9.3 Protein Comparison in Bioinformatics

23-09-2020

0 Introduction

Sequence comparison and alignment, combined with the systematic collection and search of databases containing biomolecular sequences, both DNA and protein, has become an essential part of modern molecular biology. Molecular sequence data is important because two proteins with similar sequences often have similar functions or structures. This means that we can learn about the function of a protein in humans by studying the functions of proteins with similar sequences in simpler organisms such as yeast, fruit flies, or frogs.

In this project we will examine methods for comparison of two sequences.

We will work with two strings S and T of lengths m and n respectively, composed of characters from some finite alphabet. Write S_i for the ith character of S, and $S[i, j]$ for the substring $S_i, ..., S_j$. If $i > j$ then $S[i, j]$ is the empty string. A *prefix* of S is a substring $S[1, k]$ for some $k \leq m$ (possibly empty). Similarly, a *suffix* of S is a substring $S[k, m]$ with $k \geq 1$.

1 The Edit Distance

We refer to edits made to a string by the 4 types R,I,D,M for **R**eplacing, **I**nserting, **D**eleting and **M**atching. Optimal edit transcripts are those that minimise the number of R, I and D edits.

Definition The edit distance $d(S,T)$ is the minimal number of edits between S and T of edit types R, I and D.

Definition $D(i,j) = d(S[1,i], T[1,j])$.

1.1 Question 1

Theorem 1.1. *For all $i,j > 0$ (δ_{ab} is the Kronecker Delta function):*

$$D(i,j) = \min\{D(i,j-1)+1, D(i-1,j)+1, D(i-1,j-1)+(1-\delta_{S_iT_j})\}.$$

Proof. We begin by noting the four cases for the final edit being R,I,D or M as before, and consider each case. Initially assume $i,j > 1$. If the final edit is **R**eplace, then we note $S_i \neq T_j$ and we have so far done $D(i-1,j-1)$ edits and require one more. If the final edit is **I**nsert, then we have so far done $D(i,j-1)$ moves and require one more. If the final move is **D**elete, then we note we have so far done $D(i-1,j)$ moves and require one more. If the final move is **M**atch, then we note $S_i = T_j$ and we have so far done $D(i-1,j-1)$ edits and require no more as we do not count M edits. Colating this together (using the Kronecker Delta function to combine R and M final edit cases), we arrive at the required solution as $D(\_,\_)$ is then the minimum of these possible cases.

Now we need to now check the cases in which i or j are 1. It is clear from the definitions, $D(0,a) = a$ (via all I edits) and $D(a,0) = a$ (via all D edits) for $0 < a \leq m$ (or) n respectively and $D(0,0) = 0$ trivially. It is easy to evaluate $D(1,1) = 1 - \delta_{S_1T_1} = RHS$ (using the above results). Now for $a > 1$, $D(1,a) = a-1$ (all inserts bar one match edit) if $S_1 \in T$ or $D(1,a) = a$ (one replace and the remaining insert edits) otherwise. These again match with the RHS of the theorem. The same can be argued for $D(a,1)$ with deletions in place of the previous insertions. Together this shows the theorem to hold true for all $i,j > 0$. ■

1.2 Question 2

We can write a function (seen in program 1, page 11) that calculates the value of $D(m,n)$ using the formula of Theorem 1.1 via first calculating $D(a,b)$ for $0 \leq a \leq m$ and $0 \leq b \leq n$. The loop terminates using the fact that $D(a,0) = a$ and $D(0,b) = b$, giving $(m+1)(n+1)$ results. To ensure each value is calculated only once we store each result in a matrix as they are found and refer back to this matrix to calculate successive terms. The following shows some examples of this code in practice:

Listing 1: Test 1

```
>>>S='shesells'
>>>T='seashells'
>>>edit_dist(8,9) #D(8,9)
3
```

Listing 2: Test 2

```
>>>S='chocolate'
>>>T='plants'
>>>edit_dist(9,6) #D(9,6)
7
```

Listing 3: Test 3

```
>>>S='cat'
>>>T='glass'
>>>edit_dist(3,6) #D(3,6)
'Index out of range.'
>>>edit_dist(3,-1) #D(3,-1)
'Index out of range.'
>>>edit_dist(3,0) #D(3,0)
3
>>>edit_dist(0,3) #D(0,3)
3
```

These match manual checks for minimal distance edit solution which gave a possible edit transcript of MRRMIMMMM for Test 1, RDDDDMMIMR for Test 2 and match boundary conditions shown in Test 3. The complexity of the algorithm is proportional to $(m+1)(n+1)$, the previously noted number of iterations, multiplied by the average number of simple operations ($\approx$10) for each iteration. Together this gives the complexity of the algorithm as $\mathcal{O}(mn)$.

