Akanksha Pabari, Keshav Haranath

Professor Koehl

ECS 129

2 March 2022

PROJECT 5: Gene Finder

Introduction

Analyzing the genome of an organism is an essential task key for many processes of bioinformatics. Genomes of organisms can range widely in size, with the Australian lungfish boasting a genome of 43 billion bases - fourteen times larger than that of the human genome. At the most basic level, the sequence of nucleotides of a DNA form the basis for transcription of an mRNA that can be translated to manufacture specific amino acid sequences that encode a protein. However, only specific regions of DNA in a genome directly code for the sequence of amino acids. This specific region of DNA is called the Open Reading Frame, or ORF. Finding the ORF in the billions of basepairs of a DNA sequence would be next to impossible if not for the advent of computer programs. In order to find ORF of a DNA sequence, a precise and efficient algorithm was developed in this project.

Previous algorithms have been designed to find the ORF of a DNA sequence. A notable example by the National Center for Biotechnology Information is an 'ORF Finder' algorithm that allows users to input a sample DNA sequence. The program allows the user to set specific parameters for the parameters, including length of ORF's, number of ORF's, start of ORF, and type of genetic code. The program also offers the user the option to select which strand (direct or reverse) the program should scan for an open reading frame. The ORF Finder algorithm was an inspiration for this project. However, the ORF Finder algorithm does not allow for users to view the effects of mutations to their ORF on resulting amino-acid sequences without analyzing the mutated ORF separately. In order to solve this and provide users this ability, the goal of this project was to provide a fast, easy-to-use, and accurate ORF Finder algorithm that can direct users to create their own mutations of the ORF and compare the resulting mutated and original protein sequences.

Methods:

The process of converting a DNA sequence into a protein can be broken down into four steps: creation of the complementary strand, determination of the open reading frame (ORF), transcription into mRNA, and translation into an amino acid sequence. In addition, the program allows for user-directed mutation of the ORF.

Language Used: Python was the coding language used for this project. To successfully run the code, Python 3.7 (or later) must be used.

Creation of the complementary strand: Our algorithm makes use of the fact that Adenine (A) will bond with Thymine (T) and Cytosine (C) will bond with Guanine (G). This allows for our program to loop through the sequence and match up the bases accordingly using the function **compl_strand()**.

Determining open reading frame (ORF): Open reading frames are bound by a start codon and a stop codon. Since mRNA sequences are read three base pairs at a time, our program breaks down the DNA sequence into a series of triplets in a list to easily identify the presence of start and stop codons in the function **find_ORF()**. The algorithm measures the length of the gene present in the current ORF, then proceeds to look at the next ORF by shifting the start point of the DNA sequence over by one index using the function **process_seq()**. After each ORF is produced, the length of the gene present is compared to the other ORF's until the longest gene is generated from the six open reading frames with the function **process_all_seq()**.

Mutating the Open Reading Frame (ORF): To go 'beyond the prompt', the program allows the user to mutate the Open Reading Frame that was found in the previous part of the algorithm. Users are allowed to perform the three main types of mutations seen *in vivo*: substitution of a specific nucleotide in the ORF with another valid DNA nucleotide, insertion of a specific nucleotide at a specific position in the ORF, and deletion of a specific nucleotide in the ORF. Afterwards, the mutated ORF is presented for transcription and translation. To accomplish this, the ORF is turned into a list so that it can be edited through utilizing the properties of python lists. For substitution, the mutability of python lists was utilized and the command for changing an item of a list at a specific position was utilized (i.e. **list[index] = 'new value'**). For insertion, the **.insert(index, "value")** list method was used to insert the user's nucleotide at the user provided index. In addition, for deletion of a nucleotide, the **.pop(index)** method

was used to delete a nucleotide at the provided index. Once finished, the mutated ORF 'list' is passed to further programs to calculate the associated RNA and protein sequences.

Transcribing mRNA: After determining the longest ORF, transcription into mRNA involves the replacement of Thymine (T) into Uracil (U) with the function **dna2rna()**. Since both our DNA strand and complementary strand were set to be read from 5' - 3', the conversion to mRNA does not involve any other operations other than the replacement of Thymine.

Translating mRNA: Our algorithm utilized a dictionary to store the triplet codons and their corresponding amino acid letter. With this as a guide, our program looped through the longest gene and assigned the analogous amino acid letters in the function **RNAtoAA()**.

