Spring 2020 CS123A Bioinformatics Project Report

Project Title: Comparison of Multiple Sequence Alignment Programs: MUSCLE and MAFFT

NAME: Justin Thai

ABSTRACT

This report presents the testing of multiple sequence alignment (MSA) programs MUSCLE and MAFFT that have been implemented using Python 3.8. To implement these MSA programs, various classes from the Biopython package were used to help take in sequences, run the specified algorithm that makes the MSA (according to the MSA programs), and print out the final alignment results. Additionally, the MUSCLE and MAFFT programs will be used to run their respective MSA algorithm and analyze the time it takes to complete the MSA. mRNA sequences of the hemoglobin subunit beta (HBB) protein taken from the NCBI Database will be used as the dataset that creates three test cases for the program implementations. These datasets will help analyze the performances of the MSA program implementations. From the tests, the MUSCLE program implementation computed the MSA faster than the MAFFT program implementation, but the MAFFT implementation was overall more efficient. When comparing the rates of increase in time between test cases, the MAFFT implementation had a slower rate of increase than the MUSCLE implementation, indicating that the MAFFT implementation was more efficient. Through these tests, the advantages and disadvantages of the MAFFT and MUSCLE programs are shown. Their use by bioinformaticians in real-world scenarios would vary for each situation. In the next steps of this project, larger datasets can be used to further analyze the performances and efficiencies of the MAFFT and MUSCLE program implementations.

Key Words: MUSCLE, MAFFT, multiple sequence alignment (MSA)

Table of Contents

LIST OF TABLES

LIST OF FIGURES

INTRODUCTION

Today, there are several different tools that bioinformaticians use to analyze the similarities and differences between several sequences, like MAFFT and MUSCLE. MAFFT (Multiple Alignment using Fast Fourier Transform) is a multiple sequence alignment (MSA) program developed in 2002 that uses fast Fourier transform to recognize similar sections between genomic sequences (Katoh et al., 2002). By using the fast Fourier transform algorithm, the MAFFT program has a proposed time complexity of $O(N^2L + NL^2)$, where the length of the sequence is represented by L and the number of sequences is represented by N (Katoh & Toh, 2008). On the other hand, the MUSCLE (Multiple Sequence Comparison by Log-Expectation) program was created in 2004 and produces MSAs through a series of stages and recursions (Edgar, 2004a). The suggested time complexity of the MUSCLE algorithm is $O(N^4 + NL^2)$ (Edgar, 2004b).

Both MUSCLE and MAFFT use progressive alignment methods to form the MSAs (Zvelebil & Baum, 2008). Although these two algorithms can successfully do MSAs, the efficiencies of them may be different. To understand the performance differences between MAFFT and MUSCLE, implementations of MAFFT and MUSCLE will be developed using Python 3.8. These implementations will then be given sequences to align and the performances of them will be analyzed. MAFFT and MUSCLE were chosen to be implemented as these two programs were not as widely used as other MSA programs like CLUSTALW.

BACKGROUND

MAFFT Program Algorithm

The MAFFT program has two different methods for MSAs: The progressive method and the iterative refinement method (Katoh et al., 2002). The progressive method relies on speed instead of accuracy to create the MSA, while the iterative refinement method is based on accuracy over speed (Katoh et al., 2002).

In the progressive method, a distance matrix is created using the number of shared 6-tuples (i.e. 20 amino acids are categorized into 6 groups) for each pair of sequences (Katoh et al., 2002). UPGMA is then applied to the distance matrix to create a tree and progressive alignment is done according to the branching order of the tree

(Katoh et al., 2002). Using the results from the progressive alignment, the guide tree is reconstructed, and another progressive alignment is done to compute a more accurate MSA (Katoh et al., 2002).

The iterative refinement method is just additional steps to the progressive method, therefore sacrificing speed for more accuracy (Katoh et al., 2002). After the procedure for the progressive method is completed, the MSA is split into two sub-alignments and realigned again (Katoh et al., 2002). This process is executed until no further improvements can be made to the MSA (Katoh et al., 2002).