1.3 Question 3

A modification of the previous program allows the production of one of the possible optimal edit transcripts between S and T by storing the chosen (there may be more than one possible optimal final edit at any stage) final edit for each calculated $D(a, b)$ then, using the same logic that formed the proof for Theorem 1.1, backtracking through the matrix to read one of the optimal edit paths (program 2, page 11). To consider all possible optimal edit transcripts we could consider all possible final edits at each stage as a new path branch which you could find with a slight modification of the program, however, this will be left for now.

The following shows Program 2 run on S = Protein A (duckbill platypus myoglobin protein) and T = Protein B (yellowfin tuna myoglobin protein). The program is told to return only the first 50 of 160 entries in the edit transcript.

Listing 4: Protein A vs Protein B (edit counts)

```
>>>S=Protein_A #as string
>>>T=Protein_B #as string
>>>m=len(S)
>>>n=len(T)
>>>D_matrix = np.zeros([m+1,n+1],dtype=int)
>>>D2_matrix = np.zeros([m+1,n+1],dtype=str)
>>>edit_dist2(m,n) #forms relevant matrices and returns D(m,n)
83
>>>transcript(D2_matrix) #first 50 of 160 edits
'MRRRMRDDDDMMMRMMRMMRMRRRRMRRMMRMMMMRMMMMRDMMRMMRM'
```

2 Scoring Matrix

2.1 Question 4

Using the BLOSUM matrices of Henikoff and Henikoff referenced in the project brief to score the likely hood of specific M and R edits between amino edits, with D and I edits given score -8. We will then consider $v(S,T)$ to be the maximum scoring edit transcript, similarly to $d(S,T)$ previously. It is then clear by the same logic that formed Theorem 1.1, if $V(i,j) = v(S[1,i], T[1,j])$, then

$$V(i,j) = \max(V(i,j-1) - 8, V(i-1,j) - 8, V(i-1,j-1) + B(S_i, T_j)) \quad (1)$$

Where $B(S_i, T_j)$ is the (S_i, T_j) element of the BLOSUM matrix (new boundary conditions: $V(0,i) = V(i,0) = -i*8$). Using this, Program 3 (page 13) explores Proteins A and B again, giving:

Listing 5: Protein A vs B (score)

```
>>>edit_score_Q4(m,n) #v(S,T)
290
>>>transcript_Q4(D2_matrix) #first 50 of 154 edits
'MDDRMDDRRRMMMRMMRMMRMRRRRMRRMMRMMMMRMMMMRRRMRMMRMR'
```

3 Scoring for gaps

3.1 Question 5

We now consider the possibility of large sequences of amino acids being deleted (inserted) with at a constant score u (chosen here to be -12). Let $v_{gap}(S,T)$ be the gap-weighted score between S and T, and $V_{gap}(i,j) = v_{gap}(S[1,i], T[1,j])$. We then note

$$\begin{aligned} V_{gap}(i,j) &= \max\{E(i,j), F(i,j), G(i,j)\} \\ E(i,j) &= \max_{0 \le k \le j-1} \{V_{gap}(i,k) + u\} \\ F(i,j) &= \max_{0 \le k \le i-1} \{V_{gap}(k,j) + u\} \\ G(i,j) &= V_{gap}(i-1, j-1) + B(S_i, T_j). \end{aligned}$$

We could iterate this algorithm on an n by m grid however, it has complexity $\mathcal{O}(mn(m+n))$ which can be improved upon.

Note $E(i,j) = \max_{0\leq k\leq j-1}\{V_{gap}(i,k)+u\} = u+\max_{0\leq k\leq j-1}\{V_{gap}(i,k)\} = u+R_{max}(i)$, with $R_{max}(i)$ a new variable we keep count of as the maximum score of the row so far (and what column this occurred in). Similarly, this can be done for $F(i,j)$ with $C_{max}(j)$ maximum column score so far. These two variables need only be updated when it changes, saving the i and j extra operations at each step. Thus, by introducing and storing the new variables $R_{max}(i)$ and $C_{max}(j)$ we have the basis for an algorithm of the same complexity as Program 1,2 and 3, being $\mathcal{O}(mn)$. We now have the new boundary conditions $V_{gap}(a,0) = V_{gap}(0,b) = u$ for $0 < a \leq m$, $0 < b \leq n$ and $V_{gap}(0,0) = 0$. This new algorithm (Program 4, page 14) is tested below for Protein A and B.

