Reconstruction of the Wooly Mammoth Genome

Claudia Molnar, Chandler Fajardo, Jace Rice & Jonathan Summers

Introduction

The wooly mammoth underwent extinction over 4,000 years, however a select few members of the species remain preserved in a frozen state and are being uncovered more and more frequently than before. In 2011, cells from a wooly mammoth that passed away showed signs of chemical activity after being implanted in mouse cells.[1] This finding was revolutionary because it proved that cell activity can still be recreated after death. The issue is that after so many years, the wooly mammoth's DNA has undergone significant damage so a revitalization of the species is impossible unless it's reconstructed. The purpose of this project is to develop different functions using Python code to help parse and process DNA, and ultimately work on reconstructing the wooly mammoth genome. This paper will explore the method in which these functions were created, and ultimately end in a discussion on how the project can be further expanded to other areas of study, not limited to the wooly mammoth revival.

Team Contribution Breakdown

Names & NetIDs	Roles	Milestone 1 Functions	Milestone 2 Functions
Claudia Molnar (cmolnar2)	Facilitator	s(dna) dna2rna(dna) reverse_complement(dna)	get_edges(dna_dict) assemble_genome(dna_list)
Chandler Fajardo (fajardo4)	Integrator	mendels_law(hom, het, rec) fibonacci_rabbits(n,k) gc_content(dna_list)	perfect_match(rna) random_genome(dna, gc_content)
Jace Rice (jacer2)	Recorder	count_dom_phenotype(genotypes) source_rna(protein) splice_rna(dna, intron_list)	find_splice(dna_motif, dna) shared_motif(dna_list)
Jonathan Summers (jgs5)	Strategist	rna2codon (rna) locate_substring(dna_snippet, dna) hamming_dist(dna1, dna2)	rev_palindrome(dna)

Technical Analysis

s(dna)

The purpose of the function s(dna) was to count the occurrences of the nucleotides A, G, T and C in a given DNA sequence and store the result in a dictionary. This function produces the correct output for DNA strings, but cannot be used with rna strings, reverse complements, etc. without being modified. The purpose of the function was relatively simple, so there were virtually no difficulties when it came to the execution of the code itself. The only issue that was encountered was that initially the answer was stored in a list, not a dictionary. This was debugged by turning the list into a dictionary, and storing each count with the name of the respective nucleotide as it's key (ie. ``'A' = count_A`). The function utilized material from class such as 'for' loops, dictionaries, 'if/else' statements, and the python function 'count', which was the primary tool used to count the occurrences of specific nucleophiles in the DNA string sample.

assemble_genome(dna_list)

The purpose of the function assemble_genome(dna_list) is to return the shortest superstring given a list of DNA strings. The method for constructing the superstring is as follows: if the last n-terms in the first string are the same as the first n-terms in the second string, the strings can be superimposed by removed the last n-terms of the first string and replacing it with the whole second string. The process is repeated until only one string remains. The submitted code functions for only certain test cases because it assumes that the first entry of the DNA list will always produce the shortest possible superstring, but it doesn't take into account cases where starting with the third entry will return the shortest superstring. There were multiple trials of 'for loops' and 'while loops' that were taken into consideration before the final code was achieved. A python visualizer was an especially useful tool when it came to debugging this function because it would visualize the code at each step of the program and made spotting errors a lot easier. The assemble_genome(dna_list) function utilized a great amount of class material that was covered. Splicing was used to superimpose two strings together based on the number of similarities they shared between the end of the first string and the beginning of the second string. Dictionaries were also used to store the different combinations of the words, and the shortest of the stored words would be utilized for the next step of the code. Multiple 'for' statements were

implemented that would loop through two strings in the list of strings looking for similarities, and these loops were nested in a while loop that kept the function running while there were at least 2 or more strings present in the list of strings.

splice_rna(dna, intron_list)

The purpose of this function is to take a string of DNA and convert it into a string of RNA and the subsequent list of introns that is spliced from the forementioned RNA string. This function does not return a value without the rna2codon code assembled earlier in the milestone as it uses that command. The debugging strategy used for this function was making sure that the value returned [from rna2codon()] was the same as used earlier. For example, if the end of this function returned a value for rna2codons(), but the function mentioned was actually rnatocodons(), splice_rna(dna, intron_list) would not return anything because the function names do not match despite verbally saying the same thing. This function used 'for' statements to search for variable x in the list of rna introns.