Brief Analysis and Notes:

When the ORF is presented for mutation and initially transformed into a list, the program does not transform the ORF back into a string to transcribe mRNA from it. This is due to the fact that the function **dna2rna()** utilizes a for loop to iterate through all items of the ORF 'list' and calculate the corresponding RNA nucleotide for each item. Since lists are ordered, the for loop of the **dna2rna()** function can handle the ORF in list form as input.

In addition, the program assumes that the ORF (even if mutated) will undergo transcription and translation. This means that mutations to the Start or Stop codons will not change whether the ORF is transcribed and translated or not. This is advantageous as it allows for an examination of the effect of specific mutations **in the ORF sequence itself** on the protein end product.

Results: Presentation (And How to Run the Code)

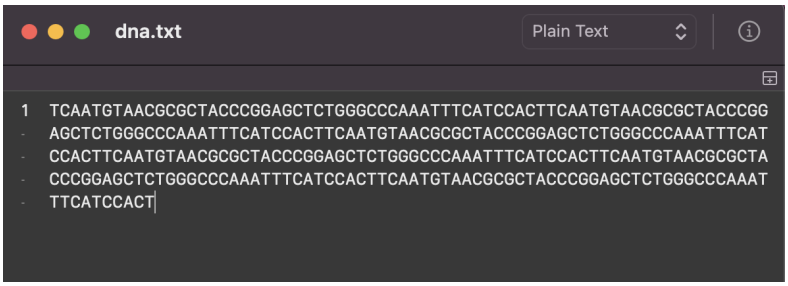
Files Included in the Program: The program contains 3 files, an input '.txt' file containing the DNA sequence to be analyzed, the file 'genes7.0.py', and the file 'genemodifyV5.py'. To run the program, only the file 'genes7.0.py' needs to be run. All 3 of these files should be in the same directory/folder as functions in the 'genemodifyV5.py' file are imported and used in the 'genes7.0.py' code.

Running the Code: To run the code from bash, use the 'cd' command in whatever Terminal program to navigate to the directory where all 3 files are located. From there, the code can be run by the bash command: 'python3 genes7.0.py'.

```
Keshavs-MacBook-Pro:~ keshav$ cd desktop
Keshavs-MacBook-Pro:desktop keshav$ cd ECS129
Keshavs-MacBook-Pro:ECS129 keshav$ python3 genes7.0.py
```

Formatting of the .txt file: The input file containing the DNA sequence **MUST** be a .txt file. After starting up the code, the program will prompt the user to enter the file name containing the DNA sequence. The name of the file name then can be provided. (**NOTE:** This file **must** be in the same directory as the genes7.0.py and genemodifyV5.py scripts required for the program). The **FIRST** line of the .txt file should contain **ONLY** the nucleotide sequence of the DNA from 5' to 3' (no need to include the 5' or 3' markers) and without extraneous spaces, special characters, or incorrect nucleotides (for example, 'X' or 'U'). As a whole the .txt file should contain **ONLY** one line (i.e. don't press 'enter' at the end of the DNA sequence because the program will interpret this as the addition of a special character to the DNA sequence). If incorrect input is detected, the program will prompt the user to recheck their input file.

Example of a properly formatted input file:



Example of an incorrectly formatted input file:

Calculation and presentation of DNA Strands and Open Reading Frame: After successfully inputting the input file containing the DNA sequence, the program presents the 'cleaned DNA strand' (i.e. the DNA input strand) and calculates the Complementary DNA strand to the input strand. Since the initial complementary strand will be in the 3' to 5' direction, the Reversed complementary strand (in 5' to 3'

direction) is presented. Afterwards, the algorithm will analyze both the DNA strand and the reversed Complementary DNA strand and identify the longest ORF sequence. The ORF sequence is then presented below.

