkdir /my_group/my_example_assembly
```

In [ ]:
```
cd /my_group/my_example_assembly
```

Locate and copy a set of paired genome reads (both R1 and R2) into a local directory.

In [ ]:
```
cp /datahaus/genomes_for_assembly/FILE_PREFIX* .
```

How many reads are in your fastq files? This command will decompress one of the fastq files and read the number of lines.

In [ ]:
```
zcat Sample_R1.fastq.gz | wc -l
```

How does the number of lines in the file relate to the number of sequences?

Step 2: Check sequence quality

Tool of choice: FastQC

Make an output directory for your results

In [ ]:
```
mkdir fastqc_out
```

Run FastQC on both read files. `-o fastqc_out` tells the program to write results to the directory you just made.

In [ ]:
```
fastqc Sample_R1.fastq.gz Sample_R2.fastq.gz -o fastqc_out
```

You can use `multiqc` to aggregate yout FastQC outputs.

In [ ]:
```
multiqc fastqc_out
```

Check out the report (.html file). You can open it on the server with firefox (`firefox filename.html`) or download them to your computer and use your favorite browser (recommended).

Questions to ask:

- How many reads do you have? How long are they?
- Are they of good quality? Is the quality poor at particular positions?
- Are sequences duplicated or overrepresented? Can you tell what they are?
- Do they contain adapter sequences?

Step 3: Trimming

Trimming reads will allow us to remove parts of reads with poor quality.

Tool of choice: Trimmomatic

In [ ]:
```
trimmomatic PE -baseout YOURFILE.trimmed.fastq.gz \
Sample_R1.fastq.gz Sample_R2.fastq.gz \
ILLUMINACLIP:/opt/Trimmomatic-0.39/adapters/combined.fa:2:30:10 \
SLIDINGWINDOW:4:15 LEADING:2 TRAILING:2 MINLEN:100
```

That's a mouthful. What does it do?

- PE: These are paired-end reads.
- baseout: Base name for your output files
- ILLUMINACLIP: Trims the adapters in the given file (combined.fna) from the reads. The numbers specify the number of mismatches allowed and the minimum match quality for palindromic and simple adapter matches.
- SLIDINGWINDOW: Moves along the read in chunks of 4bp and checks average quality; if it falls below 15, those bases are removed.
- LEADING and TRAILING: Removes bases from the beginning and end of the read if the quality is below 2.
- MINLEN: Removes reads shorter than 100bp after trimming is complete.

You should end up with 4 output files:

- YOURFILE_1P.fastq.gz
- YOURFILE_2P.fastq.gz
- YOURFILE_1U.fastq.gz
- YOURFILE_2U.fastq.gz

1P & 2P contain paired reads where both reads survived trimming; 1U & 2U contain reads where the other read in the pair was lost.

- What percentage of your reads made it through the trimming process?
- How many lost their paired read?

Step 4: Assemble reads into contigs with SPAdes

There are many excellent genome assembly tools available, largely using overlap-layout-consensus (OLC) or de Brujin graph approaches. Today we'll use SPAdes, a graph-based assembler.

Assembly time will be uncertain; consider using screen.

In [ ]:
```
spades.py -o spades_assembly -1 /path/to/YOURFILE_1P.fq.gz \
-2 /path/to/YOURFILE_2P.fq.gz -t 2 -m 20
```

Explanation:

spades.py

- -o output directory
- -1 file of forward reads
- -2 file of reverse reads
- -t number of threads
- -m maximum memory (SPAdes will abort if this limit is exceeded)

When complete, fasta files of contigs (`contigs.fasta`) and scaffolds (`scaffolds.fasta`) will appear in your output directory, along with other output files.

We can generate a quick summary of the assembly using QUAST:

In [ ]:
```
quast.py -o quast_output -t 2 /path/to/scaffolds.fasta
```

-t is the number of threads QUAST will use. Results will appear in the directory specified after -o . There are several formats available; try looking at `report.txt` or `report.tsv` .