shared_motif(dna_list)

The purpose of this function is to take a list of dna strings and return a substring that contains the longest sequence of letters possible within the given dna string list. This function requires a list of dna strings to be given to work. Without the prerequisite of those dna strings it cannot return the substring within them. Debugging this string was difficult because first it scans the dna strings from the left and checks for a pattern in the letters to make the longest substring then checks the same thing but counting each letter from the right this time. The key was making sure the multiple 'if' statements were used correctly and were defining the right process. As for the application to the class, this function used lots of 'if' and '==true/false' to get the function to do the right things in the right order. If it started scanning from either direction at the wrong point in the sequence then the whole code would return either an incorrect value or no value.

rna2codon(rna)

The purpose of this function is to take a rna string representing a rna sequence. It will then return the corresponding amino acid string from the provided codon table. The first thing that needs to be done is storing the key-value pair in a dictionary in order to represent the codon

table. There were also for-loops that needed to be implemented for the length of the rna string. The protein string is then added until we reach the STOP value. The debugging strategy was not very difficult. It mostly involved altering the for-loops to ensure that the dictionary was being accessed correctly to return the appropriate amino acid string. It took a bit of trial and error but was not too complex. The function used a few concepts from class. Namely, the use of a dictionary to store the key-value pairs from the codon table. Also, it depended heavily on for-loops and if-statements for an efficient execution of the function.

def rev_palindrome(dna)

The purpose of this function is to take a DNA string and return a list of tuples. The list should contain the position and length of every reverse palindrome in the DNA string with length between 4 and 12. The reverse palindrome is a DNA string that is also equal to its reverse complement. So it would take the complement of the DNA string and flip the order but return the same string. There was a reverse_complement(dna) function from Milestone 1 that needed to be used in order for this function to work. The function traverses the entire string length and then goes the string from length 4 to 12. Finally, it checks if the string complement is a palindrome or not before returning the list. There were virtually no problems with creating the function. It depended heavily on the reverse_complement(dna) function previously created. The main modification was making sure that it could properly verify the string as a palindrome or not, which was a major feature of the function. From the class material, it largely used for-loops and if-statements to access the string and check for a proper palindrome.

mendels_law(hom, het, rec)

The purpose of the function was to find the probability that an organism would be receiving a dominant allele by receiving values for the amount of each type of pair of alleles. It is limited in the fact that it only accounts for punnett squares that only result in allele pairs. In the case of an organism that can have more traits depending on their ability to account for more alleles they receive, the code would need to be modified to account for this. While making the code, it was difficult to understand how to account for the correct pairings needed to satisfy the equation given. The lack of understanding is what made the debugging process so difficult and looking through the equations and where a mistake could have been made was tedious.

Eventually the math was corrected and the desired outcomes were achieved. In the code, while loops were implemented but in hindsight, a for loop would have been much better and compact. Additionally, the math.comb() function would have helped condense the code better rather than creating equations with a chance for error and mistakes.

perfect_match(rna)

The purpose of the perfect_match(rna) function was to accept an RNA string and produce the total number of perfect matchings of the nucleotide bases which is important in the RNA folding process. Figuring out how to consider if a pair was valid or not was the hardest part. Conceptually it was a hard to grasp function which is where most of the problems stemmed from. Once reaching that conceptual understanding, it was easy to implement and total up what is considered a pair. The function uses two for loops and checks to see if two requirements are satisfied before adding to the grand total of perfect matches.

Literature Review

Genomics and Endangered Species

A potential extension of this project could be to investigate genome sequences of endangered species, rather than extinct species. A journal published by the International Journal of Genomics details how genomic data "can be utilized to maximize population genetic variation while simultaneously reducing unintended introduction or propagation of undesirable phenotypes".[2] The paper focuses on a comparative genomic approach, which involves comparing an endangered species genome with a domesticated species genome that is in the same family. For example, the cat genome was used alongside the comparative genomics approach and compared with the genome of a cheetah in an effort to determine the susceptibility of the cheetah population to certain diseases, including both pre existing and emerging threats. The ability to compare the two species' genomes together is due to an evolutionary conservation of a specific type of nucleic acid (PCR-primer). This type of analysis can be extended to mammalian endangered species as well by using the mouse genome or human genome as the reference to the endangered species' genome.[2]

