Genome Assembly

25 September 2024

Before you start

- In these exercises you are going to consider node-centric de Bruijn graphs of order k. This means that k-mers will be nodes and edges correspond to exact overlaps of length $k-1$.
- Files with the *.gz* suffix are compressed using *gzip*. To use them in your exercises, you can either decompress them or use the *xopen* function of the *xopen* library to handle both compressed and uncompressed files at the same time (it works similarly to the built-in *open*). After activating a conda environment, you can install it using the command

  ```
  conda install bioconda::xopen
  ```

- In the homework folder you can find the Python file *readfa.py* from which you can import the *readfq* function to iterate over the genomic sequences of a FASTA file. Example:

  ```
  from xopen import xopen
  from readfa import readfq

  with xopen(file_path) as fasta:
      for _,seq,_ in readfq(fasta):
          # process seq
  ```

de Burijn graphs (dBG)

Q 1. What is the minimal information you need to represent a de Bruijn Graph (dBG)? Propose a data structure (in Python) to store such a graph.

Q 2. Implement a function *create_dbg* that builds a dBG of order k, given the following input parameters:

- The path of a FASTA file containing genomic sequences;
- The size k of a k-mer;
- An abundance threshold t, that is, the minimum number of times a k-mer has to occur in the input sequences to be stored within the graph.

Remember that you need to properly handle k-mers belonging to the forward and reverse strand of a genome.

Download the file *ecoli_genome_150k.fa* from the course repository. It is a 150-kbp long fragment of the *Escherichia coli* genome sequence.

Q 3. Run *create_dbg* on this file, using $k = 31$ and $t = 1$. How many k-mers have been processed? And how many have been stored in the graph?

Building a dBG from sequencing reads

In this section you are going to build a dBG of different sequencing experiments. More precisely, a $20\times$ coverage of 100-bp reads has been simulated with different characteristics from the *E. coli* genome fragment you already considered in the previous question. The files are available as (gzipped) FASTA files which you can find in the `reads` sub-directory.

a. *ecoli_sample_perfect_reads_forward.fasta.gz*: perfect reads, forward strand only

b. *ecoli_sample_perfect_reads.fasta.gz*: perfect reads, both forward/reverse strands

c. *ecoli_sample_reads_01.fasta.gz*: 0.1% error, both strands

d. *ecoli_sample_reads.fasta.gz*: 1% error, only both strands

Q 4. For each dataset answer the following questions. How many k-mers are there? How many have been stored in the graph? For datasets *c* and *d*, retain only k-mers occurring at least t times. Try different values of t, what do you notice?

Q 5. *(optional)* Plot a k-mer frequency histogram of the *c* and *d* datasets. Can you propose an optimal value of t?

Note: you can use, for example, matplotlib *in Python to generate such a plot.*

Unitigs

Remember that a *unitig* is defined as the genomic sequence that can be spelled by a maximal non-branching path in a sequence graph (in our case a de Bruijn graph).

Q 6. Create a method/function *unitig_from* such that:

- the input is a dBG of order k and a k-mer s.
- the output is the unitig containing s.

Q 7. Apply *unitig_from* on the previously created de Bruijn graphs using as input the 31-mer

```
CGCTCTGTGTGACAAGCCGGAAACCGCCCAG
```

For each read dataset and for each $t \in \{1, 2, 3, 4\}$:

- What is the size of the unitig you get in output?
- What do you notice? Why?

Q 8. *(optional)* Create a method/function *create_unitigs* that, given a dBG of order k, uses *unitig_from* to output all the unitigs of the graph. Answer the previous question by considering the total size of all the unitigs of the dBG.