Step 5: Read mapping

This approach to map reads can be used on entire or partial genomes, metagenomes, transcriptomes... any DNA or RNA region*. Most genome assembly algorithms produce just the contigs (contiguous stretches of DNA sequence) and/or scaffolds (contigs determined to be adjacent to one another, but lacking DNA sequences that physically overlap). Therefore, after assembling a (meta)genome it is common to re-map the reads to the contigs in order to understand how the reads actually constitute the contigs. Read coverage can be indicative of copy number and possibly biology (e.g., is it a plasmid? Is it a transposon?).

*Unless you're working on eukaryotic RNA and need to take splicing into account.

Now that we have acceptable reads and assembled contigs it's time to align the two together!

Tool of choice: bowtie2

First generate an index file of your contigs/genome sequence to accelerate read mapping:

In [ ]:
```
bowtie2-build scaffolds.fasta YOUR_GENOME_INDEX
```

Now map the paired reads to the genome sequence index:

In [ ]:
```
bowtie2 --local -p 4 -x YOUR_GENOME_INDEX \
-1 trimmed_R1.fastq -2 trimmed_R2.fastq \
-S BOWTIE_OUTPUT.sam --no-unal
```

Your main output will be in BOWTIE_OUTPUT.sam; bowtie2 will also print some useful statistics to your terminal regarding how many reads mapped.

Alternatively, to determine the number of reads mapped to the contigs, you can count the number of lines in the SAM file that have reads using grep:

In [ ]:
```
grep -cv "^@" BOWTIE_OUTPUT.sam
```

What percentage of your reads mapped to the reference genome?

Step 6: Prepare sorted BAM file and indices for visualization

To visualize read mapping in IGV requires 1) an indexed .fasta of your contigs, and 2) a sorted and indexed .bam file of your aligned reads. BAM files are (usually) compressed binary versions of .sam files. Note these commands are not instantaneous and may take a minute or three to run.

In [ ]:
```
samtools faidx scaffolds.fasta
```

In [ ]:
```
samtools view -b BOWTIE_OUTPUT.sam  > BOWTIE_OUTPUT.bam
```

In [ ]:
```
samtools sort -o BOWTIE_OUTPUT_sorted.bam BOWTIE_OUTPUT.bam
```

In [ ]:
```
samtools index BOWTIE_OUTPUT_sorted.bam
```

Step 7: Visualize alignments in IGV

Next we'll visualize your annotations using IGV. You can use the web app here (https://igv.org/app/) or download and install it on your laptop.

Download your scaffolds (.fasta), its index (.fai), sorted BAM file (.bam) and its index (.bai) to your laptop using Cyberduck, winSCP, or your preferred file transfer tool.

Load your reference genome (scaffolds) into IGV (Genome -> Local file...; select your .fasta)

Load your alignment as a track (Tracks -> Local file...; select your .bam)

- Is coverage even across the contigs?
- Are all parts of the contigs covered?
- Are some especially deep?

Genome Annotation

Now that you have an assembled genome, it's time to see what it can do! There are many annotation pipelines available; today we'll be using prokka, which specializes in bacterial and archaeal genome annotation. Prokka will use hidden Markov models (HMMs) to find open reading frames (ORFs), tRNAs, rRNAs, and other genomic elements such as CRISPRs.

Prokka is simple to run, but has lots of options. You can run it with no arguments to see everything it can do.

In [ ]:
```
prokka
```

Run a basic annotation on your assembled genome:

In [ ]:
```
prokka --cpus 2 --mincontiglen 200 --outdir PROKKA_OUTPUT \
--prefix YOUR_GENOME_NAME scaffolds.fasta
```

In your PROKKA_OUTPUT directory, you'll find several output files.

- The .gff and .gbk files contain the annotation info (same data, two formats)
- The .ffn file contains the gene nucleotide sequences
- The .faa file contains the protein sequences
- The .txt file contains a summary of the features prokka annotated

Next we'll visualize your annotations using IGV, much like we did with the mapped reads. Again, you can use the web app here (https://igv.org/app/) or download and install it on your laptop.

Copy your .gff file from prokka to your computer using Cyberduck or other file transfer tool. You should already have your .fasta and .fai for your scaffolds from earlier.

Load your reference genome into IGV (Genome -> Local file...; select your .fasta)

Load your annotations as a track (Tracks -> Local file...; select your .gff)