Listing 6: Protein A vs B (gap-weighted score)

```
>>>D_matrix = np.zeros([m+1,n+1],dtype=int)
>>>D2_matrix = np.zeros([m+1,n+1],dtype='U4')
>>>row_max=np.full([m+1,2], -1000, dtype=int)
>>>col_max=np.full([n+1,2], -1000, dtype=int)
>>>edit_score_Q5(m,n) #v_{gap}(S,T)
305
>>>transcript_Q5(D2_matrix) #first 50 of 154 edits
'MRDDDDRRRRMMMRMMRMMRMRRRRMRRMMRMMMRMMMMRRRMRMMRMR'
```

3.2 Question 6

The following shows the gap-weighted edit score and first 50 edits of the edit transcription produced by Program 4 with S = Protein C and T = Protein D (both keratin structures in humans).

Listing 7: Protein C vs D (gap-weighted score)

```
>>>S=Protein_C
>>>T=Protein_D
>>>m=len(S)
>>>n=len(T) #then reset matricies and row/col max vectors
>>>edit_score_Q5(m,n) #v_{gap}(S,T)
1236
>>>transcript_Q5(D2_matrix) #first 50 of 468 edits
'MRIIIIIIIIIIRRMMMRRMRMRMMRRDDDDDDDDDDDDDDDDDDDDDD'
```

4 Statistical Significance

4.1 Question 7

Let U^n, V^n be independent identically distributed random vectors for proteins of length n with $\mathbf{P}(U_i^n = a) = p$ and $\mathbf{P}(U_i^n = b) = 1-p$. Use the same scoring system as section 3 with $w(l) = u$ and $s(a,a) = s(b,b) = 1, s(a,b) = s(b,a) = -1$.

Theorem 4.1. *For all $0 \leq p \leq 1$, and for all $u \leq 0$,*

$$\liminf_{n\to\infty} \frac{\mathbb{E}(v_{gap}(U^n, V^n))}{n} > 0.$$

Proof. If $u \geq 0$ it is trivial to see this result is true by simply deleting then inserting those mismatched characters and matching otherwise. Else, considering the naive edit strategy, without deletions or insertions, we have $v_{gap}(U^n, V^n) = k(1) + (n-k)(-1) + C_0 = 2k - n + C_0$ with $C_0 \geq 0$ (the possible increased score accounting for deletions/insertions) and k the number of correctly matched elements of U^n, V^n. Then by linearity of expectation, $\mathbb{E}(v_{gap}(U^n, V^n)) \geq 2\mathbb{E}(k) - n$. We see k is $\text{Bin}(n, p_1)$ with $p_1 = p^2 + (1-p)^2$. Therefore, $\mathbb{E}(v_{gap}(U^n, V^n)) \geq n(2p_1 - 1)$. Note, $2p_1 - 1 = (1-2p)^2$ therefore is 0 for $p = 1/2$ and strictly positive otherwise. Already this shows $\frac{\mathbb{E}(v_{gap}(U^n,V^n))}{n} \geq 0$. Thus, $\frac{\mathbb{E}(v_{gap}(U^n,V^n))}{n} \geq \frac{\mathbb{E}(C_0)}{n}$.

We now consider the value C_0, this represents the possible improvements to the score v_{gap} that must come about from group deletion-insertions. We will consider a subset of these possibilities where there are sub-strings (of length N) of U^n and V^n which are all mismatched. To make such an edit significant, score wise, we need the score of the deletion and insertion of N inline characters to be greater than the score for replacing N characters or $-N < 2u$. Even further, we need the expectation of this increase in score to be at least linear in n. Define $N_0 = \lfloor 2 - 2u \rfloor > 1 - 2u$ and $n = k_0 N_0 + k_1$ for some integers $k_0, k_1 \in \mathbb{N}$, with $0 \leq k_1 < N_0$. For simplicity consider the $k_0 N_0$ amino acids collected together into k_0 sections of length N_0. Each of these individually has the non-zero probability to be fully misaligned with the opposing (U^n or V^n) string of $(2p(1-p))^{N_0} = p_2^{N_0}$. Thus considering the number of sections completely misaligned as a binomial distribution, we then know the expected number of misaligned sections is $k_0 p_2^{N_0}$, then note $N_0 + 2u$ is the increase in score (over the naive strategy), given such a section exists. Therefore, summing over all valid lengths gives $\mathbb{E}(C_0) > \sum_{l=N_0}^{n} (l+2u) k_0 p_2^l > \sum_{l=N_0}^{n} (l+2u)(\frac{n}{l}) p_2^l > n \sum_{l=N_0}^{n} (p_2^l + 2u p_2^l / l)$. Then using $l \geq N_0 > 1 - 2u \Rightarrow 1 + \frac{2u}{l} > \frac{1}{l}$, so

$$\begin{aligned}
\mathbb{E}(C_0) &> n \sum_{l=N_0}^{n} \frac{p_2^l}{l} > n \int \sum_{l=N_0}^{n} p_2^{l-1} \, dp_2 = n \int \frac{p_2^{N_0-1} - p_2^n}{1 - p_2} \, dp_2 \\
&> n \int p_2^{N_0-1} - p_2^n \, dp_2 = n \left(\frac{p_2^{N_0}}{N_0} - \frac{p_2^{n+1}}{n+1} \right)
\end{aligned}$$

