Practical Computing: Assignment One

Emma Gheysen
emma.gheysen@student.kuleuven.be

James O'Reilly
james.oreilly@kuleuven.be

November 6, 2019

Introduction

This report was created as part of an assignment for the *Practical Computing for Bioinformatics* course at KU Leuven. Each of the FASTA files, clustal allignment files, and scripts are included in the compressed folder that contained this report, as well as a file displaying the outputs of our scripts.

1 Question One

1.1

The IDs all start with the prefix XP_, which are accession numbers for proteins. These model RefSeqs are from the NCBI database [1].

1.2

The IDs refer to proteins and their amino acid sequences. Some of the accession numbers correspond to identical proteins and some to isoforms. They all belong to the species *Cyanistes caeruleus*. An overview of the sequence IDs and the proteins are given in Table 1 below.

Sequence ID	Type of sequence
XP_023795524 XP_023795527 XP_023795526 XP_023795528	bcl-2-like protein 1 230 aa protein
XP_023782391	pantothenate kinase 2 mitochondrial isoform X5 243 aa protein
XP_023782390	pantothenate kinase 2 mitochondrial isoform X4 265 aa protein
XP_023782389	pantothenate kinase 2 mitochondrial isoform X3 372 aa protein
XP_023782388	pantothenate kinase 2 mitochondrial isoform X2 376 aa protein
XP_023782387	pantothenate kinase 2 mitochondrial isoform X1 364 aa protein
XP_023784933	iron-sulfur protein NUBPL isoform X3 265 aa protein
XP_023784929 XP_023784930	iron-sulfur protein NUBPL isoform X1 204 aa protein
XP_023778849	polyribonucleotide nucleotidyltransferase 1 mitochondrial isoform X2 728 aa protein
XP_023778848	polyribonucleotide nucleotidyltransferase 1 mitochondrial isoform X1 703 aa protein
XP_023799485	collagen type IV alpha-3-binding protein 624 aa protein
XP_023788958	PTB-containing, cubilin and LPR1-interacting protein isoform X2 203 aa protein
XP_023788972	mitochondrial fission factor isoform X3 311 aa protein
XP_023788970	mitochondrial fission factor isoform X1 336 aa protein
XP_023781610	myotubularin isoform X2 602 aa protein
XP_023781612	myotubularin isoform X4 632 aa protein
XP_023781603 XP_023781606 XP_023781609 XP_023781608 XP_023781605	mitochondrial fission factor isoform X1 336 aa protein

Table 1: List of sequence IDs and the type of sequence

1.3

For this question we had the problem that BioMart has been down for a few days. We would have used the ID conversion tool [2]. Ultimately we could not do this question as a result.

1.4

Because we were not able to retrieve the GO terms, we again used the XP_ RefSeqs to find the corresponding gene on NCBI [1]. Then HomoloGene was used to find homologs [3]. Every protein has homologs in *Homo sapiens*, but none exist in *S. cerevisiae*. The homologs are listed in Table 2.

Sequence ID	**Homolog in Homo Sapiens**
XP_023795524 XP_023795527 XP_023795526 XP_023795528	bcl-2-like protein 1 230 aa protein
XP_023782391 XP_023782390 XP_023782389 XP_023782388 XP_023782387	PANK 2 gene GeneID: 80025
XP_023784933 XP_023784929 XP_023784930	nucleotide binding protein-like 319 aa protein NP_079428.2
XP_023778849 XP_023778848	polyribonucleotide nucleotidyltransferase 1 783 aa protein NP_149100.2
XP_023799485	CERT1 gene GeneID: 10087
XP_023788958	phosphotyrosine interaction domain containing 1 248 aa protein NP_060403.3
XP_023788972 XP_023788970	MFF gene GeneID: 56947
XP_023781610 XP_023781612 XP_023781603 XP_023781606 XP_023781609	MTM1 Gene GeneID: 4534

Table 2: List of sequence IDs and their corresponding types of sequences

2 Question Two

2.1

The sequence was uploaded to NCBI nucleotide blast [4], which returned the full genome as the best match, rather than a specific gene. By clicking "graphics", the matching genome region could be found. This region corresponds with the recA gene of the *Cronobacter sakazakii* strain CS-09 chromosome. This is displayed in Figure 1.