So far, the extent of this type of analysis has been to better understand the illnesses that endangered species are prone to that negatively impact their population size. Comparison of the

California condor genome with the chicken genome revealed how the genes involved in bone and cartilage formation affect the formation of chondrodystrophy in the condor species. This disease is lethal to the species, so understanding its genomic significance is important to potentially saving the species from extinction. This genomic analysis also found a mutation present in the condor's estrogen receptors which affects the receptor's sensitivity to certain chemicals and can affect the species' reproductive abilities.[2]

Much of the genomic comparative analysis involves similar DNA parsing and processing that was needed for the woolly mammoth genome reconstruction project. However, one of the major differences is that this genomic analysis involves comparing two sets of genomes to one another, something that we didn't necessarily deal with in the scope of this project. The project could be modified to possibly create a function that analyzes the different genes in the two sets of genomes (maybe multiple functions if analyzing more than one gene) and returns what the difference between the two gene is. The difference between the two could be returned or an additional function could be created that tackles how the genes could be altered so that the current structure won't pose a health risk to the species.

The data collected from the genomic analysis can be used as an insight on how to selectively breed certain members of an endangered species that guarantees that no genetic mutations are being passed down to future generations that could impact their survival. More commonly however, the data collected can be shared with veterinarians to help them better understand the genomic composition of endangered species. As a result, when these species have health issues, veterinarians can more easily diagnose and treat them because they are aware of certain genomic patterns in their genetic makeup that might cause the issues.

Neanderthal DNA in Modern Humans

Another extension of this project would be to look at the connection found between modern humans and neanderthals. A 2010 article by the name of, *Modern People Carry Around Neandertal DNA, Genome Reveals* details the discovery of neanderthal genomes inside modern-day human DNA. Although these genomes are between 1 and 4, this is still a great scientific discovery because it's the missing evidence to prove scientists' theory on neanderthal-human inbreeding. As it stands today, humans are the only hominoids on the planet but 650,000 years ago Neanderthals had their own branch on the hominoid tree until all traces of

them disappeared around 30,000 years ago. However, their disappearance was not caused by extinction or another kind of cataclysmic event. This evidence suggests that they were assimilated into the human pool over time.

This discovery was found by analyzing neanderthals, chimpanzees, and human DNA. The conclusion was that each hominoid was not too far away from the genetic make-up of one another. For example, it was discovered that only 73 proteins found in humans differed from the ones found in chimpanzees and Neanderthals meaning that very few changes in genes have occurred throughout the evolution of the hominoid tree. It is unsure if these changes occurred for some kind of genetic advantage but this is likely the case due to humans currently being at the pinnacle of intelligence in modern-day. If some kind of interbreeding did occur it was probably to make the human genome sequence stronger; combining both neanderthal and human DNA for a more well-rounded human.

The project would work very well in this experiment as well. Since there are functions that analyze DNA and scan them for matching sequences in other DNA strands. By using the functions we created in the project, more matching genomes or other DNA sequences may be found because of the specificity of our coding. This also goes for the wooly mammoth genomes. As chimpanzee, Neanderthal, and human DNA were compared to find matching genome sequences, the same things could be done with elephants and/or other members of the Mammalia tree.

Genomics: Testing the Limits of De-extinction

Another possible way to augment this project would be to consider the reality of how genomics would be used to bring back an extinct species. In an article from April 2022, titled "Genomics: Testing the Limits of De-extinction", the ways in which genomics can be used to revive ancient species is detailed. Similar to the movie, Jurassic Park, there have been countless experiments and attempts to revive extinct species such as the Wooly Mammoth and the Passenger Pigeon. Most of the methods to conduct such an experiment involved the cloning of the DNA and subsequent development in vitro, as well as using a reference genome from a closely related species to fill in the gaps from the ancient DNA fragments. In this project, we have been tasked with creating coding functions capable of sequencing and processing Mammoth DNA. This paper could take it a step further.

