Protecting Privacy while Translating Protein Sequences and Saving ORF Positions

Gardner, William

ECS 129: Computational Structural Bioinformatics

Dr. Koehl

March 2, 2022

Introduction

To understand life, we must first understand the foundations that our very existences are built upon. If you strip down every creature there is one thing that connects us all, our use of deoxyribonucleic acids (DNA) to store genetic information. Understanding DNA sequencing was the first step to understanding protein synthesis and function.

Gene mapping was a very new concept to scientists until Robert Sinsheimer, then chancellor of University of California, Santa Cruz, proposed mapping the whole human genome in 1985. (Collins et al., 2003) It did not take long for the project to receive funding from the National Institute of Health and become the most famous project in genome sequencing. Sinsheimer's idea became the Human Genome Project that resulted in the first genetic map of the human genome in 2001. Human genomics became a large field of research in the medicine for disease prevention and treatment. An important leap forward for genomics was the first individual human genome sequenced in 2007 by Craig Venter. His sequence led to the discovery of several variants in genes associated with coronary artery disease, hypertension, and myocardial infarction. (Gonzaga et al., 2012) Through gene mapping of one individual we advanced our understanding of three different genetic diseases.

Today we have the complete genome sequences of over 3,500 organisms. New sequencing methods along with gene mapping algorithms have led to an explosion of personal genome sequencing. Private ownership of human genetic data is leading to the loss of individual genetic privacy. Examples of this phenomenon are all around us. Patients in confidential medical studies have had their identities released by "re-identification attacks" revealing their identity along with the diseases they have. (Hansson et al., 2016) Genetic privacy has become a big problem that Bioinformaticians are positioned well to address in the future.

The project I chose placed me in the footsteps of giants as they began the undertaking of the Human Genome Project. Given an unknown strand of DNA in the 5' to 3' direction I must identify all the possible protein sequences. To understand these protein coding regions, it is important to record their position on the DNA sequence. I am also posed with the issue of handling genetic sequences and must evaluate the most ethical and best programming solution to ensure privacy.

Note to User

To prevent directory errors, place this python program in its own folder with the .txt documents you plan to run through it. Also ensure the python on the user's computer has the modules: os, random, and hashlib installed. If this is a problem then you can run the code without the hash feature by commenting out lines: 8-10, 293, 295, 296, 322, 323 and then deleting functions: random_DNA, hash, hash_open_file, enter_values, access, search_secure, and hash_out_file. I hope you can install these modules and the use my hash functions.

Methods

My algorithm calls upon many different functions to ultimately return a complete protein sequence of all genes on the inputted DNA. Firstly, the user must input the DNA sequence in the form of a .txt document in the same folder as the Python program. The program will prompt the user to enter the filename. The function clean_DNA() preserves my programs strict input requirements by checking that the input DNA only contains "CGTA" and it is the user's responsibility to enter the code 5' to 3'. The DNA must all be one string of characters without any returns or newlines. Lower case "cgta" are also allowed as the program converts the input

into uppercase immediately. Inputs containing anything else will result in the program terminating.

My code then reads the inputted DNA sequence (now referred to as the template) and creates a complement strand of DNA by base pair matching. Then both codes are turned into mRNA strands by two different methods. First method: The complement's mRNA matches the template strand except it has all Thymine(T) instead of Uracil(U) so a simple .replace("T", "U") python function transforms the template DNA into a complement_RNA. Second method: For the template's mRNA it is slightly more complicated. First the code transcribes the template DNA by complementing the template by base pairing and also using the RNAise() function which replaces (T) with (U). Now there is an mRNA, but it is in the 3' to 5' direction and to be read by the ribosome it must be read from 5' to 3'. Using template_RNA[::-1] the mRNA is reversed in linear time. Now the template mRNA is oriented correctly for translation.

The ribosome can read an mRNA strand from three different positions and there is a complement DNA so there are two mRNAs being read and a total of 6 ways to read the template DNA. My algorithm handles translation in two parts. First the positions() function uses a window by iterating over the DNA with a while loop. The window is checking each letter the loop is on (pos), the one after it (pos+1), and the one two after(pos+2). This method is effectively imitating a codon size and checking for the start codon. If it does not detect AUG in the position, the loop advances one position and the window now moves up one, but the frame still remains three wide. This function returns a list of all positions of start codons. It then passes the position list onto the protein sequence builder named gene_list().