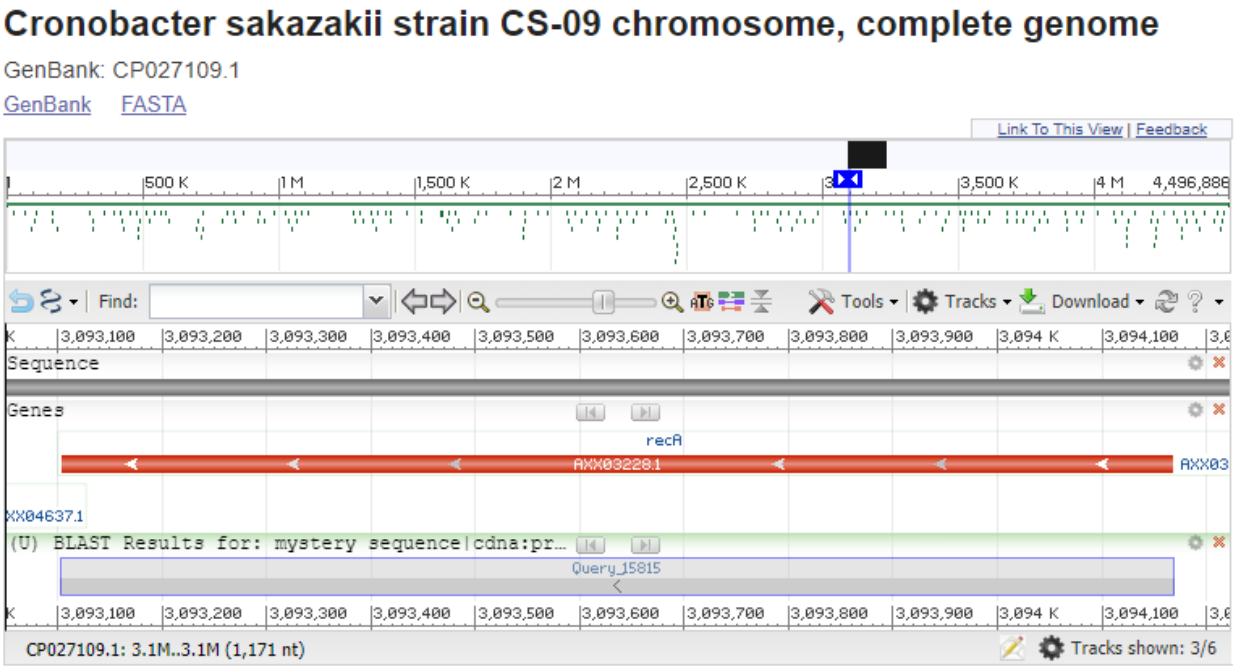

Figure 1: RecA gene of theCronobacter sakazakii

2.2

First, the fasta protein sequences for RAD51 in *H. sapiens* and *S. cerevisiae* were retrieved through NCBI [1]. Clustal Omega was used to perform the multiple sequence alignment [5]. This result was then analyzed using the MView tool on the EMBL-EBI website [6]. As can be seen in the MSA file, the region that displays the most homology is between amino acid numbers 150 up to 410. The amino acids that are identical have a non-white background. The results can be seen in Figure 2.

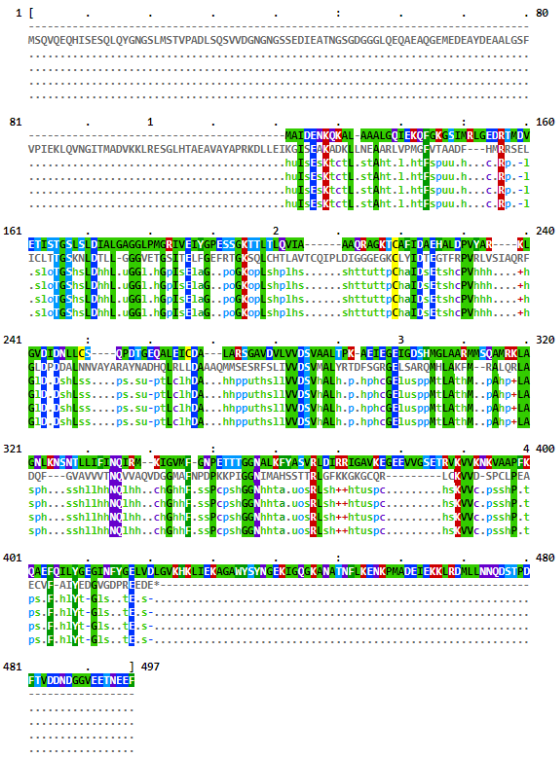

Figure 2: Multiple sequence alignment using ClustalW and viewed with MView

3 Question Three

For this question we have assigned the mutated strains text file to the `$file` variable, for readability in both this report and the accompanying script, `Question3`.sh.

3.1

Looking at the structure of the mutated strains text file, we can see that the third column contains information about chromosomes. To return the number of chromosomes in the dataset, we first need to access this column with `cut -f4 $file`. The unique elements in the column can then be returned by piping to `sort | uniq`. Visualising this output, we noticed that it included the column title "chr". The column title can be removed by first piping to `sed "1 d"` to remove the first line before returning the unique elements of the column.

