Kai Hakomori Bio 131 Final

Repl link: https://repl.it/@Thankless/Final-Project
Note: I am okay with my professor, Anna Ritz, posting this writeup to the course webpage

My initial intention in embarking on this genomic journey was to investigate precisely how different our genes are from our neandertal cousins—more specifically, I wanted to see exactly how different our *most distinct* genes are from each other's counterpart orthologs. To that end, I wanted to develop three measures of comparison: the hamming distance of the genes' genetic context, the longest common sequence alignments of the gene's resultant products, and the "Grantham Score" of said alignments. To my (admittedly limited) knowledge, Grantham Scores are used to determine evolutionary distance between two orthologous proteins in closely related species, wherein amino acid replacements are given a quantifiable score that describes their distance from each other. For two such similar species, it seemed an appropriate test. However, it turns out that labeled neanderthal genes are exceedingly difficult to come by. As a result, I performed the same series of analyses on mice and humans instead.

For the first few steps of this process, I borrowed somewhat heavily from my earlier homework assignments, though with some adjustments to cut out the excess baggage. Transform(), reverse(), translate(), hamming(), and LCS(), for example, made their way into my program relatively unchanged. I also borrowed a couple functions from our homework assignments' helper function files—readFasta() and getAminoAcid(), to be precise—as they were necessary for my project, and I didn't think they really needed to be altered in any way. GetAminoAcid(), however, proved a very useful template for one of my new functions, which I will elaborate on later in this essay.

This is not to say, of course, that nothing changed. In order to adjust the exon start and end positions to the particular starting location of the genome sequence in question, I created a shift() function that takes as its argument either a list of exon starting positions and a string ('start', in this case), or a list of exon end positions, a string ('end'), and its corresponding list of exon start positions. If the string portion of the argument is 'start', the function subtracts the first element of its list argument from all other elements of the list, then returns a new list with those new values. If the string portion of the argument is 'end', the function takes the first element of the optional list argument (that is, the starting position of the first exon) and subtracts it from every element of the main list, then returns a new list with these values. This optional list argument is necessary because if we were to simply set both the exon-starts and the exon-ends

to 0, then all of the exons would be shifted back by the length of the first exon, reducing the number of exons overall and displacing their location in the genomic sequence.

Another noteworthy change is the rather different transcribe() function in my project. Besides adjusting it to take the dna strand type as an argument—and transform the exon positions accordingly—I also needed to add an argument for the mRNA's coding sequence location. This was something of a sticking point in the creation of this project. Despite verifying my computed exons with NCBI, the website I had pulled the original genome from, my translate function kept returning an incorrect (and lengthened) peptide chain. I tried quite a few different things to troubleshoot—computing the exon locations differently, using a different database, transcribing the complement strand, etc—but it was only once I attempted to translate an mRNA sequence that I had downloaded directly from NCBI (and got the same results) that I realized my mistake. When I scrolled through the mRNA's documentation to verify it was for the right protein, I noticed a section under "features" that provided a sequence labeled "CDS"—or "coding sequence", as I now understand it. In other words, I had to take a slice of the mRNA at this coding sequence's location before moving on to translation. After realizing this, implementing it was simple enough—I just set the default value of the CDS_pos argument to an empty list, and wrote the function such that it would take a slice of its computed mRNA from the value of CDS_pos' first element to the value of CDS_pos' second element should it be anything other than empty.

The biggest area of difference, though, is the addition of my GranthamScore() function. This is where the getAminoAcid() function became a truly useful reference, as without it I would not have thought to use a dictionary in my program, let alone a dictionary of dictionaries. The easiest way to generate a Grantham score for any particular pair of amino acids would be to reference a table, such as this one from wikipedia:

The only issue with this is the labor it takes to convert the table into a form that a python interpreter could interpret. My first thought was to create a series of nested if statements within a for loop, wherein every element of an amino acid alignment is matched to one condition, and then is matched to another condition within that condition, and then has that condition's score added to the

Difference *D* for amino acids[4]

