Report on project in MVE405

Aug 21, 2013

1 Abstract

This report documents the outcome of a project in the Chalmers course MVE405 - Individual project in mathematics and mathematical statistics.

1.1 Background

Broad-spectrum fluoroquinolone antibiotics are heavily used in modern healthcare for prevention and treatment of bacterial infections. Fluoroquinolone antibiotic resistance (qnr) genes give bacteria a way to resist these antibiotics and often spread between species. **Boulund** published a paper with an accompanying tool for identifying such genes using Hidden Markov Models (HMM) and gene clustering techniques. While giving satisfactory results for that particular case, it was difficult both to use the pipeline for more complex models, other types of data and huge datasets. This project aims to generalize, modularize and optimize the pipeline to both expand its use cases to identify, for example, beta-lactamese enzymes, while enabling it to work on huge sets of short, fragmented reads on the scale of tens of terabases (TB).

1.2 Results

While retaining the original method and statistical power of the original pipeline, the new toolset - called SofT - enables a fast and easy way to combine substeps in hierarchical and parallell order to use several Hidden Markov Models. This paper will also show that new functionality enables analysis of either nucleotide or amino acid representation of sequences in each step, giving both flexibility and security when fragments are later assembled as nucleotides after an initial filtering an analysis of their possible translations as proteins. Finally, datasets of arbitrary size can be used as input and we will

explore the feasability with regards to memory, storage and computational time. The main part of the project has been revolving around the technical design and implementation of the new toolchain, and this will be reflected in the report.

1.3 Conclusion

2 Background

3 Results

3.1 Overview

SofT is developed for Python 2.6, but should run fine under 2.7 as well. For details on installation and dependencies, refer to the online documentation.

Being a pipeline, there are several steps in the process. Each such step is called a *sieve*. Each sieve will take an input file with sequences in FASTA or FASTQ format to run, optionally with two reference databases - one for nucleotide sequences and one for protein translations - generated by any previous sieves in the current execution. After processing, it in its turn generates a new output file of similar format with reference databases. Generally, an output FASTA file is used as input data in the next sieve while the databases are used as backreference to quickly retrieve either representation of the sequence from an id. Databases also store metadata generated during a run, such as hmmsearch domain scores.

Input is supplied as nucleotide sequences in FASTA or FASTQ format, in clear-text or gzipped. The treatment of the input file is determined by the filename ending.

3.2 qnr sieve example

Below, we will briefly use the qnr gene identification case used in Boulund's original pipeline as an example to illustrate the flow and structure of SofT and the presupplied sieves. See figure 2 for an illustration.

3.2.1 readfasta

The first step in any pipeline will usually be a preprocessing step performed by the *readfasta* sieve. It simply translates the nucleotide sequences in the input files in all six reading frames and outputs the result to database files and an output text file.

3.2.2 hmmsearch

The hmmsearch sieve takes a FASTA file with protein sequences and performs a HMMER hmmsearch against a predefined model file. Only sequences whose domain score passes a predetermined classification function will be outputted. The default classification function is the one derived from the results in **Boulund**

3.2.3 sga

The output sequences from hmmsearch are considered as potential fragments from qnr genes. As input is usually from metagenomic samples sequenced from so-called Next-Generation Sequencing technologies such as Illumina, an attempt to assemble them into full-length genes is made. In this case, the String Graph Assembler (SGA) package is used. Since SGA doesn't keep a reference to the fragments that form a contig, this information is lost. Furthermore, the non-zero error tolerance rate (default 0.02) makes it non-trivial to derive them from the databases. As a consequence, the current implementation of SGA does not put anything in its output databases but only generates output files. This could, in the future, be resolved by using an aligner such as MAFFT.

3.3 beta-lactamase example

The beta-lactamase sample functions as the qnr one, with a notable difference. The different classes of qnr genes can be classified using a single hmmer model. The structure of beta-lactamase genes makes this more difficult for them. To get satisfactory results, a hierarchical model is used: An overall model used to classify as potential qnr or not, with a subsequent model for each of the serine-based subclasses. This suggests a hierarchical model, illustrated in figure **??**.