Input of the sample DNA sequence 'TCAATGTAACGCGCTACCCGGAGCTCTGGGCCCAAATTTCATCCACT' yields the following result.

```
Name of DNA File: dna.txt
in file:  <_io.TextIOWrapper name='dna.txt' mode='r' encoding='UTF-8'>
No erroneous DNA input!
Cleaned DNA Strand: TCAATGTAACGCGCTACCCGGAGCTCTGGGCCCAAATTTCATCCACT
Cleaned Complementary DNA Strand:  AGTTACATTGCGCGATGGGCCTCGAGACCCGGGTTTAAAGTAGGTGA
Reversed Complementary DNA Strand:  AGTGGATGAAATTTGGGCCCAGAGCTCCGGGTAGCGCGTTACATTGA
Longest open reading frame: ATGAAATTTGGGCCCAGAGCTCCGGGTAGCGCGTTACATTGA
```

User-Directed Mutation of the Open Reading Frame Sequence: After successful calculation of the ORF, the ORF is then presented to the user so that the user can make specific mutations to the ORF. The user is presented with the length of the ORF and the option to induce mutations.

```
Longest open reading frame: ATGAAATTTGGGCCCAGAGCTCCGGGTAGCGCGTTACATTGA
Your ORF Length is 42
Choose a starting position to view ORF (Press 'q' or 'Q' to quit and continue to ORF translation):
```

At this step, the program directs the user to choose a window of 10 sequential nucleotides from the ORF so that the user can get a better idea of where to make mutations. In this step, the program asks the user to enter the starting position of this 10 nucleotide window. For example, if the user wants to view the nucleotides at positions 2-11 in the ORF, the user needs input the value '2'. In general, the X*th* nucleotide will be at the X*th* position, so inputting X as the starting position will show X*th* nucleotide to the X+9*th* nucleotide.

For example, if '2' is input in this step, the ten nucleotides at positions 2 - 11 are displayed.

```
Your ORF Length is 42
Choose a starting position to view ORF (Press 'q' or 'Q' to quit and continue to ORF translation): 2

2  3  4  5  6  7  8  9  10  11
T  G  A  A  A  T  T  T  G   G
```

If the user enters 'q' or 'Q' at the above step, the program will skip straight to transcription of the mRNA from the calculated ORF. Bogus/incorrect input at this step will make the program kindly remind the user why their input wasn't accepted, and then prod them to select a new window.

```
2    3    4    5    6    7    8    9    10   11
T    G    A    A    A    T    T    T    G    G

What position should the mutation be made at? (Type 'q' or 'Q' to quit and continue to ORF translation). To select a new window, type 'w' or 'W'):
```

Once the window has been selected, the program will then ask which position of the sequence the mutation should be created at. The user must enter a position value that is included in the ten positions displayed in the window. For example, input of ‘4’ will be accepted while input of ‘A’ or ‘36’ or ‘1’ or ‘-100’ will not be accepted. If the input ‘2’ is typed in the program will continue to ask the user what type of mutation should be made.

```
2    3    4    5    6    7    8    9    10   11
T    G    A    A    A    T    T    T    G    G

What position should the mutation be made at? (Type 'q' or 'Q' to quit and continue to ORF translation). To select a new window, type 'w' or 'W'): 2
What kind of mutation should be made at this position? Type 'S' for substitution, 'I' for insertion or 'D' for deletion. Type 'Q' to quit and continue to ORF
translation:
```

The options of mutations available at the selected position are **S**ubstitution, **I**nsertion, or **D**eletion. The user can input **S**, **I**, or **D** at the direction of the program to make their desired mutation. The user is given another chance to quit the mutation inducement and skip directly to ORF transcription and translation through input of ‘q’ or ‘Q’. The user can also press ‘w’ or ‘W’ to select another window to view.

If an **I**nsertion is desired at this position, then the program will prompt the user to input the nucleotide that should be inserted at the position they selected. The input **must** be a valid DNA nucleotide - Uracil or other nonsensical letters will not be accepted.

```
Choose a starting position to view ORF (Press 'q' or 'Q' to quit and continue to ORF translation): 2

2    3    4    5    6    7    8    9    10   11
T    G    A    A    A    T    T    T    G    G

What position should the mutation be made at? (Type 'q' or 'Q' to quit and continue to ORF translation). To select a new window, type 'w' or 'W'): 2
What kind of mutation should be made at this position? Type 'S' for substitution, 'I' for insertion or 'D' for deletion. Type 'Q' to quit and continue to ORF
translation: I
Insert what Nucleotide?: Q
Incorrect Input. Your input should either be 'A', 'C', 'T', or 'G'. Try again
Insert what Nucleotide?: G

2    3    4    5    6    7    8    9    10   11
G    T    G    A    A    A    T    T    T    G

Your ORF Length is 43
Choose a starting position to view ORF (Press 'q' or 'Q' to quit and continue to ORF translation):
```

After the nonsensical input was rejected, the user entered a valid input nucleotide. The nucleotide was then added at the position desired, and all other nucleotides in the ORF were shifted to new positions. The user then can choose a new window to continue the process and make further mutations of the ORF sequence. When the user is finished, they must type in ‘Q’ or ‘q’ to quit the mutation inducement part of the program.

