Jennifer Yu Bi/CS183 PSET #1

1. (1) Using the letters "ATCG", the length 3 strings AAA, TTT, CCC and GGG must not appear in a comma free code. Concatenating, for example, AAA with itself would result in a valid overlapping code, which violates the concept of a comma free code with $|\Sigma|=4$, $K=3$.

(2) For the remaining length 3-strings, we can group them into 20 cycles of length 3 strings. They are listed below:

TTC	TTA	TTG	TCC	TAA	TGG	TGC	TGA	TGC	AGC	ACG
CTT	ATT	GTT	CTC	ATA	GTG	CTG	ATG	CTG	CAG	GAC
TCT	TAT	TGT	CCT	AAT	GGT	GCT	GAT	GCT	GCA	CGA
CGT	AGT	GAG	GTC	CCA	CCG	AAG	AAC	GGC		GACT
TCG	TAG	AGG	CGT	ACC	GCC	GAA	AAA	CGG		TAC
GTC	GTA	GGA	TCG	CAC	CGC	AGA	ACA	GCG		CTA

From each triple, in order to form a comma free code, two of the length 3 strings must be removed. For example, if TTC is concatenated to itself ⇒ TTCTTC, there exists overlaps which form TCT and CTT, the other two length 3-strings in its cycle. Thus, these strings cannot appear with TTC at the same time in the comma free code. This eliminates 40 more possibilities in the construction of a comma free code, so the maximum amount of length 3 strings in a comma free code is 20.

(3) Based on (1)(2), we determined 20 groups of triples of length 3 strings can be used to form comma free codes with $|\Sigma|=4$, $K=3$. For an upper bound, we consider all possible ways to combine strings from the 20 groups. From each group of 3, we choose 1 string and there are 20 groups:

$$\binom{3}{1}^{20} = 3^{20}$$

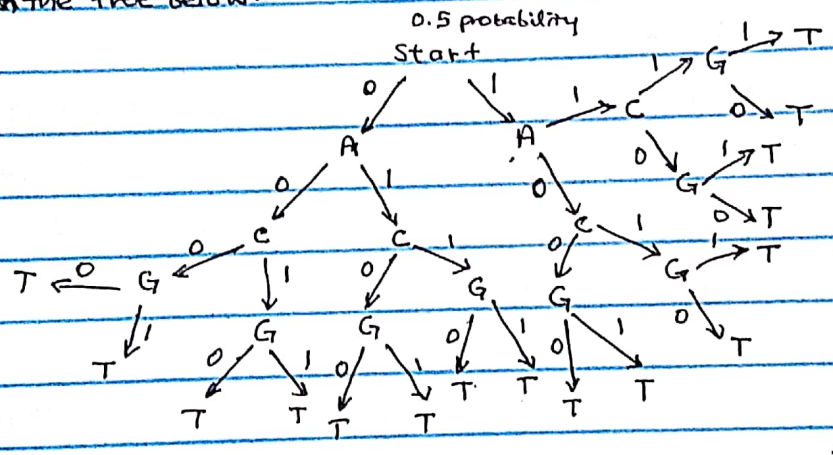
2. K=2, l=4 (A, C, G, T), n=4

initial probability of picking state 1 or state 2: 0.5, 0.5

	0	1
0	0.8	0.2
1	0.05	0.95

	A	C	G	T
0	0.2	0.5	0.1	0.2
1	0.1	0.25	0.25	0.4

To compute P_{ACGT}, we need to add up the probabilities of each path ending at T in the tree below:



$0.5(0.2)(0.8)(0.5)(0.8)(0.1)(0.8)(0.2) = P_{ACGT}$ with 0 state drive every time

⋮

In the code ~~on the next page~~ provided, the code for determining the probability of ACGT is written. The final answer is around $\boxed{0.00317967187 5}$.