It can also be seen that some of the mutated strains are actually from mitochondrial DNA, and therefore not on a chromosome. The chromosomes in the files are represented as capitalised Roman numerals, which are capitalised alphabetical characters. As other forms of DNA have prefixes with lowercase alphabetical characters (eg. MtDNA), the `grep` command can be used with the `-v` argument and the regular expression given below to filter out these

types of DNA.

```
grep -v '[+a-z]'
```

The amount of unique elements in the column can be counted with `wc -l`. This returns six elements, not including the mitochondrial DNA. These elements are chromosomes 1, 2, 3, 4, 5, and 10. The complete code is included below.

```
no_chr=$( cut -f4 $file | sed "1 d"| sort | uniq | grep -v '[+a-z]' | wc -l )
echo 'Number of chromosomes: ' $no_chr
```

3.2

The feature on which the mutation is situated is stored in column six of the mutated string text file. To determine the number of mutations on both introns and exons, we must return the number of entries in this column which are introns, and the number of entries which are exons. This is done using the `awk` function, and then passing a string comparison as an argument, as shown below. This output is then piped to `wc -l` to count the number of lines present.

```
mut_intron=$( awk '( $8=="intron" )' $file | wc -l )
echo 'Number of mutations on introns: ' $mut_intron

mut_exon=$( awk '( $8=="coding_exon" )' $file | wc -l)
echo 'Number of mutations on exons: ' $mut_exon
```

Number of mutations on introns: 576
Number of mutations on exons: 607

3.3

To return the types of non-synonymous mutations found on exons, we must first access only those mutations that are on exons. This was already demonstrated in the previous question (see listing...). To return the non-synonymous mutations, we must only add a second condition to our `awk` command, checking that the entry in the 11th column is not equal to "synonymous":

```
awk '( $8=="coding_exon" && $11!="synonymous" )' $file
```

Now using `cut -f11` to access the "effect" column and then returning the unique values, we can see the types of non-synonymous mutations found in exons. The complete code is included below.

```
non_syn_effects=$( awk '( $8=="coding_exon" && $11!="synonymous" )'\
                   $file | cut -f11 | sort | uniq )
echo 'Types of non-synonymous mutations: ' $non_syn_effects
```

Types of non-synonymous mutations on exons: Missense and Nonsense

4 Question Four

We first want to return the names of all the files which do not contain the substance, so that we can search them later for names and numbers. To do this we use `grep` with the `-r` and `-L` arguments. The `-r` argument applies `grep` recursively, so that we can access the files in the 'media' folder from a level up in the file system. The `-L` argument returns all the files in which there is no match. Using this command to search for the substance "HCL" would return the names of all files that do not contain HCL. An example output would be:

⋮

DSMZ_Medium1650.txt

DSMZ_Medium1651.txt

DSMZ_Medium1653.txt

DSMZ_Medium1654.txt

⋮

We now want to search each of these files for the number and name of the medium. Looking at the files, each medium number consists of four numerical characters, followed by a period. In Bash regular expressions, the "." character is a special character, and so we must escape it using backslash. The name is listed on the same line as the number. Using `grep` with a regular expression would allow us to find the lines that match this pattern.

```
grep -r "[0-9][0-9][0-9][0-9]\."
```

However looking at the media files, there are other instances where this pattern occurs, for example with measurements "1000.00 g". Noting that the media number is always at the beginning of the line allows us to update the regular expression to avoid this. In Bash regular expressions, the '^' character can be used to return patterns that only occur at the beginning of a line. Furthermore, `grep` also returns the names of the files, along with the lines in which the pattern matches. To output the lines exclusively, we can use the `-h` argument. The final regular expression is given below.

```
grep -rh "^[0-9][0-9][0-9][0-9]\."
```

Piping the two commands listed above together and then wrapping them in a function gives the final script:

```
function names_numbers () {
  grep -rL "$1" media | grep -rh "^[0-9][0-9][0-9][0-9]\."
}

names_numbers "Yeast extract"
```

If the user calls the function like so: `names_numbers "Yeast extract"`, the script above will take the first argument given to the function and treat it as a string. But what if the input string has spaces and the user does not wrap the input in quotes? For example, `names_numbers Yeast extract`. Then the function will take only "Yeast" as the first argument. This issue can be avoided by taking every argument given to the function and treating them as a single string, with spaces in-between each argument. The code is given below.

```
function names_numbers_spaces () {
  substance="'$*'"
  grep -rL "$substance" media | grep -rh "^[0-9][0-9][0-9][0-9]\."
}

names_numbers_spaces Yeast extract
```

References

[1] National center for biotechnology information. `https://www.ncbi.nlm.nih.gov/`. Accessed: 2019-11-5.

[2] Biomart id conversion tool. `http://www.biomart.org/notice.html`. Accessed: 2019-11-5.

[3] Ncbi homologene. https://www.ncbi.nlm.nih.gov/homologene. Accessed: 2019-11-3.

[4] Ncbi nucleotide blast. https://blast.ncbi.nlm.nih.gov/Blast.cgi?PAGE_TYPE=BlastSearch. Accessed: 2019-11-2.

[5] Clustal omega multiple sequence alignment. https://www.ebi.ac.uk/Tools/msa/clustalo/. Accessed: 2019-11-2.

[6] Mview multiple sequence alignment viewer. https://www.ebi.ac.uk/Tools/msa/mview/. Accessed: 2019-11-2.