Thus (for $n > N_0$),

$$\begin{aligned}
\frac{\mathbb{E}(v_{gap}(U^n, V^n))}{n} &> \left(\frac{p_2^{N_0}}{N_0} - \frac{p_2^{n+1}}{n+1} \right), \\
\therefore \liminf_{n\to\infty} \frac{\mathbb{E}(v_{gap}(U^n, V^n))}{n} &\geq \frac{p_2^{N_0}}{N_0} > 0
\end{aligned}$$

■

We can extend this to an even stronger statement in the following theorem.

Theorem 4.2. $\lim_{n\to\infty} \frac{\mathbb{E}(v_{gap}(U^n,V^n))}{n}$ *exists and is strictly positive.*

Proof. Firstly

$$\begin{aligned}\mathbb{E}(v_{gap}(U^n,V^n)) &= \mathbb{E}(v_{gap}(U^{n-1},V^{n-1})) + p_1 - p_2 + \mathbb{E}C_n \\ &= \mathbb{E}(v_{gap}(U^{n-1},V^{n-1})) + (2p-1)^2 + \mathbb{E}C_n,\end{aligned}$$

where $\mathbb{E}C_n \geq 0$ is the expected score improvements by accounting for more complicated delete/insert sequences with the first $n-1$ characters of U^n, V^n interacting with the last. Thus, we see $\mathbb{E}(v_{gap}(U^n,V^n))$ is monotonically increasing. Similarly, we conclude $\mathbb{E}C_n$ must also be monotonically increasing as $\mathbb{E}C_n = \mathbb{E}C_{n-1} + \mathbb{E}D_n$ with $\mathbb{E}D_n \geq 0$ the expected score improvement by considering deleting and reinserting all n characters.

Now we note the best possible score is $v_{gap}(U^n,V^n) = n$ (if $u \leq 0$) thus, $\frac{\mathbb{E}(v_{gap}(U^n,V^n)}{n} \leq 1$. Now consider cases for $\mathbb{E}C_n$ as $n \to \infty$. If it is unbounded, then by monotonicity, $\mathbb{E}C_n \to \infty$. Then, there is an N_1 such that $n > N_1$ implies $\mathbb{E}C_n > 1$ and so $\frac{\mathbb{E}(v_{gap}(U^n,V^n)}{n}$ must be monotone increasing eventually, but we know it is bounded above so the limit $\lim_{n\to\infty} \frac{\mathbb{E}(v_{gap}(U^n,V^n)}{n}$ must converge. If $\mathbb{E}C_n$ is bounded above, then by monontonicity, $\mathbb{E}C_n \to \kappa$ as $n \to \infty$. If $\kappa > 1$ then, similarly to above, there exists an N_2 such that for $n > N_2$, $\frac{\mathbb{E}(v_{gap}(U^n,V^n)}{n}$ is monotonic increasing and so it converges. If $\kappa \leq 1$, then $\frac{\mathbb{E}(v_{gap}(U^n,V^n)}{n}$ is monotonic decreasing for all n, and so, using the lower bound for the limit as $n \to \infty$ found in the previous theorem we conclude $\frac{\mathbb{E}(v_{gap}(U^n,V^n)}{n}$ must converge as $n \to \infty$. This covers all possibilities and so the limit must exist and this must be in the interval $\left[\frac{p_2^{N_0}}{N_0}, 1\right]$ with N_0 as defined previously. ■

4.2 Question 8

In Program 5 (page 16) we estimate $n^{-1}\mathbb{E}(v_{gap}(U^n,V^n))$ for $u = -3$ and $p = 0.5$, by randomly generating protein string pairs U^n and V^n then calculating the exact score of $v_{gap}(U^n,V^n)/n$. We then average this over a large number of attempts (N) to attain an approximation of the expected value. The following shows some examples.