What the code does is compute the probabilities through recursion. It shaves down the bases one at a time from the sequence until there is only one letter left (base case). Then, the probability of obtaining that base (in this case, T) is determined with the inputted parameters of the previous state, the current state and the base, which is returned. The recursive cases then collapse; note that there are two recursive calls, one with the new state as 0 and one with the new state as 1.

3. $k=l=2, n=3 \Rightarrow P_{011}=P_{110}, P_{100}=P_{001}$

We want to show that P_{000} and P_{111} cannot both be 0. First, we assume for contradiction that $P_{000}=P_{111}=0$. As shown in the code in my jupyter notebook, the probabilities of P_{000} and P_{111} are as follows: ($A_1 \Rightarrow$ probability of 1 in state A; $AB \Rightarrow$ prob of state A→B; init(A) = initial prob. of A)

P_{111}: $init(A)\,A_1(AA\cdot A_1\cdot AA\cdot A_1 + AA\cdot A_1\cdot AB\cdot B_1 + AB\cdot B_1\cdot BA\cdot A_1 + AB\cdot B_1\cdot BB\cdot B_1) +$

$init(B)\,B_1(BA\cdot A_1\cdot AA\cdot A_1 + BA\cdot A_1\cdot AB\cdot B_1 + BB\cdot B_1\cdot BA\cdot A_1 + BB\cdot B_1\cdot BB\cdot B_1)$

P_{000}: $init(A)\,A_0(AA\cdot A_0\cdot AA\cdot A_0 + AA\cdot A_0\cdot AB\cdot B_0 + AB\cdot B_0\cdot BA\cdot A_0 + AB\cdot B_0\cdot BB\cdot B_0)$ * + large term refers to those with sums of products

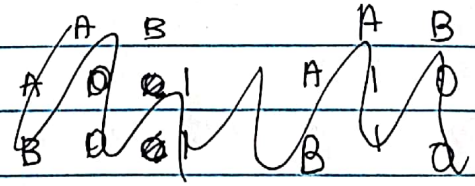
$init(B)\,B_0(BA\cdot A_0\cdot AA\cdot A_0 + BA\cdot A_0\cdot AB\cdot B_0 + BB\cdot B_0\cdot BA\cdot A_0 + BB\cdot B_0\cdot BB\cdot B_0)$

As we can see, we need both terms in the equations for P_{000} and P_{111} to be equal to 0 for $P_{000}=P_{111}=0$. Thus, we need to consider these cases:

1) $init(A)=0$ only ($init(B)>0$)

If this were the case, then we would have to look at shared terms to see how the second large term could produce a 0, since the first large term, multiplied by $init(A)=0$, is now 0. We see that only the terms BA and BB, if both equal to 0, can make the second large term become 0. However, then the transition matrix would become:

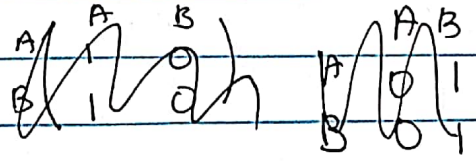
	A	B
A	1	1
B	0	0



Which is not valid.

2) $init(B)=0$ only ($init(A)>0$)

This time, we see that terms $AA=AB=0$ can make the first large term become 0 and $T=$



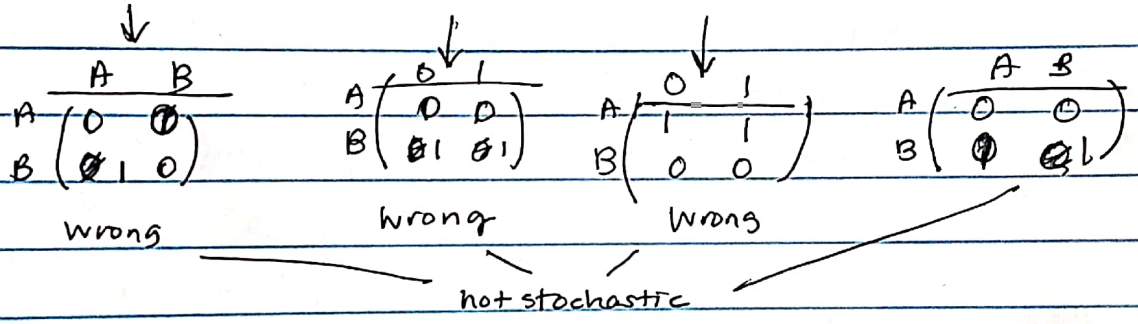
	A	B
A	0	0
B	1	1

Which is not valid.