Gene_list() requires three variables, the mRNA sequence, the positions, and an Amino Acid table. Gene_list() first pops a value from the positions list and then runs it through the

translator() helper function. Translator() starts to build the protein sequence by indexing the mRNA using the position number that was popped using mRNA[pos:pos+3]. mRNA[pos:pos+3] results in a three-letter codon from the mRNA starting at the position found by the positions function. A while loop that is restricted by ensuring the mRNA is not shrunk smaller than a length of 3 letters keeps the protein sequence building. The codon is sent to a helper function that matches the codon with an Amino Acid table and returns the one letter code. It is then concatenated until it reaches a stop codon in which it is added to a dictionary. If it never reaches a stop codon the function returns a value of None and the gene_list() function does not add it to the dictionary.

I then formed a dictionary using the key as the protein's position respective to the template DNA and its value is the translated protein sequence. There are two different mRNAs that need to be translated. Therefore I utilize two different functions for gene_list(), gene_list() and gene_list_template() because the last argument in the function is a local variable. Keeping the template and complement mRNA separate allows me to concatenate the position with its respective relation to the template strand and keep its direction correct. When the complement mRNA gets run through the gene_list the position now has its direction added in front, reading (3' -> 5', pos). After both mRNA's have been iterated over the two dictionaries are merged. This is the complete dictionary of all the genes, including their positions respective to the template DNA.

The function that finds the longest protein sequence from the gene dictionary is simple. Values are stored as tuples to allow the position of the longest gene to stay associated with its sequence. It iterates through the dictionary comparing the protein sequence's length. If it is longer than the last value, then it replaces that value with the new longer sequence. In the rare

case that a sequence encounters a sequence of equal length it then appends it to a list. When the longest is a list then it only compares the length of one sequence in the list because they are all equal. The longest_gene () will return the longest protein sequence unless there are two or more equally long sequences, then it returns the list of tuples in (position, sequence) format.

Finally, to address the problem of genetic privacy my algorithm implements a hash function to keep the user's DNA sequence private. The hash function builds a hash table out of the hashed template DNA sequence as the key and the complete protein dictionary as it's value. The hash table stores the previous sequences in a .csv format and every time the algorithm is run again it reads the .csv file and builds a hash table into the function. The template DNA sequence is hashed using SHA 256 which is an unbroken hash function famous for being the basis of Bitcoin's cryptosecurity. If the user enters a DNA sequence already in the database, the search_secure() function catches it and returns the protein sequence stored in the hash table instead of rerunning the program on an already sequenced piece of DNA. When a new DNA is sequenced, it is finally hashed to ensure private storage that only the owner has the key to.

DNA is an incredibly unique variable and because of this it is incredibly rare that two sequences are the same. This attribute and SHA 256 prevent collision between variables in the hash table. When the protein is sequenced for the first time my algorithm returns: the total number of genes, the longest gene with its position, the gene density (No. of genes/Total bp), the complete gene map (which may be very lengthy so scroll to the top for other variables), and then hashes the sequence onto the hash table. The program then writes the complete .csv database into the same folder which the user can open and inspect. The first column is the hash code and the second is the hashed DNA's protein sequences. If the user wants to restart a new protein/genome

database, they just have to delete the Protein_database.csv file found in the same folder as this program.

My algorithm was built to handle large sequences of DNA. Originally my translation only had one function instead of one with two helper functions and was recursive to reduce time complexity. It ran into a maximum recursion depth error and when I extended the recursion depth of my system it resulted in stack overflow. It was reaching these errors around 3000 bp sequences which was unacceptable because the human genome is a million times longer than that. I redesigned my translation function so it could handle much larger DNA sequences but left the original translator function named translate_dict() to compare the new translation function to. My goal was to be able to sequence at least the smallest known living genome which the bacteria *Carsonella ruddii* proudly holds the record for with 159,662 bp. (Ball, 2006) My algorithm handles this genome size with ease. The run time of my algorithm is not incredibly fast for sequencing large sequences of DNA because of the many for loops it relies on for codon matching and base pairing.