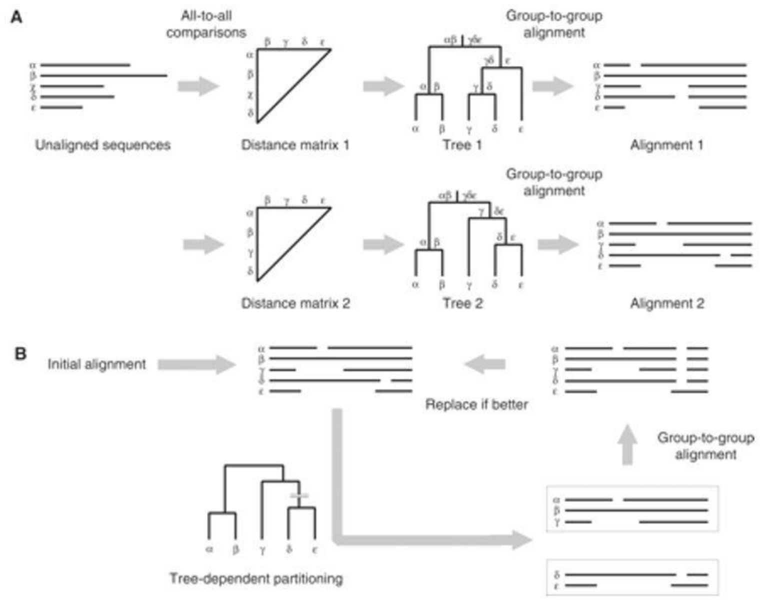


Figure 1. Diagram of the MAFFT Program Algorithm (Katoh & Toh, 2008). Diagram A represents the progressive method. Diagram B represents the iterative refinement method.

MUSCLE Program Algorithm

To compile an accurate MSA, the MUSCLE program goes through three stages in which each stage creates an MSA (Edgar, 2004a). The first stage is called Draft Progressive and the MSA made at this stage is based on speed instead of accuracy (Edgar, 2004a). To create the MSA, the program uses the unaligned sequences to calculate the k-mer distance scores and form a distance matrix from the k-mer scores (Edgar, 2004a). From this distance matrix, a phylogenetic tree called TREE1 is formed using the UPGMA algorithm (Edgar, 2004a). The first MSA (called MSA1) is compiled after a progressive alignment is done on TREE1 (Edgar, 2004a).

Improved Progressive, the second stage of the MUSCLE algorithm, is where the program focuses on creating a more accurate MSA, unlike the Draft Progressive stage (Edgar, 2004a). Using MSA1, the Kimura distance score is computed for each pair of sequences, and a distance matrix is created from the Kimura distances (Edgar, 2004a). Similar to the first stage, a tree called TREE2 is made from using UPGMA on the Kimura distance matrix (Edgar, 2004a). TREE2 then goes through progressive alignment to get an MSA called MSA2 (Edgar, 2004a).

The last stage, Refinement, produces the final MSA by iterating the steps of the stage until no further improvements could be made to the MSA or a limit to the iterations is set by the user (Edgar, 2004a). First, TREE2 is split at a random location into two subtrees and each subtree is used to make a profile of the MSA (Edgar, 2004a). These two profiles are then re-aligned to produce an MSA (Edgar, 2004a). If there are no further changes made (i.e. no further improvements can be done), the algorithm is complete (Edgar, 2004a). Otherwise, the MUSCLE program goes through the Refinement stage again (Edgar, 2004a).

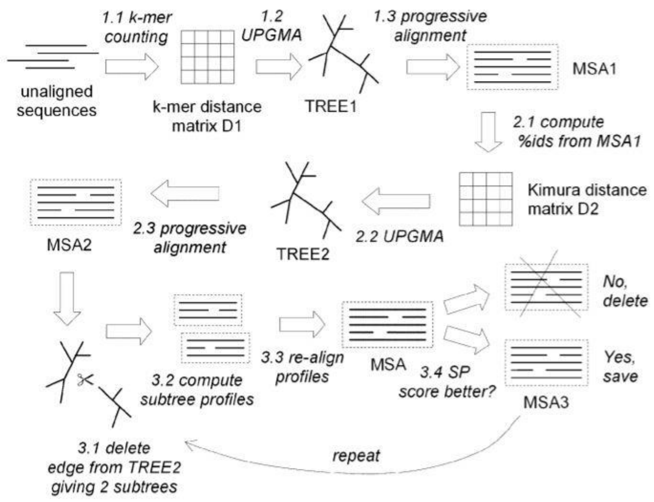

unaligned sequences
1.1 k-mer counting
k-mer distance matrix D1
1.2 UPGMA
TREE1
1.3 progressive alignment
MSA1
2.1 compute %ids from MSA1
Kimura distance matrix D2
2.2 UPGMA
TREE2
2.3 progressive alignment
MSA2
3.1 delete edge from TREE2 giving 2 subtrees
3.2 compute subtree profiles
3.3 re-align profiles
MSA
3.4 SP score better?
No, delete
Yes, save
MSA3
repeat