3.4 Selecting values for String Graph Assembler

SGA (String Graph Assembler) is a complex pipeline in itself, with several parameters to tweak that will heavily influence the result of an assembly. It is thus important to pick the variables carefully in order to get sane results. In order to select the default values for the SGA sieve, an evaluation set was created by taking sven known qnr genes and randomly slicing and mutating them, creating a file with 100 basepairs long sequences. The script that was used to do this is included in scripts/fb_fragments_from_fasta.py, with a random mutation rate of 1%.
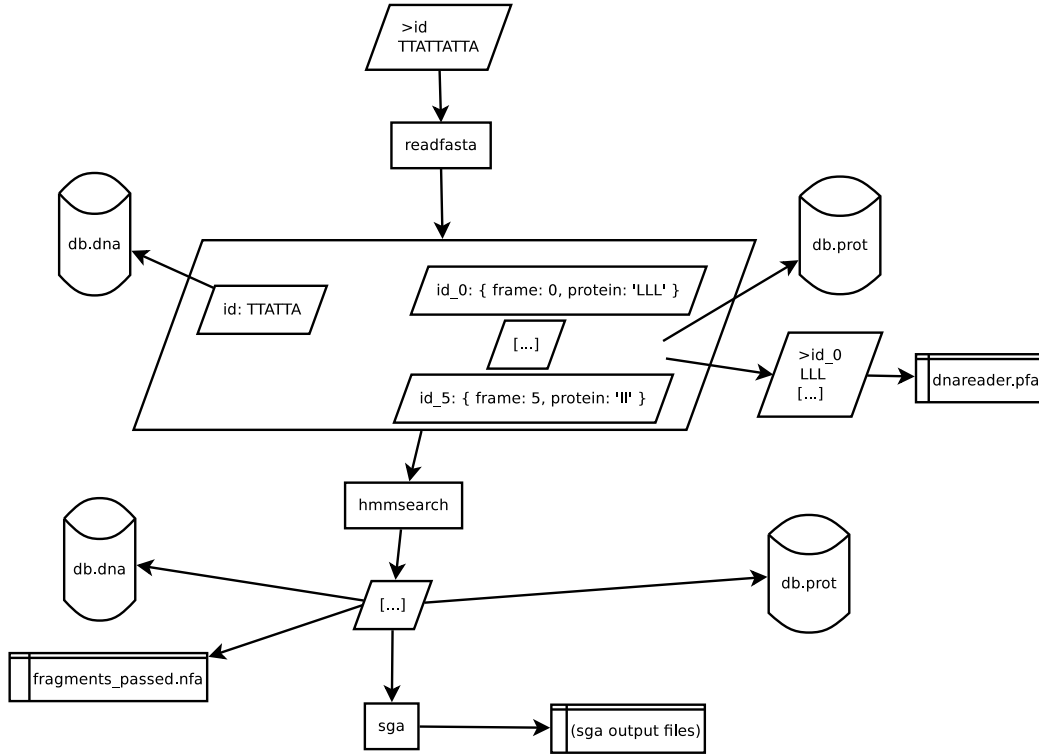
Figure 1: Flow for qnr sieve sample. The structure of the output data of all sieves are the same. For the sake of brevity, only the first is shown in detail.

The evaluated parameters and their values tested:

- error_rate: [0.01, 0.02, 0.03],

- min_assembly_overlap: [0, 5, 10, 15, 20, 25, 30],

- min_merge_overlap: [5, 10, 15, 20, 25, 30],

- resolve_small: [0, 5, 10, 500]

Running the multirunner sieve with these paramaters initiated an SGA sieve run with each of all the possible combinations of these values. Output sets were then sorted based on contig count, average contig length and length of longest contig. They were then manually evaluated. Balanced results seemed to be acieved with the following values:

- error_rate: 0.02
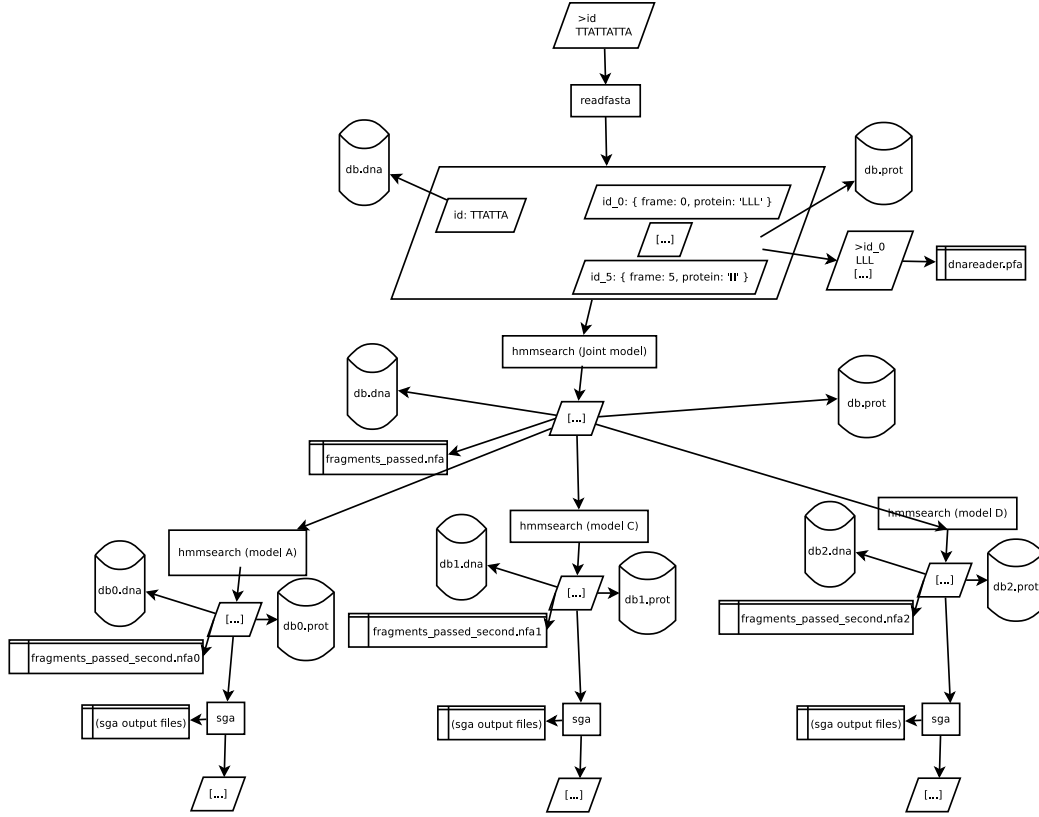
- min_assembly_overlap: 20

Figure 2: The hierarchical flow used in the betalactamase sample. Each serine subclass creates an individual fork of the sieve for the rest of the run.

- min_merge_overlap: 20

- resolve_small: 5

This gave 54 contigs with an average length of 84 and a longest contig length of 592. This seems to suggest that further scaffolding or alignment might be needed to get really meaningful results. Since the scaffolding part of SGA requires paired-end or mate pair reads and the framework is designed for a use case with more general data in mind, this option has been discarded for the moment. If this functionality is desired, the SGA sieve can easily be modified accordingly.

3.5   Performance

Execution time logs for gzip format:
      Input: 100 bp Immunina reads

37,168,092 fragments, gzipped = 3,716,809,200 bases $\sim=$ 3.7 GB (resulting pfa file: 9.7 GB)

Step 1	Translate DNA, insert into database	1:20:39
Step 2	hmmsearch (1076 sequences passed)	0:02:14
Step 3	SGA	0:00:6

$\sim$2.5 GB / h = 0.4 h / GB = 400 h / TB $\sim=$ 17 days / TB = 167 days / 10 TB.

3.6   Design choices

3.6.1   Translator

Since input data is in nucleotide format and anlasys is done for amino acids, the input sequences need to be translated before analysis. Boulund's original QNR pipeline used transeq for this purpose. Transeq is stable and has very good performance. However, the output from transeq does not contain any reference to the input sequence. Since the original DNA sequence is used for analysis in later stages and the startup overhead involved in running transeq sequence by sequence was too large to be fast enough, an alternative had to be found. Biopython was evaluated but proved to be too slow. A custom implementation in Python worked better, but even after aggressive optimizations it was still more than ten times slower than transeq and took a considerable fraction of the total running time of the pipeline. The code was ported to Cython, a statically typed superset of Python that compiles to C code with Python bindings. Adding type declarations alone gave significant improvements, and after all the heavy parts of the process had been optimized, it was fast enough so that database and file I/O and serialization were the only significant bottlenecks.

3.6.2   Serialization

Sequences and their metadata are stored in dictionaries. In Python, JSON is the fastest serialization option, and the most concise text-based one, which made the choice natural. Unfortunately, there is significant overhead involving type-lookups that slows down the process. Therefore, a string-concatenation-based approach is used for serialization while the bundled json solution is still used for deserialization. Given the low percentage of sequences that make it through in the QNR and beta-lactamase use cases, serialization is performed to a much higher extend than deserialization so the somewhat suboptimal deserialization is still acceptable.