Den Ivanov
Intro to Computational Biology
Anna Ritz
11 May 2017

Final Project Writeup

The motivation of this project was to find a way to create degenerate DNA sequences from a series of identical-length sequences. The primary use for this algorithm would be for the design of “universal” primers (e.g. primers designed for the same gene across multiple species or sometimes families), but they have even been used in reverse to *identify* the species of bacterial pathogens present in samples of cerebrospinal fluid (Lu et al, 2000[1]). Data for this project were acquired from the supplemental data from a paper published by Dong et al in 2015 titled “*ycf1*, the most promising plastid DNA barcode of land plants”[2]. In this paper, they created taxon-specific primers for 131 families of plants ranging from bryophytes to gymnosperms, and then created universal primers for the bryophytes, monilophytes, gymnosperms, and angiosperms. I used the taxon-specific primers for a handful of the families by copying the list of primers given in the supplementary material (Table S3) and then ensuring that there were no erroneous spaces. So long as the list of primers were space-delineated, my algorithm will splice them into a list of strings as its first step. Next it loads in and initializes the scoring matrix I devised by hand. This matrix is used to score the primer created by my algorithm against the primers designed and published by Dong et al. The matrix was calculated by taking each pairwise combination of

---

[1] Lu, Jang-Jih, et al. "Use of PCR with universal primers and restriction endonuclease digestions for detection and identification of common bacterial pathogens in cerebrospinal fluid." *Journal of Clinical Microbiology* 38.6 (2000): 2076-2080.

[2] Dong, Wenpan, et al. "ycf1, the most promising plastid DNA barcode of land plants." *Scientific reports* 5 (2015): 8348.

degenerate nucleotide, creating a sub-matrix for that, and then comparing the number of matches compared to the size of the sub-matrix. Take as an example R vs D, which is equivalent to A or G vs A or G or T. This creates the following matrix:

	A	G	T
A	0	1	1
G	1	0	1

The matrix simplifies to 4/6, or 0.66. Doing this for every pairwise combination (of which there are about 113) yields this[3]:

	A	C	T	G	R	Y	S	W	K	M	B	D	H	V	N
A	**0**	1	1	1	0.5	1	1	0.5	1	0.5	1	0.33	0.33	0.33	0.25
C	1	**0**	1	1	1	0.5	0.5	1	1	0.5	0.33	0.33	1	0.33	0.25
T	1	1	**0**	1	0.5	1	0.5	1	0.5	1	0.33	0.33	1	0.33	0.25
G	1	1	1	**0**	1	0.5	1	0.5	0.5	1	0.33	0.33	0.33	1	0.25
R	0.5	1	0.5	1	**0**	1	0.25	0.25	0.25	0.25	0.17	0.33	0.17	0.33	0.5
Y	1	0.5	1	0.5	1	**0**	0.25	0.25	0.25	0.25	0.33	0.17	0.25	0.17	0.5
S	1	0.5	0.5	1	0.25	0.25	**0**	1	0.25	0.25	0.33	0.17	0.17	0.33	0.5
W	0.5	1	1	0.5	0.25	0.25	1	**0**	0.25	1	0.17	0.33	0.33	0.17	0.5
K	1	1	0.5	0.5	0.25	0.25	0.25	0.25	**0**	1	0.33	0.33	0.17	0.17	0.5
M	0.5	0.5	1	1	0.25	0.25	0.25	1	1	**0**	0.17	0.17	0.33	0.33	0.5
B	1	0.33	0.33	0.33	0.17	0.33	0.33	0.17	0.33	0.17	**0**	0.22	0.22	0.22	0.75
D	0.33	1	0.33	0.33	0.33	0.17	0.17	0.33	0.33	0.17	0.22	**0**	0.22	0.22	0.75
H	0.33	0.33	1	0.33	0.17	0.33	0.17	0.33	0.17	0.33	0.22	0.22	**0**	0.22	0.75
V	0.33	0.33	0.33	1	0.33	0.17	0.33	0.17	0.17	0.33	0.22	0.22	0.22	**0**	0.75
N	0.25	0.25	0.25	0.25	0.5	0.5	0.5	0.5	0.5	0.5	0.75	0.75	0.75	0.75	**0**

Following the initialization of this matrix, I proceed to ignore it for another few functions. Next I implemented a standard “count matrix” algorithm as we implemented in HW5. It was tweaked slightly to best fit this project (notably: no pseudocounts), but left mostly untouched. At this point, I begin to venture into new territory with the meat of this function: degen(). It takes as its input the count matrix created by my prior function, as well as a “tolerance” value (ranging from 0.0 to 1.0) for the math.isclose() function it

[3] Again: I did this by hand. It was awful.

utilizes. The function operates as follows: iterating through the count matrix, at each step, it finds the most frequently occurring nucleotide. At this point, it checks for “near matches” using the math.isclose() function. This function compares two values and checks if they are within a given tolerance value (which can be absolute or relative, I used relative), returning true or false depending. Based on which nucleotide the maximum is and what other nucleotides are approximately (my default threshold was 10%) equal to it, it appends the nucleotide based on the following IUPAC nomenclature[4] to the primer as it assembles:

Symbol[1]	Description	Bases represented				
A	Adenine	A				1
C	Cytosine		C			
G	Guanine			G		
T	Thymine				T	
U	Uracil				U	
W	Weak	A			T	2
S	Strong		C	G		
M	aMino	A	C			
K	Keto			G	T	
R	puRine	A		G		
Y	pYrimidine		C		T	
B	not A (B comes after A)		C	G	T	3
D	not C (D comes after C)	A		G	T	
H	not G (H comes after G)	A	C		T	
V	not T (V comes after T and U)	A	C	G		
N	any Nucleotide (not a gap)	A	C	G	T	4
Z	Zero					0

Now it’s time to bring back that hideous scoring matrix. Now that the degenerate primer has been calculated, it compares its length to the length of the published primer (if one exists). If they are not the same length, it prints such and the function ends. If they are

[4] https://en.wikipedia.org/wiki/Nucleic_acid_notation

the same length, it implements a partial hamming distance based on the score. Iterating through the two strings, it finds the index in the scoring matrix based on the value of each string at that step of the iteration and adds the value at that index to a variable score. It does this for every position in the primer and spits out the total score at the end such that you can compare the two strings and test how robust the function/chosen threshold is.