3) init(A) > 0, init(B) > 0

I coded the impossibility of obtaining an adequate matrix in Case 3. under Problem 3 of my homework. You end up with at best, three numbers that must be 0 in order for $P_{111} = P_{000} = 0$. They are:

(AA, AB, BB) (AA, B_0, B_1) (BB, A_0, A_1) (AA, BA, BB)

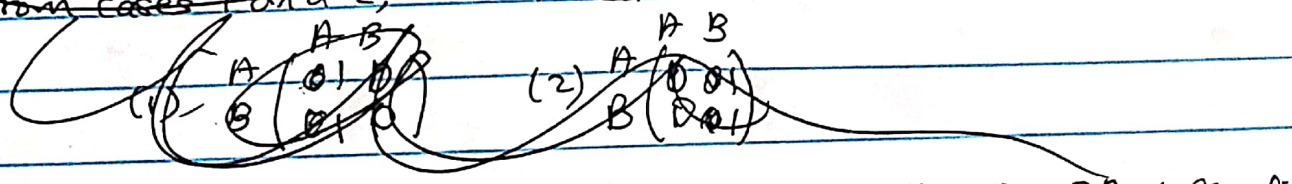


None of these matrices are valid for transition or emission, so we cannot determine a solution for nonzero initial probabilities.

4) init(A) = 0, init(b) = 0

This case is not relevant, because at least one initial probability must be greater than 0 for evaluation.

From cases 1 and 2, we obtained the transition matrices:



We can now try to plug in the matrices to the $P_{100} = P_{001}$ equality.

(1) AA=0, AB=0, BA=1, BB=1

P_{100}: init(B) B_1 ($B_0A_0 + B_0^2$) , P_{001}: init(B) B_0 ($B_0A_1 + B_0B_1$)

init(B) B_1 ($B_0A_0 + B_0^2$) = init(B) B_0 ($B_0A_1 + B_0B_1$)

$B_1A_0 + B_1B_0 = B_0A_1 + B_0B_1$

(1) ~~AA=1, AB=0 1, BA~~=0, BB=0

~~P00: init(B) B (=0~~ P001:

As seen, the proof beforehand, we were able to take cases where depending on the initial ~~value~~ probability, we derived invalid transition matrices from equating the probabilities of $p_{000} = p_{111} = 0$. Code was written to find the invalid matrices in the case where both initial probabilities are nonzero, and it accounts for all possible combinations of length 3,* since length 4, by pigeonhole, would mean either S or T had 3 zeroes (invalid) or both S & T had two zeroes and therefore only consisted of 0s and 1s (also invalid).

*of terms that must be 0 for the equation to be true

5. PHMM learning was designed to overcome the limitations of HMMs. While HMMs do not consider positional information and don't recognize insertions/deletions, which is important in bioinformatics sequence comparison, PHMMs can alleviate many of these pit-falls. Hidden states are further divided into ~~substitute~~ match, insert and delete states, meaning that states have their own emission matrix. A forward algorithm is used for scoring PHMMs, and a gap penalty exists for insertion. Gaps are normally used to account for insertions/deletions in a sequence when aligning two sequences of DNA. However, gap costs are important and nontrivial, so a good gap penalty model avoids low scores and improves the chances of finding an alignment. PHMM learning takes $O(n^t)$ runtime since each state has its own emissions matrix of dimensions $n \times t$. If brute forced, we would choose one option from each column with n choices, leading to n^t possible choices. The worst case is some multiple of n^t iterations, so PHMM learning is $O(n^t)$ runtime.