Next Generation Bioinformatics Tools

Hesham Ali

Exercise Four

**Objective:** This exercise will review sequence assembly process and familiarize students with different parameters affecting the assembly process. It will also demonstrate how assembly is visualized.

Theoretical Background:

1. Types of assemblers
	1. overlap-layout-consensus (newbler, celera, edena) vs. De Bruijn Graph (Abyss, Velvet, Soap de novo)
	2. de novo(Abyss, Velvet, Celera) vs. reference based (AMOScmp,454gsMapper)
2. Sequencing Types
	1. Single-end (reads from one end of the fragment)
	2. Paired-end (sequenced reads from both ends of a fragment)
3. Assembly parameters
	1. Kmer size (k)
	2. minimum overlap length (m)
4. Assembly statistics
	1. Size
	2. Number of contigs (n)
	3. N50

Part1: Example Assembly Run

|  |
| --- |
| mkdir day4cd day4abyss-pe k=20 name=test se=/data/day4/test\_reads.fa |

Input file format

The input file format is FASTA or FASTQ. To open the input file and see the reads type the following:

|  |
| --- |
| gedit /data/day4/test\_reads.fa |

Output files

Several output files are produced from the assembly process. The most important one is the contig file.

1. **test-se-contigs.fa**

|  |
| --- |
| gedit test-se-contigs.fa>0 99889 2001860CTACGGCGAAGATAATGAGATCGGTAG …….0 = contig id99889 = length of contig2001860 = total kmer coverage |

1. **test-bubbles.fa and test-indel.fa**

|  |
| --- |
| These files have candidate SNPs in the reads. In particular, the indel file has bubbles with indels, but contains only the branch of the bubble that was removed from the main assembly. This can be used to find indels when mapped to final assembly. |

1. **coverage.hist**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| This file has coverage information.

|  |  |
| --- | --- |
| Times appeared | Number of kmers |
| 1 | 269106 |
| 2 | 6686 |
| 3 | 161 |
| 4 | 16 |
| … | … |

 |

Assembly statistics

|  |
| --- |
| abyss-fac test-se-contigs.fa |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| n | n:200 | n:N50 | min | N80 | N50 | N20 | max | sum |  |
| 1 | 1 | 1 | 99889 | 99889 | 99889 | 99889 | 99889 | 99889 | test-se-contigs.fa |

What do these measures mean?

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|

|  |  |
| --- | --- |
| **n**  |  number of contigs |
| **n:200**  |  number of contigs > 200bp  |
| **min**  |  minimum contig size |
| **N50**  |  length of a contig from a set of longer contigs whose combined length represent 50% of assembly. |
| **n:N50**  |  number of contigs greater than N50 length |
| **max**  |  maximum contig size |
| **sum**  |  total assembly size (if multiple contigs, all lengths are added) |

Ideally, we want to get least number of contigs, which are very long.  |

Process Overview:

In this exercise, different parameters are changed during the assembly process and the goal is to analyze the assembly output and find the best set of parameters. For one set of sequence reads, K-mer size, overlap size and number of processors can be changed. Number of processors doesn’t affect the quality of the assembly but can significantly alter the time to produce the result.

ABySS Assembler

Sequence Reads

Kmer size

Overlap size

Contigs

No. of

Proc.

## Case Study1: What effect do kmer and overlap sizes have in the process of assembly?

Let us first change kmer size parameter in Abyss assembler. This assembler has “k=” switch, that can be assigned different kmer sizes. “k=25” means kmer size is 25.

Shown below is a ‘for loop’, that iterates from k=20 to k=30.

|  |
| --- |
| for k in {20..30}; do mkdir k$k; cd k$k; abyss-pe k=$k name=test se=/data/day4/test\_reads.fa; cd ..; donek=$k changes k-mer size from 20 to 30 |

All the outputs will be stored inside directories labeled as k20, k21, k22 ….. k30.

Next, let us run abyss-fac and compare the assembly results.

|  |
| --- |
| for x in `ls k\*/ test-se-contigs.fa`; do echo $x; abyss-fac $x; doneThis outputs the assembly statistics for output from all values of “k”. Here, we notice that the best result is given when minimum k is 20.  |

|  |
| --- |
| Also, notice that when the value for k is increased, the number of contigs will increase, which means there are multiple shorter contigs. |

Next, let us change the overlap length size parameter in Abyss assembler. This assembler has “m=” switch that can be used to assign different overlap length sizes.

Shown below is a for-loop, that iterates from m=20 to m=30.

|  |
| --- |
| for m in {20..30}; do mkdir m$m; cd m$m; abyss-pe k=20 m=$m name=test se=../test\_reads.fa; cd ..; donem=$m changes overlap lengthTo compare the assembly results, abyss-fac is run for all the outputs in m20, m21, m22 … m30 directories.for x in `ls m\*/ test-se-contigs.fa`; do echo $x; abyss-fac $x; doneFor all values of m, the assembly result is same. |

## Case Study2: Does the use of more processors always yield fast results?

In this case study, we will evaluate the relation between the number of processors and the assembly process. Although the quality of assembly remains unaffected, the time needed to complete the process varies with the number of processors.