As stated, there are currently multiple ways to go about reviving an extinct species. However, one of the major setbacks for such goals is the fact that DNA recovered from ancient remains are often littered with chemical damage and heavily fragmented. This can severely hinder the reconstruction and recovery of their DNA. This would mean that for most ancient species, the complete sequencing of their DNA from the recovered fragments is essentially impossible. One of the most viable options is to use the fragments from the ancient DNA and map it to a closely related species' genome to reconstruct the extinct genome. This would make up for the gaps in the DNA, however, it would need to be a species that does not have too great of an evolutionary distance between itself and the extinct species. For the Wooly Mammoth, this could mean the Asian Elephant. There are several experiments already using this method with other species and they have varying degrees of success.

To wrap it up, the article brings about the realistic ways that the genome of an extinct species can be utilized to resurrect it. The DNA of an extinct species is rarely ever intact, but usually suffers significant damage over time. The main idea is the use of a closely-related living relative to reconstruct the DNA. This can be factored into the project by having students modify the raw DNA provided to create useful DNA sequences. They would need to develop code to sort out unusable DNA segments from DNA strings. They could then take DNA strings from a relative such as the Asian Elephant and piece them together into a final DNA sequence. It would surely complicate the coding process, but it would be the next logical step in advancing the project.

Assessment of Protein Coding Measures

A final way to extend this project would be to look more into recognizing the types of proteins that may be within the DNA strand. This can help look at exactly what kinds of traits can be produced with a given DNA string or possibilities of traits with offspring. An article by James Fickett and Chang-Shung Tung titled "Assessment of protein coding measures" discusses this topic of recognizing protein patterns in DNA.

The article discusses how an issue faced is how the recognition systems would need constant access to databases from a multitude of labs and experiments in order to recognize the proteins[5] . If a solution to efficiently allow the constant checking of various lab data to identify and see what proteins are present, it could possibly be simplified to be fitted for this project. A

much less complex version can be imitated to identify sections of DNA that match the patterns displayed in data. Knowing the proteins, the project can also be extended to figure out what the specific traits are after finding matching codes and possibly even the allele pairs that can be used to obtain such traits.

The usage of this article was to introduce the idea of a larger desire to use coding to evaluate and recognize protein DNA patterns and the possibilities it could bring to types of functions that can be created. Protein strings and their recognition bring a gateway of possibilities to a coding project in this area of study. While the article does go into much greater detail about methods they used, it must be noted how dated the article is and that their given issue may have already been solved. It would be a good prompt to give when explaining how to extend the project in this direction and how students can attempt this problem on their own terms in creative ways.

Conclusion

Our research has found that the scope of this project is not limited to genome reconstruction and the potential revitalization of the woolly mammoth. Genomics is a growing field whose relevance to computer science only grows stronger as technology advances. Genome assembly (and alteration) can be used to help bring species back from extinction and save endangered species from potential extinction. As more research is conducted on the subject, especially regarding the protein structures in DNA, the genome assembly may even be used to prevent medical anomalies in a wider range of species.

References

1. C. Ciaccia, “Woolly mammoth cells brought back to life in Shocking scientific achievement,” *Fox News*, 12-Mar-2019. [Online]. Available: https://www.foxnews.com/science/woolly-mammoth-cells-brought-back-to-life-in-shocking-scientific-achievement. [Accessed: 02-May-2022].
2. K. J. Irizarry, D. Bryant, J. Kalish, C. Eng, P. L. Schmidt, G. Barrett, and M. C. Barr, “Integrating genomic data sets for knowledge discovery: An informed approach to management of captive endangered species,” *International Journal of Genomics*, vol. 2016, pp. 1–12, 2016.
3. Saey, Tina Hesman. “Modern People Carry around Neandertal DNA, Genome Reveals.” *Science News*, vol. 177, no. 12, 2010, pp. 5–6, http://www.jstor.org/stable/25677885. Accessed 4 May 2022.
4. Schlebusch, Carina M. “Genomics: Testing the Limits of De-Extinction.” *ScienceDirect.com*, 11 Apr. 2022, https://www-sciencedirect-com.proxy2.library.illinois.edu/science/article/pii/S0960982222004183. Accessed 4 May 2022.
5. J. W. Fickett and C.-S. Tung, “Assessment of protein coding measures,” *Nucleic Acids Research*, vol. 20, no. 24, pp. 6441–6450, Dec. 1992.