After successful mutation of the sequence, the mutated and unmutated protein sequences are displayed along with the original transcribed RNA strand (5’ to 3’).

```
Choose a starting position to view ORF (Press 'q' or 'Q' to quit and continue to ORF translation): q
Your protein sequence: SEIWAQSSGRVTL
Original sequence RNA strand: AUGAAAUUUGGGCCCAGAGCUCCGGGUAGCGCGUUACAUUGA
Original sequence Protein sequence: MKFGPRAPGSALH
```

As a whole, the program looks like this:

```
Name of DNA File: dna.txt
in file:  <_io.TextIOWrapper name='dna.txt' mode='r' encoding='UTF-8'>
No erroneous DNA input!
Cleaned DNA Strand: TCAATGTAACGCGCTACCCGGAGCTCTGGGCCCAAATTTCATCCACT
Cleaned Complementary DNA Strand:  AGTTACATTGCGCGATGGGCCTCGAGACCCGGGTTTAAAGTAGGTGA
Reversed Complementary DNA Strand:  AGTGGATGAAATTTGGGCCCAGAGCTCCGGGTAGCGCGTTACATTGA
Longest open reading frame: ATGAAATTTGGGCCCAGAGCTCCGGGTAGCGCGTTACATTGA
Your ORF Length is 42
Choose a starting position to view ORF (Press 'q' or 'Q' to quit and continue to ORF translation): 2

2     3     4     5     6     7     8     9     10    11
T     G     A     A     A     T     T     T     G     G

What position should the mutation be made at? (Type 'q' or 'Q' to quit and continue to ORF translation). To select a new window, type 'w' or 'W'): 2
What kind of mutation should be made at this position? Type 'S' for substitution, 'I' for insertion or 'D' for deletion. Type 'Q' to quit and continue to ORF
translation: I
Insert what Nucleotide?: Q
Incorrect Input. Your input should either be 'A', 'C', 'T', or 'G'. Try again
Insert what Nucleotide?: G

2     3     4     5     6     7     8     9     10    11
G     T     G     A     A     A     T     T     T     G

Your ORF Length is 43
Choose a starting position to view ORF (Press 'q' or 'Q' to quit and continue to ORF translation): q
Your protein sequence: SEIWAQSSGRVTL
Original sequence RNA strand: AUGAAAUUUGGGCCCAGAGCUCCGGGUAGCGCGUUACAUUGA
Original sequence Protein sequence: MKFGPRAPGSALH
```

Results: Analysis

In approaching the problem of decoding a DNA sequence, our program chose to identify, transcribe, and translate the longest gene rather than all possible genes. Our team believed that identifying long genes could have implications for mutation rate and protein function. Researchers in the field have also discovered potential significance in gene length. Researchers at the University of Liverpool have proposed that longer and shorter genes tend to play differential roles during a human life. For example, the researchers proposed that “longer genes tend to be associated with functions that are important in the early development stages, while smaller genes tend to play a role in functions that are important throughout the whole life, like the immune system, which requires fast responses” (Lopez 2021).

Program analysis: Our algorithm currently involves three traversals each for the DNA strand and the complementary strand to gather the six total open reading frames. A possible optimization of this could be traversing each strand once while generating the three ORF lists for each. Another consequence of the structure of the program is its ability in handling large genome sequences. Looping through the list

multiple times over the course of transforming DNA to protein is computationally intensive. A possible workaround follows a similar principle to the BLOSUM62 matrix. Based on the idea of a score, one could assign a unique numerical value to the four nucleotides, and from there, implement a NumPy array for operations in the array (numpy.org). Our best understanding of the NumPy array is that it can efficiently help transcription and translation by leveraging well-defined pairing systems in the Central Dogma. Following the operations, the NumPy array could be converted back to a Python list.

