Ph.D. position in Jan Gorodkin lab

Evaluation Assignment on CRISPR

Here I describe and report how I developed the scripts for the Ph.D. position assignment.

Firstly, I Download the necessary files from the terminal:

```
wget ftp://ftp.ebi.ac.uk/pub/databases/gencode/Gencode_human/release_35/
GRCh38.primary_assembly.genome.fa.gz

wget ftp://ftp.ebi.ac.uk/pub/databases/gencode/Gencode_human/release_35/
gencode.v35.annotation.gtf.gz
```

Now I create a working directory and copy the two files there, where I will also create the Python scripts. I decompress the two files:

```
gunzip gencode.v35.annotation.gtf.gz

gunzip GRCh38.primary_assembly.genome.fa.gz
```

I look for information about the General Transfer Format (GTF) files, and how they are structured, to understand how to work with them. I explore the data lines of these files in Python and understand the importance of each field and column.

Assignment part a)

First, I develop the function to parse the DNA sequence from the FASTA file belonging to the desired query. I check the number of base pairs parsed from chromosome 22, and I get 51,665,451. I compare this length in online databases, and I see that this number makes sense.

Now I develop the function to find the positions in the reference genome assembly of the 5’ UTR of chromosome 22. I make sure that all UTRs listed in the file belong to protein-coding genes. Using NumPy’s unique() function, I discover there are other types of genes with UTR fields:

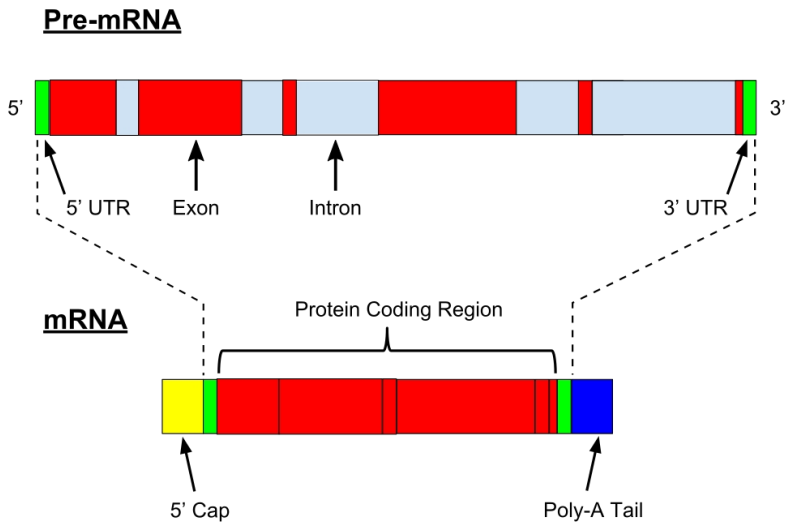
['"IG_C_gene";' '"IG_J_gene";' '"IG_V_gene";' '"polymorphic_pseudogene";'
 '"protein_coding";']

I filter out the ones that don’t belong to the protein_coding type. Now I need to figure out which UTR is the 5’ for each gene. First, I collect all UTR positions from the same gene ID. When a new gene is being parsed, I compute which is the 5’ UTR of the previous gene:

- If the UTR comes from the sense strand (+), I choose the UTR region with the smallest starting position. This UTR is the first among the collected ones that is closer to the 5’ end of the sequence, and is the region upstream the coding region (CDS) in the primary transcript and the mRNA.

- If the UTR comes from the anti-sense strand (-), I choose the UTR region with the largest starting position. In the anti-sense DNA strand, the direction of the transcription is reverted in comparison to the sense strand, and the 5’ UTR region in the mRNA corresponds to the

one closer to the 3’ end of the sense DNA sequence, which is the one we extracted from the FASTA file.



By Nastypatty - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=49282051

I save the coordinates of the starting position and ending position of each 5’ UTR as a list of tuples. I obtain a total of 435 5’ UTRs. I learn that there are around 414 genes in chromosome 22 (checked here), so I guess this number makes sense. I extract the DNA sequence belonging to these UTR coordinates and collect a list of these DNA fragments.

Assignment part b)

Now I want to scan these fragments to design all possible guide RNAs. I need to look for fragments of 20 nucleotides followed by the PAM signal (NGG). I perform this search in each 5’ UTR using a regular expression and the *re* Python library. I conduct this search in both the straight DNA and the reverse complementary of these DNA sequences because the guide RNAs can hybridize with any of the two strands. The method *re.findall()* returns only non-overlapping matches, but I am also interested in the overlapping matches. To achieve this, I create a lookahead pattern *(r'(?=([AGCT]{21}GG))'),* and capture the matches by group. This way, the string is not consumed after each match, allowing to collect also overlapping matches.

I could have also extracted the strand of each 5’ UTR and then use this information to construct the reverse complementary of the DNA sequence for the anti-sense strand. However, because the match scanning is performed in both DNA senses, I decided to omit this step.

Now I design the guide RNA for each of the obtained matches. I do this by extracting only the first 20 nucleotides of each match (the PAM sequence is left out). The gRNA is constructed as the transcription of the DNA for that match, obtaining a complementary RNA sequence that is able to hybridize with it.

Assignment part c)

After thinking and reading about ways to rank the designed gRNA, I decided to rank them by the number of Hydrogen bonds formed during DNA-gRNA hybridization. The H-bonds are an important factor that contributes toward the stability of these DNA-RNA complexes, and the computation is easy to implement:

- I give a score of 2 for A-T pairs (because they form 2 Hydrogen bonds)
- I give a score of 3 for G-C pairs (because they form 3 Hydrogen bonds)

The score of each gRNA is computed by summing the score of each nucleotide in its sequence composition.

Apart from the gRNA stability when forming the DNA-RNA complex with the desired target, these gRNAs can also match with undesired matches in the reference sequence (namely off-targets). I have also implemented a way to penalize each guide RNA by the total number of targets that are complementary to this sequence in chromosome 22 DNA sequence. Uniquely-matching targets will have only 1 match, but I see that some of these designs are able to match in tens and even hundreds of other regions. A quick formula that came to my head was:

$$\text{Score}_{\text{gRNA}} = \text{number of H-bonds} - \text{total number of targets}$$

Of course, scanning for potential targets for each gRNA of each 5'UTR region in the chromosome 22 is the most computationally-expensive part of this script in the way that I have implemented it. One could use a suffix array and the Burrows-Wheeler transform to speed up this process, but I left it like this for simplicity and available time.

After computing this score, the script prints the obtained results for the user. The output of this code is a line for each 5'UTR region, which contains a list of tuples. Each tuple has the sequence of the designed guide, along with the score given to this gRNA. The tuples are ordered in descending order, with the top hits first and the worst hits last:

[(58, 'CCGCUCGGCCCGGCCCGCCG'),
(57, 'GGCCCGGCCCGCCGUCCCUC'),
(57, 'CGCUCGGCCCGGCCCGCCGU'),
(56, 'UGACCCGCUCGGCCCGGCCC'),
(56, 'GGAGAGGGACGGCGGGCCGG'),
…,]

This script can be run from the terminal. There are 3 required arguments:

1. Fasta file containing the reference genome
2. GTF file containing genome annotation
3. Number of the chromosome to work with

Example command for running the script:

```
python3 design_5utr_guideRNA.py GRCh38.primary_assembly.genome.fa
gencode.v35.annotation.gtf 22
```