Listing 8: $n = 1$ over 1000 tests

```
>>> estimate(1,1000) #true value is known to be exactly 0
0.038
>>> estimate(1,1000)
-0.02
```

Listing 9: Varying increasing values of n

```
>>> estimate(10,1000)
0.10930000000000031 #results vary within ~0.02
>>> estimate(100,1000)
0.37032000000000004 #results vary within ~0.005
>>>estimate(1000,10)
0.42693999999999993 #results vary within ~0.005
>>>estimate(10000,1)
0.4388 #results vary within ~0.005
>>>estimate(15000,1)
0.44426666666666664  #results vary within ~0.005
```

Listing 8 shows the program appears to show consistency with the known result of 0. Listing 9 shows with increasingly accurate estimates for the limit of $n^{-1}\mathbb{E}(v_{gap}(U^n, V^n))$ as $n \to \infty$. Note that for higher values of n the variance of the distribution becomes negligible and so large values of N are not required (to assist with computation time). With each tenfold rise in n (starting at n=10) we see the estimate rise by an amount 1/4th of the previous increase, extrapolating this gives: $\lim_{n\to\infty} n^{-1}\mathbb{E}(v_{gap}(U^n, V^n)) \approx 0.1 + 0.27(1 + 1/4 + (1/4)^2 + ...) = 0.46$.

5 Local Alignment

In this section we consider local alignment and attempt to find the pair of substrings S' and T' of S and T with the highest alignment score namely,

$$v_{sub}(S,T) = \max\{v(S',T') : S' \text{ a substring of } S, T' \text{ a substring of } T\}.$$

We will use the scoring system of section 2 for simplicity, write $s(-,a) = s(a,-) < 0$ for the score of an insertion or deletion, and use the following notation,

$$v_{sfx}(S,T) = \max\{v(S',T') : S' \text{ a suffix of } S, T' \text{ a suffix of } T\}.$$

5.1 Question 9

Theorem 5.1. $v_{sub}(S,T) = \max\{v_{sfx}(S',T') : S' \textit{ a prefix of } S, T' \textit{ a prefix of } T\}$.

Proof.

$$\begin{aligned} RHS &= \max\{v_{sfx}(S',T') : S' \text{ a prefix of } S, T' \text{ a prefix of } T\} \\ &= \max\left\{\sum_{0\le i\le m, 0\le j\le n} v_{sfx}(S[1,i], T[1,j])\right\} \\ &= \max\left\{\sum_{0\le i\le m, 0\le j\le n} \max\left\{\sum_{1\le a\le m, 1\le b\le n} v(S[a,i], T[b,j])\right\}\right\}. \end{aligned}$$

Then as n and m are finite,

$$\begin{aligned} RHS &= \max\left\{ \sum_{0\leq i\leq m, 0\leq j\leq n, 1\leq a\leq m, 1\leq b\leq n} v(S[a,i],T[b,j]) \right\} \\ &= \max\{v(S',T') : S' \text{ a substring of } S, T' \text{ a substring of } T\} \\ &= v_{sub}(S,T) = LHS. \end{aligned}$$

Thus, $v_{sub}(S,T) = \max\{v_{sfx}(S',T') : S' \text{ a prefix of } S, T' \text{ a prefix of } T\}$. ■

5.2 Question 10

Theorem 5.2. *Define* $V_{sfx}(i,j) = v_{sfx}(S[1,i],T[1,j])$*, then*

$$V_{sfx}(i,j) = \max\begin{cases} 0, \\ V_{sfx}(i-1,j-1) + B(S_i,T_j), \\ V_{sfx}(i-1,j) + s(S_i,-), \\ V_{sfx}(i,j-1) + s(-,T_j), \end{cases}$$

with boundary conditions $V_{sfx}(i,0) = V_{sfx}(0,j) = 0$.

Proof. Similar to Theorem1.1 we consider the possible cases. We first note that the final edit does not affect the beginning of the suffix (this trivially leads to a contradiction on the previous string being optimal if assumed otherwise) unless the maximum suffix of non-zero length is now negative, in which case the maximum suffix becomes the empty string with 0 score. Otherwise, if the last edit should be a matching or replacement then it follows $V_{sfx}(i,j) = V_{sfx}(i-1,j-1)+B(S_i,T_j)$. Similarly if the last edit should be a delete then $V_{sfx}(i,j) = V_{sfx}(i-1,j)+s(S_i,-)$ and if the last edit should be an insert then $V_{sfx}(i,j) = V_{sfx}(i,j-1)+s(-,T_j)$. This covers all possible cases leaving $V_{sfx}(i,j)$ as the maximum of the four, as seen in the theorem. The boundary conditions follow naturally as the suffix must be an empty string as otherwise the optimal score would be negative with only deletions/insertions. ■

5.3 Question 11

Using the Theorems 5.1 and 5.2 we can write a program of complexity $\mathcal{O}(mn)$ to calculate the score, exact sub-strings and required edits that correspond to $v_{sub}(S,T)$. This is completed by first calculating the mn values of $V_{sfx}(i,j)$ using a similar $\mathcal{O}(mn)$ algorithm to section 2 then determining the maximum of these, which must correspond to $v_{sub}(S,T)$. Then it is only a matter of forming the edit transcript following back until the start of these sub-strings (again done previously with complexity $\mathcal{O}(mn)$). Program 6 (page 17) completes this and returns the following results:

Listing 10: Calculating v_{sub}

```
>>>v_sub() #returns a summary and the first 50 of 421 edits
v_sub = 1312, substrings: S[33,430], T[6,411]
MMMRIRMRMMDMDDDDDDMDMMDDRDMRRMIMRIRMIIMRMMRMRRMRMM
```

6 Appendices

* Note all code written in Python using packages as defined within each program.