Figure 2. Diagram of the MUSCLE Algorithm Program Stages (Edgar, 2004a)

DATA COLLECTED / ACCESSED

To test and compare the performances of MAFFT and MUSCLE, three different cases of MSAs will be used. These cases will vary in the number of different sequences to be aligned and the relativeness of the sequences. For the sequences, this project will focus on the mRNA sequences of the hemoglobin subunit beta (HBB) protein of 18 different organisms. HBB was chosen to be used to test the MSA programs as the protein is found in many organisms, thus allowing the programs to find similarities in the HBB of different organisms. Additionally, HBB is a well-known protein that has been greatly studied by bioinformaticians. This helps ensure that the programs are using sequences that are supported by evidence. All of the sequence data used in this project is obtained from the NCBI Database and can be found in Table 1.

The first test case will consist of three mRNA sequences, two of which are closely related while the other is distantly related. For the second test case, nine sequences will be aligned in which five are from the same taxonomic order and the other four are from different orders. In the last test case, 18 sequences will be used in the MSA programs, eight of which are from the same taxonomic order. Having each test case vary in relatedness of the sequences will ensure that the programs have a variety of sequences to compare against each other.

Table 1. Table of mRNA Sequences Used. Sequence #1-3 are used in Test Case 1. Sequence #1-9 are used in Test Case 2. Sequence #1-18 are used in Test Case 3.

#	Accession Number	Organism	Seqeuence Length
1	NM_000518	*Homo sapiens* (human)	628 bp
2	XM_508242	*Pan troglodytes* (chimpanzee)	748 bp
3	NM_001164018	*Equus caballus* (horse)	444 bp
4	XM_002822127	*Pongo abelii* (Sumatran orangutan)	627 bp
5	XM_032166808	*Hylobates moloch* (silvery gibbon)	753 bp
6	XM_004090649	*Nomascus leucogenys* (northern white-cheeked gibbon)	753 bp
7	NM_001304885	*Ailuropoda melanoleuca* (giant panda)	644 bp
8	XM_008709612	*Ursus maritimus* (polar bear)	666 bp
9	NM_001279263	*Condylura cristata* (star-nosed mole)	626 bp
10	NM_001168847	*Papio anubis* (olive baboon)	444 bp
11	NM_001283367	*Macaca fascicularis* (crab-eating macaque)	604 bp
12	XM_032240524	*Sapajus apella* (Tufted capuchin)	678 bp
13	NM_033234	*Rattus norvegicus* (Norway rat)	620 bp
14	XM_005380076	*Chinchilla lanigera* (long-tailed chinchilla)	648 bp
15	NM_001097648	*Ovis aries* (sheep)	438 bp
16	NM_173917	*Bos taurus (cattle)*	633 bp
17	NM_001144841	*Sus scrofa* (pig)	496 bp
18	NM_001201019	*Ictalurus punctatus* (channel catfish)	607 bp

APPROACH AND METHOD

MUSCLE Program Implementation

The main goal of this implementation is to take any user-provided sequences in FASTA format, align the sequences using the MUSCLE algorithm, and print out the MSA result and time taken to create the MSA (see Appendix A). To implement the MUSCLE program in Python, several different tools and classes are needed. The MUSCLE program itself is necessary to help run the algorithm proposed by Edgar (2004a) on the input sequences and can be retrieved from https://drive5.com/muscle/. Also, Biopython is a necessary package that needs to be installed as it includes many necessary classes that help with formatting the MSA in the output of the program. Additionally, to test the efficiency of the MUSCLE algorithm in computing MSAs, a timing tool is needed. The time class in Python can provide the tools necessary to allow the program to time the computation of the MSA.