There are portions of my algorithm that are hard to understand because I had to carry around the position variable associated with various start sequences. It resulted in the implementation of tuples and dictionary key value storing which also slowed down the program and decreased readability. The positions are also stored using array index values which starts from 0,1,2, etc. and therefore when you look to identify the start codon on the template it is actually one base further due to this. For example, reading the DNA 5' AACATG 3' results in the position of 3 instead of 4. I opted to keep this as the standard as this value is more useful for further coding of the DNA instead of having to constantly add 1 or subtract 1 from the values.

Results

From only one sequence of DNA my algorithm is able to return many important pieces of information. Firstly, an accurate translation of all possible Open Reading Frames on the DNA gives the user an accurate estimation of the number of genes the DNA encodes. It does not account for introns or upstream ORF's which results in an approximate overestimation of the number of genes by including multiple genes that all control one gene's expression or completely untranslated mRNA. It does however eliminate any sequence without a stop codon by removing any position with an incomplete protein sequence. Using a common bioinformatics technique, we could introduce a threshold to try and skip shorter protein sequences. Unfortunately, this runs the risk of possibly removing important genes on accident.

Gene density is a simple metric used for comparing DNA sequences importance. DNA strands with higher gene densities most likely are from high expression sections of the genome and vice versa.

Identifying all protein sequences' positions relative to the template DNA is a feature of my program with many real-world applications. Looking at the returned protein sequence the user can identify the start of each gene on the template DNA (Remember to count using the array method starting at 0 instead of 1). Locating hot spots of transcription helps to understand gene expression. (Fraser et al., 2007) Comparing gene positions to known positions of euchromatin and heterochromatin can help geneticists to understand translation frequency. My positioning system allows for this capability and could be applied to a well-researched genome. Given a fragment of DNA from an unknown position on the genome my code can sequence the fragment and help identify what section of the DNA the fragment is from.

Distance between genes is also important for understanding translation expression with many genes controlling just one protein's translational expression. Finally, positions are

important for comparison of DNA sequences between one another. Comparing two similar DNA sequences gene positions helps biologists to locate precise positions of genes and identify differences between individuals and observe those effects.

The function returns the longest gene identified across the whole DNA sequence. The relationship between gene length and function is an ongoing field of genomics research. It is currently believed that shorter gene sequences have higher translation expression while longer DNA sequences are highly conserved across species. (Lopes et al,. 2021) Therefore, locating the longest gene has possible applications in Phylogenetic Research as a measure of similarity between species.

The portion of my code I am the proudest of is the application of a hash table for genetic privacy and building a database of previously run sequences. SHA 256 is a cryptographic hash function that is capable of handling large inputs and returning a unique hash function. SHA 256 can handle inputs up to 2^64 bits. This along with its possibility of collision being virtually zero or 4.3*10^-60 makes SHA 256 the perfect hash function for my application. (Ramirez 2015) DNA sequences inputted into my database receive a unique hash code which promises the user's data is private. The personal information held in the database is all hashed, removing the identity of the user from the database.

The hash table served a second important use. It created a database that would store all previously sequenced DNA. This prevents multiple researchers from sequencing the same sample DNA multiple times and instead relies on one person doing it in the past. It also allows any user who has previously used this program to always have the ability to access their protein sequence.

Discussion

Taking on the role of translating DNA into a readable protein sequence proved invaluable. It allowed me to formulate my own algorithm for sequencing DNA strands that gave me a greater appreciation for modern sequencing algorithms. My algorithm relied on reading windows 3 bp wide over the entire strand which lent itself well to DNA because of DNA's degenerate trait. After days of trying to increase recursion depth for codon pairing, I shifted to iterative window reading for protein codon matching. My code is now able to accurately sequence over 1 million base pairs in less than a minute, an ability that I'm sure those working on the Human Genome Project would have envied.

