Sequence Assembly

The essential problem that sequence assembly is trying to overcome is that the average microbial genome is 2,000,000 bp, while the typical sequence read length is 150-300 bp for Illumina sequences and upto 50,000 bp for PacBio or Nanopore sequences.

With such (relatively) short sequences, how can we assemble a whole, or nearly whole, genome?

The answer is by repetitively sequencing the same thing over and over again! If we start each sequence at a random location, and we have enough sequences, eventually we can join those sequences together to form what we call **contigs**.

There are four types of sequence assembly algorithms:

1. Naive assemblers which just try and find all matching pairs of reads
2. Greedy assemblers which start with one read and keep adding reads until you can not find any more matches, and then start with the next read.
3. Overlap-layout-consensus assemblers which layout the reads looking for overlaps between them. The overlaps are usually refined by a Smith-Watermann search, and then a consensus constructed.
4. de Bruijn graph assemblers

This table describes some of the common sequence assemblers that you will run across.

Name	Type	Sequencing Tech	Citation	Documentation	Homepage
SPAdes	genomes, single-cell, metagenomes, ESTs	Illumina, Solexa, Sanger, 454, Ion Torrent, PacBio, Oxford Nanopore	Nurk *et al*, 2013	version 3.12 manual	SPAdes
Velvet	genomes	Sanger, 454, Solexa, SOLiD	Zerbino and Birney, 2008	version 1.12 manual	EBI

Name	Type	Sequencing Tech	Citation	Documentation	Homepage
Canu	genomes	PacBio/Oxford Nanopore reads	Koren *et al.* 2017	manual for all versions	Git repo
MaSuRCA	Any size, haploid/diploid genomes	Illumina and PacBio/Oxford Nanopore data, legacy 454 and Sanger data	Zimin A, *et al.* 2017	Git Repo	Git Repo
Hinge	Small microbial genomes	PacBio/Oxford Nanopore reads	Kamath *et al.* 2017	Jupyter notebook	Git repo

We use the St. Petersburg genome assembler, SPAdes and the version installed on the AWS instances is 3.12.0 for which the manual is here

For Nanopore reads we typically use the CANU assembler.

Running SPAdes

SPAdes is easy to run! The basic command is

```
spades.py
```

The program takes a couple of inputs - your `fastq` files, for example that you download from ../Databases/SRA.

If you have paired end reads, you need to add `-1` for the left pairs (the file called xxx_1.fastq) and `-2` for the right pairs (the file called xxx_2.fastq). Note that spades handles `gzip` compressed files, and you do not need to decompress them!

If you unpaired reads, you can specify that with the `-s` flag.

You also need to provide an output directory name where the results will be written using the `-o` flag.

Your final command might look something like:

```
spades.py -1 fastq/ERS011900_pass_1.fastq.gz -2 fastq/ERS011900_pass_2.fastq.gz -o assembly
```

SPAdes output files

SPAdes makes a lot of files and directories in the output, and this summarizes what those files are. Of course, more details can be found in the SPAdes manual

- **`scaffolds.fasta`** contains the scaffolds generated by SPAdes and is the **output file you want to use**.
- the directory `/corrected/` contains reads corrected by BayesHammer in compressed fastq format
- **`contigs.fasta`** contains the contigs before they are scaffolded into scaffolds. Often this is similar to the scaffolds.fasta depending on how much scaffolding information there is
- `assembly_graph.gfa` contains the assembly graph and scaffolds paths in GFA 1.0 format
- `assembly_graph.fastg` contains the assembly graph in FASTG format
- `contigs.paths` contains paths in the assembly graph corresponding to contigs.fasta. This is how the graph is resolved into contigs.
- `scaffolds.paths` contains paths in the assembly graph corresponding to scaffolds.fasta.
- `K21`, `K33`, `K55`, etc are directories containing the de Bruijn graph assemblies for different lengths of k
- `before_rr.fasta` are the assembled contigs before repeat resolution has been applied.
- `dataset.info` and `input_dataset.yaml` contain information about the sequence read files that were supplied.
- `params.txt` is a summary of all the spades parameters
- `spades.log` is the log that was printed to the screen while SPAdes was running. This contains lots of information about the assembly process.