In the implementation program, the user will first be asked to provide the directory in which the FASTA file with the desired sequences is found. After the directory of the FASTA file is given, the MSA process and the timing of it can begin. Using the time method from the time class, a timer is started to track the time of the MSA procedure. To run the MUSCLE program and print the results of the MSA, the MUSCLE_seq_align method is called with the FASTA file as the parameter (see Appendix B).

Within the MUSCLE_seq_align method, the MuscleCommandline class from the Biopython will be used as a command-line wrapper for the MUSCLE program (which is called “muscle_exe” in the code). This gives the code the ability to pass on the FASTA file to the MUSCLE program to do the MSA. After the alignment is done, the results will be written to the standard output (stdout). Using the AlignIO class from Biopython and StringIO class, the alignment results in the standard output can be converted into the desired format to be printed (this converted result is called “align” in the code). This align variable will then be printed (see Appendix D).

After the printing of the MSA, the time method will then be used again to compute the total time taken for the MUSCLE program to complete the MSA (see Appendix C). The total time to run the program will then be printed as well.

MAFFT Program Implementation

The Python implementation of the MAFFT program aims to accomplish the same goals (see Appendix A) as the MUSCLE program implementation and uses similar tools and classes. Although the approach is similar, the MAFFT program is needed to run the MAFFT algorithm and can be retrieved from https://mafft.cbrc.jp/alignment/software/. With this, the method of incorporating the MAFFT program into the code to compute the MSA is different from how the MUSCLE program was incorporated as it is a more complicated procedure.

In the MUSCLE implementation, only the file name (.exe file) of the actual MUSCLE program needs to be stated (e.g. muscle_exe variable is initialized by the file name) to use the command-line wrapper. With the MAFFT implementation, the directory of where the MAFFT program file (.bat file) is found needs to be stated. This may cause some trouble for other users as the directory of the program file is different for each user.

There are two ways in which this problem can be solved. The default method of the implementation program will first ask the user to provide the directory of the MAFFT program file. Providing the directory will allow the program to call the MAFFT program file and compute the MSA. This solution may be tedious if the implementation program needs to be used several times. With the alternative method, the user can go into the implementation program and initialize a variable (called "mafft_exe") as the directory. Doing this procedure will allow the user to not provide the directory every time the implementation program is used.

The MAFFT_seq_align method works the same way as MUSCLE_seq_align but instead of using the MuscleCommandline class, the MAFFT version will use the MafftCommandline class from Biopython as the wrapper for the MAFFT program. As there are two methods of adding the MSA program directory to the implementation, there are two versions of the MAFFT_seq_align method. One version will have two parameters, one for the directory of the MAFFT program file and the other for the FASTA file containing the sequences. The other version will just have one parameter for just the FASTA file.

Like the MUSCLE implementation, the MAFFT program implementation will ask the user to provide the FASTA file with the sequences, start a timer, run the MAFFT program to align the sequences, and finally print the alignment results and time taken to complete the alignment (see Appendix D).

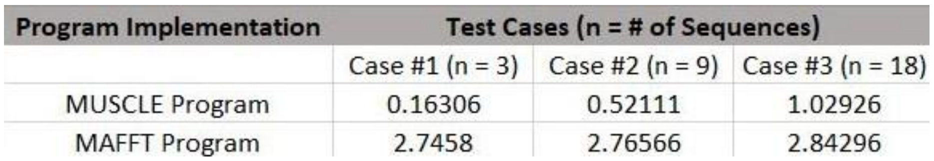
EVALUATION OF RESULTS

The results of testing the MSA program implementations using the different test cases were surprising. As expected, the time it takes for the programs to complete the MSA increases as the number of unaligned sequences increases, but the rate at which the time increases were different. For the MUSCLE program, the rate of increase in time was gradual, but this is anticipated as the test cases didn’t use a large number of sequences (i.e. 100+ sequences). This is different for the MAFFT program as its rate of increase was much more gradual in comparison to the MUSCLE program.

The total time taken for each program implementation was unexpected as there was a large gap between the two MSA programs. With the MUSCLE implementation, all the test cases took about one second or less to align and print the MSA results. When these test cases were used on the MAFFT implementation, the program took at least 2.5 seconds to do the same alignment process for all cases. According to the program website, the MAFFT program may take some time before the algorithm is started if there is an anti-virus software (Katoh, 2013). This explains the longer computation times that the MAFFT program implementation had.