6.1 Program 1

Consisting of two functions program 1, specifically $edit\_dist(m, n)$, is able to calculate the value of $D(i, j)$ for some pre-set strings S and T as defined with the report.

Listing 11: Program 1

```
def D(i,j,S,T,D_matrix):
    if i==0:
        answer = j
    elif j==0:
        answer = i
    else:
        S_i = S[i-1]
        T_j = T[j-1]
        if S_i==T_j:
            delta = 1
        else:
            delta = 0
        a = D_matrix[i][j-1]+1
        b = D_matrix[i-1][j]+1
        c = D_matrix[i-1][j-1]+1-delta
        answer = min(a, b, c)
    return answer

def edit_dist(m,n):
    if m > len(S) or m < 0 or n > len(T) or n < 0:
        return 'Index out of range.'
    D_matrix = np.zeros([m + 1, n + 1], dtype=int)
    b=0
    while b <= n :
        a=0
        while a <= m:
            D_matrix[a][b] = D(a,b,S,T,D_matrix)
            a=a+1
        b=b+1
    return D_matrix[m][n]
```

6.2 Program 2

Extending upon the approach of Program 1 the following three programs will update two matrices D_matrix and $D2\_matrix$ which record the scores $D(i, j)$ and possible (optimal) edits at each step as a string. The code is to be used as seen in Listing 4.

Listing 12: Program 2

```
def D2(i,j,S,T):
    if i==0 and j==0:
        D_matrix[i][j] = 0
        D2_matrix[i][j] = ''
    elif i==0 and j!=0:
        D_matrix[i][j] = j
        D2_matrix[i][j] = 'I'
    elif j==0 and i!=0:
        D_matrix[i][j] = i
        D2_matrix[i][j] = 'D'
    else:
        S_i = S[i-1]
        T_j = T[j-1]
        if S_i==T_j:
            delta = 1
        else:
            delta = 0
        a = D_matrix[i][j-1]+1
        b = D_matrix[i-1][j]+1
        c = D_matrix[i-1][j-1]+1-delta
        answer = min(a, b, c)
        next_edit = ''
        #Separate 'if' options to add all of the possible edit choices.
        if answer == a:
            next_edit = 'I'
        if answer == b:
            next_edit += 'D'
        if answer == c and delta == 1:
            next_edit += 'M'
        elif answer == c and delta == 0:
            next_edit += 'R'
        D_matrix[i][j] = answer
        D2_matrix[i][j] = next_edit
    return

def edit_dist2(m,n):
    b=0
    while b<=n:
        a=0
        while a<=m:
            D2(a,b,S,T)
            a=a+1
        b=b+1
    return D_matrix[m][n]

def transcript(D2_matrix):
    edit_transcript = ''
    i=m
    j=n
    while i!=0 or j!=0:
        #if more than one edit possible choose the first.
        edit = D2_matrix[i][j][0]
        edit_transcript += edit
        if edit == 'I':
            j=j-1
        elif edit == 'D':
            i=i-1
```

```
        else:
            i=i-1
            j=j-1
    #print(len(edit_transcript))

    #reverse order
    edit_transcript = edit_transcript[::-1]
    return edit_transcript[0:50]
```

6.3 Program 3

Using biopython's blosum62 matrix and the scoring system described within the project the following program implements a similar algorithm to calculate the maximal scores $V(i, j)$ and a possible corresponding edit transcript.