Beyond the prompt: Our "beyond the prompt" idea involved offering the user the ability to introduce mutations into the longest gene our program provides. In the larger picture, this tool could be built into a game or app for elementary, middle school or high school students. The game would allow students to actively engage with the topic of DNA and proteins, while also getting to tinker around with different proteins. A possible extension would be linking the app/game to the RCSB Protein Data Bank (PDB) so that the student can learn more about the protein that they have constructed. In the scientific realm, our mutation program could allow scientists to better understand how mutations affect certain phenotypes. Using prior knowledge or data to examine locations with high probability of mutation, scientists could study the effects of the mutation on the creation of a valid protein sequence. Further studies can be completed by generating a 3-D visual of the mutated protein compared to its normal counterpart.

Discussion

The creation of an amino acid sequence from a DNA sequence is a significant step in deciphering the sequence and understanding its potential role in biological mechanisms. However, this process has a caveat: not every amino acid sequence that is created will ultimately lead to a protein. This leads to a question that our program does not address: how do you distinguish which amino acid sequences will lead to a protein and which will not? Computational biologists have taken on this challenge by developing a computational test that can predict - with 99.6% accuracy - the likelihood of an amino acid sequence forming a protein. The team's algorithm is based on mapping the individual vectors of amino acids in a sequence and drawing conclusions from its comparison with the *UniProtKB* database (Yau 2015).

Another topic of discussion relates to the process of reading the DNA sequence itself. Our program makes the assumption that the entire DNA sequence is entirely human. However, "Eight percent of our DNA consists of remnants of ancient viruses, and another 40 percent is made up of

repetitive strings of genetic letters that are also thought to have a viral origin" (Cold Spring Harbor Laboratory). Accounting for our viral nature requires a method of differentiating our own genome parts from that which comes from a virus. Scientists are already working to better illustrate the presence of viral genomes in human DNA through a program known as ViraMiner. ViraMiner employs machine learning techniques with the aim to precisely predict the probability of a given sequence being of human or viral origin (Tampuu 2019).

A side note on ethical use of this program: As developers of this program, our intention is to ensure a quick and easy experience for users to analyze sample DNA sequences. However, in compliance with discussions of ethics that took place during the ECS129 class, we believe it is necessary to clearly state how this program should be used ethically.

W**e implore (in the strongest terms possible)** users of this program to obtain informed consent from the sources of genetic samples. Federal Law, specifically, the Federal Policy for Protection of Human Subjects (the 'Common Rule') requires researchers to obtain informed consent from "human subjects" prior to experimentation. For DNA analysis, privacy is also a major concern as DNA sequences from specific ethnicities or people may be more easily identifiable than others due to genetic patterns. Sample input data and output data of this program should be treated **responsibly** and **ethically**, with the value of the health and safety of participants outweighing profit concerns at all points of usage of this program. This program is performed locally on the computer of the user, so there should be no issues with data breaches associated with this project. However, users should take care to keep inputs and outputs of this experiment confidential and safe. We also direct users to take precautions to protect the confidentiality of patients whose data will be analyzed by this program. We implore users to follow all relevant HIPAA rules at all times.

Bibliography

Arnold, Carrie. “The Non-Human Living inside of You.” *Cold Spring Harbor Laboratory*, 13 Apr. 2020, www.cshl.edu/the-non-human-living-inside-of-you/.

Lopes, Inês, et al. “Gene Size Matters: An Analysis of Gene Length in the Human Genome.” *Frontiers*, Frontiers, 1 Jan. 2021, www.frontiersin.org/articles/10.3389/fgene.2021.559998/full.

Tampuu, Ardi, et al. “Viraminer: Deep Learning on Raw DNA Sequences for Identifying Viral Genomes in Human Samples.” *PloS One*, Public Library of Science, 11 Sept. 2019, www.ncbi.nlm.nih.gov/pmc/articles/PMC6738585/.

Wheeler, David L, et al. “Database Resources of the National Center for Biotechnology.” *Nucleic Acids Research*, Oxford University Press, 1 Jan. 2003, www.ncbi.nlm.nih.gov/pmc/articles/PMC165480/.

Yau, Stephen S.-T., et al. “Distinguishing Proteins from Arbitrary Amino Acid Sequences.” *Nature News*, Nature Publishing Group, 22 Jan. 2015, www.nature.com/articles/srep07972.

“NumPy: The Absolute Basics for Beginners¶.” *NumPy*, numpy.org/doc/stable/user/absolute_beginners.html.