Even though the MAFFT program took longer to compute the MSA than the MUSCLE program, the MAFFT program is more efficient in creating the alignments according to the results. As the rate of increase in time between the test cases was less for the MAFFT program than the MUSCLE program, the MAFFT program can handle larger numbers of sequences more efficiently than the MUSCLE program as a result. These results show that in a scenario where the MSA programs need to align several hundred sequences, the MAFFT program should perform better than the MUSCLE program.

Table 2. Table of the MSA Program Implementation Test Results. The time for each of the test cases is measured in seconds.



Program Implementation	Test Cases (n = # of Sequences)		
	Case #1 (n = 3)	Case #2 (n = 9)	Case #3 (n = 18)
MUSCLE Program	0.16306	0.52111	1.02926
MAFFT Program	2.7458	2.76566	2.84296

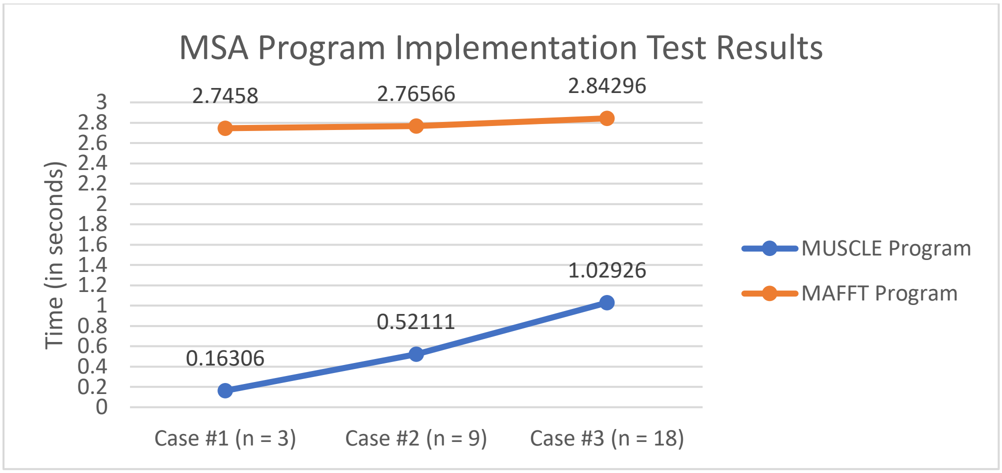


Figure 3. Graph of the Results from the MSA Program Implementations

CONCLUSION AND DISCUSSION

Although this project was done on a small scale, the results of the tests do reflect the stated efficiencies of the MSA programs. It was expected for the MAFFT program to compute the MSAs more efficiently than the MUSCLE program as the time complexity of the MAFFT program is better than MUSCLE's time complexity (Edgar, 2004b & Katoh & Toh, (2008)). From the tests, the MAFFT implementation followed its expectations and was significantly faster at completing the MSA than MUSCLE implementation (aside from the MAFFT program time delay).

As there are several tools that bioinformaticians can use to help them compute MSAs, the MSA program able to accurately align sequences and do so in the least amount of time would be the most desired program. The MAFFT and MUSCLE programs are two of these tools that have their advantages and disadvantages. The MUSCLE program is great for cases where only a small number of sequences need to be aligned but will perform poorly as more sequences are aligned. This is different for the MAFFT program as it handles the alignment of large numbers of sequences well but is poor with small amounts of sequences. Both programs are great for computing MSAs, but using them to their best potential depends on the situations that bioinformaticians are placed in.

FUTURE WORK

To add on to this project, the MUSCLE and MAFFT programs can be given more datasets to deal with so that their performances can be further analyzed. As the datasets in this project only had a small number of sequences, the MSA programs did not have too much trouble with aligning and printing the MSAs. By creating datasets that contain a larger number of sequences (similar to real-world scenarios), the MSA programs' handling of these large datasets can be observed. This will show the efficiency of the MSA programs as the number of sequences greatly increase with these datasets.