Listing 13: Program 3

```
from Bio.SubsMat import MatrixInfo as matrixFile
blosum = matrixFile.blosum62

def D_Q4(i,j,S,T):
    if i==0 and j==0:
        D_matrix[i][j] = 0
        D2_matrix[i][j] = ''
    elif i==0 and j!=0:
        D_matrix[i][j] = -j*8
        D2_matrix[i][j] = 'I'
    elif j==0 and i!=0:
        D_matrix[i][j] = -i*8
        D2_matrix[i][j] = 'D'
    else:
        S_i = S[i-1]
        T_j = T[j-1]
        if blosum.get((S_i,T_j)) == None:
            s = blosum.get((T_j,S_i))
        else:
            s = blosum.get((S_i,T_j))
        a = D_matrix[i][j-1]-8
        b = D_matrix[i-1][j]-8
        c = D_matrix[i-1][j-1] + int(s)
        answer = max(a, b, c)
        next_edit = ''
        if answer == a:
            next_edit = 'I'
        if answer == b:
            next_edit += 'D'
        if answer == c and S_i==T_j:
            next_edit += 'M'
        elif answer == c and S_i!=T_j:
            next_edit += 'R'
        D_matrix[i][j] = answer
        D2_matrix[i][j] = next_edit
    return

def edit_score_Q4(m,n):
    b=0
```

```
    while b<=n:
        a=0
        while a<=m:
            D_Q4(a,b,S,T)
            a=a+1
        b=b+1
    return D_matrix[m][n]

def transcript_Q4(D2_matrix):
    edit_transcript = ''
    i=m
    j=n
    while i!=0 or j!=0:
        edit = D2_matrix[i][j][0]
        edit_transcript += edit
        if edit == 'I':
            j=j-1
        elif edit == 'D':
            i=i-1
        else:
            i=i-1
            j=j-1
    #print(len(edit_transcript))

    edit_transcript = edit_transcript[::-1]
    return edit_transcript[0:50]
```

6.4 Program 4

This program further extends upon the previous methods now allowing for gaps of arbitrary length to be deleted or inserted. Note: you must reset the variables *row_max*, *col_max*, *D_matrix* and *D2_matrix* each time.

Listing 14: Program 4

```
row_max=np.full([m+1,2], -1000, dtype=int)
col_max=np.full([n+1,2], -1000, dtype=int)

def D_Q5(i,j,S,T):
    if i==0 and j==0:
        D_matrix[i][j] = 0
        D2_matrix[i][j] = ''
        row_max[0] = [0,0]
        col_max[0] = [0,0]
    elif i==0 and j!=0:
        D_matrix[i][j] = -12
        D2_matrix[i][j] = 'I0'
        col_max[j] = [max(col_max[j][0], -12), i]
    elif j==0 and i!=0:
        D_matrix[i][j] = -12
        D2_matrix[i][j] = 'D0'
        row_max[i] = [max(row_max[j][0], -12), j]
    else:
        S_i = S[i-1]
        T_j = T[j-1]
        if blosum.get((S_i,T_j)) == None:
```

```
            s = blosum.get((T_j,S_i))
        else:
            s = blosum.get((S_i,T_j))
        p = row_max[i][0] - 12
        q = col_max[j][0] - 12
        r = D_matrix[i-1][j-1] + int(s)
        answer = max(p, q, r)
        new_row = max(row_max[i][0],answer)
        new_col = max(col_max[j][0],answer)
        if row_max[i][0] < answer:
            row_max[i] = [new_row, j]
        if col_max[j][0] < answer:
            col_max[j] = [new_col, i]
        next_edit = ''
        if answer == p:
            next_edit = 'I' + str(row_max[i][1])
        elif answer == q:
            next_edit = 'D' + str(col_max[j][1])
        elif answer == r and S_i==T_j:
            next_edit = 'M'
        elif answer == r and S_i!=T_j:
            next_edit = 'R'
        D_matrix[i][j] = answer
        D2_matrix[i][j] = next_edit
    return

def edit_score_Q5(m,n):
    b=0
    while b<=n:
        a=0
        while a<=m:
            D_Q5(a,b,S,T)
            a=a+1
        b=b+1
    return D_matrix[m][n]

def transcript_Q5(D2_matrix):
    edit_transcript = ''
    i=m
    j=n
    while i!=0 or j!=0:
        edit = D2_matrix[i][j][0]
        if edit == 'I':
            gap = j - int(D2_matrix[i][j][1:])
            edit = edit * gap
            j=j-gap
        elif edit == 'D':
            gap = i - int(D2_matrix[i][j][1:])
            edit = edit * gap
            i=i-gap
        else:
            i=i-1
            j=j-1
        edit_transcript += edit
    #print(len(edit_transcript))
    edit_transcript = edit_transcript[::-1]
    return edit_transcript[0:50]
```

6.5 Program 5

The first two functions are essentially equivalent to those in program 4, but with unnecessary calculations removed for improved efficiency. The third function then computes an estimate for the limit $n^{-1}\mathbb{E}(v_{gap}(U^n, V^n))$ as $n \to \infty$.