Abyss assembler has “np=” switch to assign number of processors.

Following is a command to run the Abyss assembler with 2, 4 and 8 processors and record the time needed to run the assembly process.

|  |
| --- |
| for p in {2,4,8}; do mkdir proc$p; cd proc$p; time abyss-pe k=20 np=$p name=test se=../test\_reads.fa; cd ..; done |

|  |  |
| --- | --- |
| Processors | Time to Finish |
| 2 | 0m3.322s |
| 4 | 0m2.941s |
| 8 | 0m3.965s |

|  |
| --- |
| We notice that increasing the processors from 2 to 4 reduced the time needed for assembly, however increasing it to 8 increased the running time. Hence, based on input size and parameters, the number of processor is to be chosen. |

## Case Study3: Assembly of paired-end data (approx. 10min)

In this case study, we will run the assembler for paired-end data. The paired end data can be available in two different ways, viz.

1. In a single FASTQ file containing the pair or
2. In two separate FASTQ files
	1. Multiple fastq files can be placed in quotes

|  |
| --- |
| abyss-pe k=25 name=paired\_end in=’file1.fastq file2.fastq’ |

|  |
| --- |
| abyss-pe k=25 name=paired\_end in=’/data/day4/paired\_end\_1.fastq /data/day4/paired\_end\_2.fastq’ |

Part2: Visualization of assembly

In this exercise, we will align the reads with the contigs and visualize the depth of contigs, candidates for SNPs, etc.

Software Needed (already installed):

1. Bowtie (http://bowtie-bio.sourceforge.net/index.shtml)
2. Tablet (http://bioinf.scri.ac.uk/tablet/)



Steps for visualization (all in assembly directory)

Step1: Index all the contigs using bowtie indexor

|  |
| --- |
| bowtie2-build paired\_end-3.fa paired\_end-3.index |

Step 2: Align all the reads to the indexed contigs (approx. 10 min)

|  |
| --- |
| bowtie2 -x paired\_end-3.index -1 /data/day4/paired\_end\_1.fastq -2 /data/day4/paired\_end\_2.fastq -S paired\_end-3.sam |

Step 3: Index the contig file

|  |
| --- |
| samtools faidx paired\_end-3.fa |

Step 4: Convert sam to bam file

|  |
| --- |
| samtools import paired\_end-3.fa.fai paired\_end-3.sam paired\_end-3.bam |

Step 5: Sort the bam file

|  |
| --- |
| samtools sort paired\_end-3.bam paired\_end-3.bam.srt |

Step 6: Index the sorted bam file

|  |
| --- |
| samtools index paired\_end-3.bam.srt.bam |

Step 7: Open Tablet software from Desktop. Cancel any update options.

Step 8: Load Sorted Bam file and the Contigs file in the visualization software (Tablet)

Click on Open Assembly and give path to the sorted bam file from step6 and the contigs file (paired\_end-3.fa).







Part3: Cascading of assemblers (Meta-genomics or multiple chromosomes)

Sometimes, sequence reads can come from multiple chromosomes/organisms. Can we cascade reference-based assembly with a de novo assembly?

Yes, we can first separate reads by aligning it to reference chromosomes/organisms and separate reads based on these alignment. After the reads are separated, we can perform de novo assembly.

Step 1: Creating bowtie index of reference chromosome or organism

|  |
| --- |
| bowtie2-build /data/day4/reference.fasta reference.index |

Step2: Align the reads using bowtie

Here is an example of paired end data again. file1.fastq and file2.fastq are two fastq files for paired end data. The output file is file.sam.

|  |
| --- |
| bowtie2 –x reference.index -1 /data/day4/paired\_end\_1.fastq -2 /data/day4/paired\_end\_2.fastq –S alignment.sam |

Step3: Get the mapped read ids from the alignment file

|  |
| --- |
| cat alignment.sam| grep -v '^@' |grep 'reference' |cut -f1 > filter.ids |

Step4: collect sequences from original fastq files with the ids from filtered list.

|  |
| --- |
| fastqselect -infile /clab\_bdb/course\_supplement/workshop/data/day4/paired\_end\_1.fastq -name filter.ids –outfile paired\_end\_1.filtered.fastq |
| fastqselect -infile /clab\_bdb/course\_supplement/workshop/data/day4/paired\_end\_2.fastq -name filter.ids –outfile paired\_end\_2.filtered.fastq |

This will produce paired\_end\_1.filtered.fastq and paired\_end\_2.filtered.fastq. These fastq files have filtered set of reads after mapping to reference sequence.

Step5: Fastq files from step4 can be used to do de-novo assembly as shown in previous section.

|  |
| --- |
| abyss-pe k=25 name=paired\_end in=’paired\_end\_1.filtered.fastq paired\_end\_2.filtered.fastq’ |