The MAFFT program implementation can be further improved to work more efficiently. One potential change is to find a better solution for handling the directory of the MAFFT program. As the current version of the implementation could create some inconvenience when calling the MAFFT program file, a more stable method of resolving this inconvenience is desired. Additionally, the timer in the program implementation can be improved so that it starts timing once the MAFFT program algorithm starts instead of starting at the preparations for the MAFFT program.

REFERENCES

Edgar, R.C. (2004a). MUSCLE: Multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research, 32(5)*, 1792-1797. https://doi.org/10.1093/nar/gkh340

Edgar, R.C. (2004b). MUSCLE: A multiple sequence alignment method with reduced time and space complexity. *BMC Bioinformatics, 5*. https://doi.org/10.1186/1471-2105-5-113

Katoh, K., Misawa, K., Kuma, K., & Miyata, T. (2002). MAFFT: A novel method for rapid multiple sequence alignment based on fast Fourier transform. *Nucleic Acids Research, 30(14)*, 3059-3066. https://doi.org/10.1093/nar/gkf436

Katoh, K., & Toh, H. (2008). Recent developments in the MAFFT multiple sequence alignment program. *Briefings in Bioinformatics, 9(4)*, 286-298. https://doi.org/10.1093/bib/bbn013

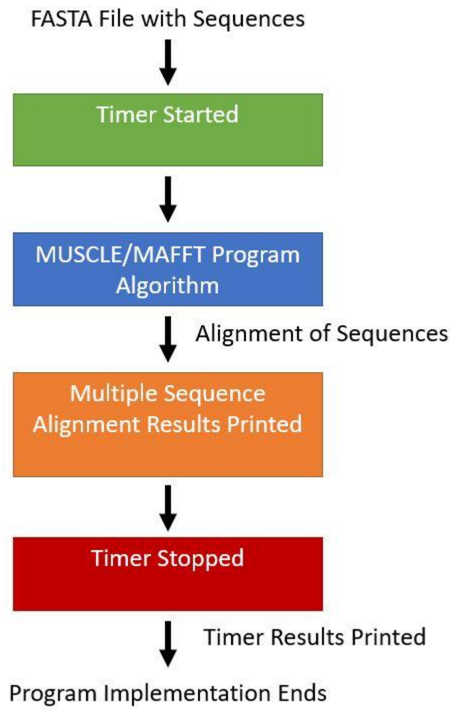
Katoh, K. (2013). *MAFFT version 7*. Kazutaka Katoh https://mafft.cbrc.jp/alignment/software/

Zvelebil, M., & Baum, J.O. (2008). *Understanding bioinformatics*. Garland Science.

APPENDICES

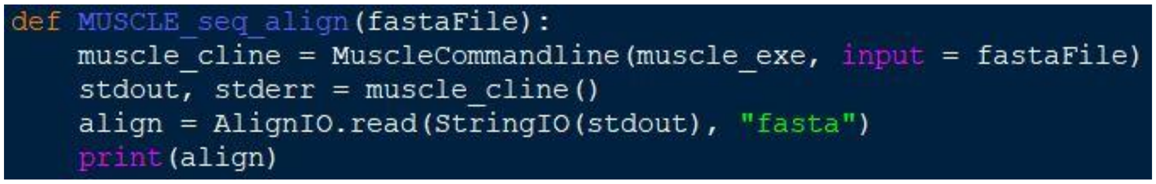
Appendix A: Diagram of the MSA Program Implementation Steps

This diagram shows the process of both the MUSCLE and MAFFT program implementations. The process starts with a FASTA file input and ends with printing out the total time taken to align and print the MSA.


FASTA File with Sequences
Timer Started
MUSCLE/MAFFT Program Algorithm
Alignment of Sequences
Multiple Sequence Alignment Results Printed
Timer Stopped
Timer Results Printed
Program Implementation Ends

Appendix B: MUSCLE_seq_align Method

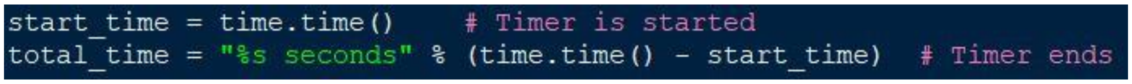
The format of the MAFFT sequence alignment methods (MAFFT_seq_align and MAFFT_seq_align2) is similar to the format of the MUSCLE_seq_align method.



```
def MUSCLE_seq_align(fastaFile):
    muscle_cline = MuscleCommandline(muscle_exe, input = fastaFile)
    stdout, stderr = muscle_cline()
    align = AlignIO.read(StringIO(stdout), "fasta")
    print(align)
```