Listing 15: Program 5

```
u=-3
p=0.5

def D_Q8(i,j,S,T, D_matrix, row_max, col_max):
    if i==0 and j==0:
        D_matrix[i][j] = 0
        row_max[0] = [0,0]
        col_max[0] = [0,0]
    elif i==0 and j!=0:
        D_matrix[i][j] = u
        col_max[j] = [max(col_max[j][0], u), i]
    elif j==0 and i!=0:
        D_matrix[i][j] = u
        row_max[i] = [max(row_max[j][0], u), j]
    else:
        S_i = S[i-1]
        T_j = T[j-1]
        if S_i==T_j:
            delta=1
        else:
            delta=-1
        p = row_max[i][0] + u
        q = col_max[j][0] + u
        r = D_matrix[i-1][j-1] + delta
        answer = max(p, q, r)
        new_row = max(row_max[i][0],answer)
        new_col = max(col_max[j][0],answer)
        if row_max[i][0] < answer:
            row_max[i] = [new_row, j]
        if col_max[j][0] < answer:
            col_max[j] = [new_col, i]
        D_matrix[i][j] = answer
    return

def edit_score_Q8(m,n,S,T):
    b=0
    D_matrix = np.zeros([n + 1, m + 1], dtype=int)
    row_max = np.full([m + 1, 2], -1000, dtype=int)
    col_max = np.full([n + 1, 2], -1000, dtype=int)
    while b<=m:
        a=0
        while a<=n:
            D_Q8(a,b,S,T,D_matrix, row_max, col_max)
            a=a+1
        b=b+1
    return D_matrix[m][n]

def estimate(num, tests):
    total_est = 0
```

```
    i=0
    while i < tests:
        j=0
        S=''
        T=''
        while j<num:
            #a=0,b=1
            amino_acid1 = random.randint(0, 1)
            amino_acid2 = random.randint(0, 1)
            S = S + str(amino_acid1)
            T = T + str(amino_acid2)
            j=j+1
        est0 = edit_score_Q8(num,num,S,T)
        est=est0/num
        total_est = total_est + est
        i=i+1
    avg_est = total_est / tests
    return avg_est
```

6.6 Program 6

This final program contains two functions, calling *v_sub* will use *form_matrix* to produce a summary of the most similar substrings between proteins C and D.

Listing 16: Program 6

```
def form_matrix():
    S = Protein_C
    T = Protein_D
    m = len(S)
    n = len(T)
    V_sfx_matrix = np.zeros([m + 1, n + 1], dtype=int)
    V_sfx_matrix2 = np.zeros([m + 1, n + 1], dtype='U4')
    b=0
    while b<n+1:
        a=0
        while a<m+1:
            if a == 0 or b == 0:
                V_sfx_matrix[a][b] = 0
                V_sfx_matrix2[a][b] = ''
            else:
                S_a = S[a-1]
                T_b = T[b-1]
                if blosum.get((S_a, T_b)) == None:
                    s = blosum.get((T_b, S_a))
                else:
                    s = blosum.get((S_a, T_b))
                Rep_match = V_sfx_matrix[a-1][b-1] + s
                Del = V_sfx_matrix[a-1][b]-2
                Ins = V_sfx_matrix[a][b-1]-2

                V_a_b = max(0, Rep_match, Del, Ins)
                V_sfx_matrix[a,b] = V_a_b

                if V_a_b == 0:
```

```
                    V_sfx_matrix2[a][b] = 'B'
                elif V_a_b == Rep_match and S_a == T_b:
                    V_sfx_matrix2[a][b] = 'M'
                elif V_a_b == Rep_match and S_a != T_b:
                    V_sfx_matrix2[a][b] = 'R'
                elif V_a_b == Del:
                    V_sfx_matrix2[a][b] = 'D'
                elif V_a_b == Ins:
                    V_sfx_matrix2[a][b] = 'I'
            a=a+1
        b=b+1
    return [V_sfx_matrix, V_sfx_matrix2]

def v_sub():
    matricies = form_matrix()
    max = np.max(matricies[0])
    idx = np.where(matricies[0] == max)
    start_i = idx[0][0]
    start_j = idx[1][0]

    i=start_i
    j=start_j
    edit = ''
    edit_transcript = ''
    while edit != "B":
        edit_transcript += edit
        edit = matricies[1][i][j]
        if edit == 'I':
            j=j-1
        elif edit == 'D':
            i=i-1
        elif edit == 'M' or edit == 'R':
            i=i-1
            j=j-1
    if edit_transcript[-1] == 'I':
        end_i=i
        end_j=j+1
    elif edit_transcript[-1] == 'D':
        end_i=i+1
        end_j=j
    else:
        end_i=i+1
        end_j=j+1

    output = 'v_sub = ' +str(max) + ', substrings: S[' + str(end_i) + ',' + str(start_i) + '],
    edit_transcript = edit_transcript[::-1]

    print(output)
    #print(len(edit_transcript))
    print(edit_transcript[0:50])
```