My passion for genetic privacy drove my project in a completely new direction. How is anyone supposed to protect their genetic information in an age where medicine relies on genetic analyses provided by a privately owned healthcare industry? Even if the healthcare industry does not share our genetic data, it is still not secure. In 2019, nearly 10,000 research subjects had their medical information stolen in a security breach of Massachusetts General Hospital. (Landi 2019) The individual right to one's genetic information is of crucial importance and deserves further consideration. If corporations are able to classify you using your DNA, they could advertise to you on a biological level. For example, a person identified as having a genetic predisposition to alcoholism could be intentionally targeted by alcohol corporations to increase their sales without moral regard for that person.

The Genetic Information and Nondiscrimination Act of 2008 (GINA) was created to protect our genetic privacy, but it is not enough. This only prevents medical providers from releasing patients' information. There are still no checks on the government's ability to store records of individual citizen's genetics. The danger of unprotected genetic information in the

hands of immoral third parties rival its possession by an authoritarian power. The Chinese government has a database of each citizen's blood samples which it has used as a method to identify Uighurs, victims of China's ongoing genocide. (Simon 2018) Genetic privacy is no longer science fiction; it is a matter of life and death.

This project gave me the ability to further my understanding of Bioinformatics in an amazing way. Relating biology to coding challenged my biology knowledge more than any other class. I am incredibly proud of this program and only wish that this program shows the user how simple but incredibly necessary genetic privacy is.

Bibliography

Ball, P. (2006). Smallest genome clocks in at 182 genes. *Nature*, *2*(October), 10–12. https://doi.org/10.1038/msb4100090

Collins, F. S., Morgan, M., & Patrinos, A. (2003). The Human Genome Project: Lessons from large-scale biology. *Science*, *300*(5617), 286–290. https://doi.org/10.1126/science.1084564

Cortez, D. M. A., Sison, A. M., & Medina, R. P. (2020). Cryptographic Randomness Test of the Modified Hashing Function of SHA256 to Address Length Extension Attack. *ACM International Conference Proceeding Series*, 24–28. https://doi.org/10.1145/3390525.3390540

Daley, J. (2018). *Ambitious Project to Sequence Genomes of 1.5 Million Species Kicks Off | Smart News | Smithsonian*. 1–6. https://www.smithsonianmag.com/smart-news/ambitious-project-sequence-genomes-15-million-species-kicks-180970697/

Fraser, P., & Bickmore, W. (2007). Nuclear organization of the genome and the potential for gene regulation. *Nature*, *447*(7143), 413–417. https://doi.org/10.1038/nature05916

Hansson, M. G., Lochmüller, H., Riess, O., Schaefer, F., Orth, M., Rubinstein, Y., Molster, C., Dawkins, H., Taruscio, D., Posada, M., & Woods, S. (2016). *The risk of re-identi fi cation versus the need to identify individuals in rare disease research*. *August 2015*, 1553–1558. https://doi.org/10.1038/ejhg.2016.52

Gonzaga-Jauregui, C., Lupski, J. R., & Gibbs, R. A. (2012). Human genome sequencing in health and disease. *Annual Review of Medicine*, *63*, 35–61. https://doi.org/10.1146/annurev-med-051010-162644

Landi, B. H. (2019). *Massachusetts General Hospital privacy breach exposed 10 , 000 patients ' records , genetic information*. 1–5.

Lopes, I., Altab, G., Raina, P., & de Magalhães, J. P. (2021). Gene Size Matters: An Analysis of Gene Length in the Human Genome. *Frontiers in Genetics*, *12*(February). https://doi.org/10.3389/fgene.2021.559998

Ramirez, G. (2015). *MD5 : The broken algorithm now Overview of the broken algorithm : MD5*. *July 2015*, 6–11.

Schneider, R., & Grosschedl, R. (2007). Dynamics and interplay of nuclear architecture, genome organization, and gene expression. *Genes and Development*, *21*(23), 3027–3043. https://doi.org/10.1101/gad.1604607

Simon, S. (2019). *Uighurs And Genetic Surveillance In China : NPR*. https://www.npr.org/2019/12/07/785804791/uighurs-and-genetic-surveillance-in-china