Appendix C: How to Time the MSA Program Implementations

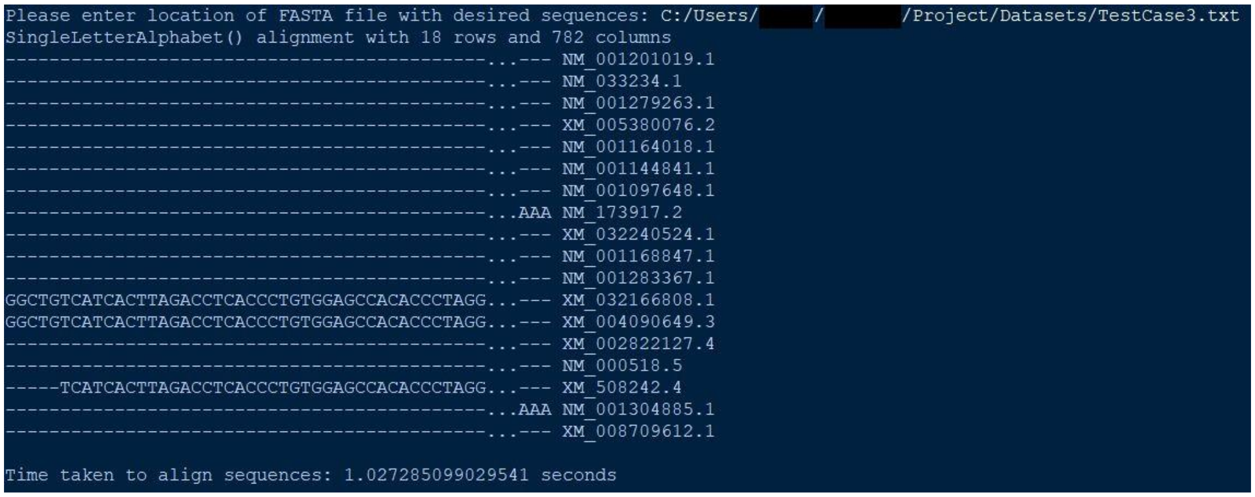
To get the total time taken for the MSA program implementations, the time method is called to start the timer (called “start_time”). Once the program is finished, the time method is called again, and this given time is subtracted by start_time to get the total time taken to align and print the MSA.



```
start_time = time.time()    # Timer is started
total_time = "%s seconds" % (time.time() - start_time)   # Timer ends
```

Appendix D: MSA Program Implementations Sample Output

This is a sample output of the MSA program implementations that uses a FASTA file with 18 different mRNA sequences.



```
Please enter location of FASTA file with desired sequences: C:/Users/[redacted]/[redacted]/Project/Datasets/TestCase3.txt
SingleLetterAlphabet() alignment with 18 rows and 782 columns
--------------------------------------------...--- NM_001201019.1
--------------------------------------------...--- NM_033234.1
--------------------------------------------...--- NM_001279263.1
--------------------------------------------...--- XM_005380076.2
--------------------------------------------...--- NM_001164018.1
--------------------------------------------...--- NM_001144841.1
--------------------------------------------...--- NM_001097648.1
--------------------------------------------...AAA NM_173917.2
--------------------------------------------...--- XM_032240524.1
--------------------------------------------...--- NM_001168847.1
--------------------------------------------...--- NM_001283367.1
GGCTGTCATCACTTAGACCTCACCCTGTGGAGCCACACCCTAGG...--- XM_032166808.1
GGCTGTCATCACTTAGACCTCACCCTGTGGAGCCACACCCTAGG...--- XM_004090649.3
--------------------------------------------...--- XM_002822127.4
--------------------------------------------...--- NM_000518.5
-----TCATCACTTAGACCTCACCCTGTGGAGCCACACCCTAGG...--- XM_508242.4
--------------------------------------------...AAA NM_001304885.1
--------------------------------------------...--- XM_008709612.1

Time taken to align sequences: 1.027285099029541 seconds
```