| Arg | Leu | Pro | Thr | Ala | Val | Gly | Ile | Phe | Tyr | Cys | His | Gln | Asn | Lys | Asp | Glu | Met | Trp | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 110 | 145 | 74 | 58 | 99 | 124 | 56 | 142 | 155 | 144 | 112 | 89 | 68 | 46 | 121 | 65 | 80 | 135 | 177 | Ser |
| | 102 | 103 | 71 | 112 | 96 | 125 | 97 | 97 | 77 | 180 | 29 | 43 | 86 | 26 | 96 | 54 | 91 | 101 | Arg |
| | | 98 | 92 | 96 | 32 | 138 | 5 | 22 | 36 | 198 | 99 | 113 | 153 | 107 | 172 | 138 | 15 | 61 | Leu |
| | | | 38 | 27 | 68 | 42 | 95 | 114 | 110 | 169 | 77 | 76 | 91 | 103 | 108 | 93 | 87 | 147 | Pro |
| | | | | 58 | 69 | 59 | 89 | 103 | 92 | 149 | 47 | 42 | 65 | 78 | 85 | 65 | 81 | 128 | Thr |
| | | | | | 64 | 60 | 94 | 113 | 112 | 195 | 86 | 91 | 111 | 106 | 126 | 107 | 84 | 148 | Ala |
| | | | | | | 109 | 29 | 50 | 55 | 192 | 84 | 96 | 133 | 97 | 152 | 121 | 21 | 88 | Val |
| | | | | | | | 135 | 153 | 147 | 159 | 98 | 87 | 80 | 127 | 94 | 98 | 127 | 184 | Gly |
| | | | | | | | | 21 | 33 | 198 | 94 | 109 | 149 | 102 | 168 | 134 | 10 | 61 | Ile |
| | | | | | | | | | 22 | 205 | 100 | 116 | 158 | 102 | 177 | 140 | 28 | 40 | Phe |
| | | | | | | | | | | 194 | 83 | 99 | 143 | 85 | 160 | 122 | 36 | 37 | Tyr |
| | | | | | | | | | | | 174 | 154 | 139 | 202 | 154 | 170 | 196 | 215 | Cys |
| | | | | | | | | | | | | 24 | 68 | 32 | 81 | 40 | 87 | 115 | His |
| | | | | | | | | | | | | | 46 | 53 | 61 | 29 | 101 | 130 | Gln |
| | | | | | | | | | | | | | | 94 | 23 | 42 | 142 | 174 | Asn |
| | | | | | | | | | | | | | | | 101 | 56 | 95 | 110 | Lys |
| | | | | | | | | | | | | | | | | 45 | 160 | 181 | Asp |
| | | | | | | | | | | | | | | | | | 126 | 152 | Glu |
| | | | | | | | | | | | | | | | | | | 67 | Met |

total—only, it turns out that this process is not only labor intensive, but deeply confusing as well. After a few misplaced if statements and a series of errors, I realized that I could instead just create a dictionary where the value of each key is another dictionary. This way, I could basically just search by column and row. That said, this table is set up such that there is both a missing column and a missing row (Ser and Trp, respectively), in addition to all of the missing entries. With the column and row missing, I spent a while getting errors without understanding what was going wrong, but by adding one more dictionary and one more key:value pair to each dictionary, the problem was solved relatively painlessly. As for the other issue, it would probably be easier to explain if I just broke down the actual code within the function.

The GranthamScore() function takes as its argument two strings: peptide1 and peptide two. In my case, these are the two LCS alignments computed from my mouse and human RPTN peptides, but theoretically any peptide could be plugged into here. It first opens my Grantham score document and evaluates it as Python code, so it can recognize it as a dictionary. It then creates a variable "score" and sets it to zero. Within a for loop, it creates two variables that change with each loop: amino1 and amino2, representative of index i of their respective peptide strings. After an if statement that tells the function to skip the current iteration of the for loop if either amino acid is a "-" or a "*" (as a result of the alignment process or the stop codon), the function then pulls the value corresponding to the amino2 key in the amino1 dictionary and the value corresponding to the amino1 key in the amino2 dictionary. If the first value is 0, it adds the second value to the overall score and continues the for loop. If not, it adds the first value. In the end, it returns the overall score. In this way, it is possible to find the correct Grantham score for a pair of amino acids regardless of the order in which they appear, and without doubling the work needed to create the dictionary.

As the primary novel element of my project, I thought it was relevant to discuss the algorithm behind my GranthamScore() function. Otherwise, I think I will focus on the results. First, I took the hamming distance of the genomic context of each species' RPTN gene. This revealed a hamming distance of 4,195 nucleotides out of roughly 5,700 nucleotides. To all appearances, this in itself reveals a rather wide evolutionary distance between the two. However, to know more it is necessary to analyze further. And so, after transcribing and translating the two genes using their shifted exon positions and their CDS location, I attempted to find the longest common sequence alignment of the resultant proteins. This generated a longest common sequence of 507 amino acids out of 785 in the humans' case and 1119 in the mice's—by all accounts, a higher ratio, but not necessarily revelatory of any genetic or

evolutionary truths. The final step, then, was to calculate the Grantham Score. Unfortunately, I had started making this function with Neandertals in mind, and it is not actually a very useful measure of evolutionary or genetic distance in such distantly related species; it was meant, really, to examine a handful of amino acid replacements in otherwise identical peptides. As a result, the Grantham score I produced was a ridiculous 27,582. Normally, a score of over 100 is considered a fairly dramatic difference.

In any case, it is safe to say that mice and humans are very different, even on the scale of one gene that encodes one protein with a common purpose. If possible, though, I would like to someday apply the code I have developed to my original target, the neandertal, and to the other species who bore our semblances, such as the Denisovans. Ultimately, this program is best suited to comparing two closely related species, and I have long wanted to learn more about our interbreeding evolutionary cousins.