Species Identification through DNA String Analysis

Submitted in partial fulfilment
of the requirements of the degree of

Bachelor of Science (Honours)

of Rhodes University

Mark Vorster

Grahamstown, South Africa

November 2012

Abstract

The Rhodes University Department of Biochemistry, Microbiology and Biotechnology has found need to identify the distinct species of bacteria given a large set of DNA sequences. However, without existing tools to solve this specific problem, they have found it takes an inordinate amount of time. Based on assumptions that they are able to make given, their specific problem, an approximate string matching algorithm can be applied to speed up this process and aid them in their research. By implementing this algorithm we will show that significant speedup was indeed attained.

Acknowledgements

There are a number of people I would like to thank for continued support throughout this year and in life in general. Firstly my parents who have supported me for nearly 25 years and who have given me all that I have needed to prepare myself for my future. Secondly, my fiancée, who has made my life complete and is always full of faith and motivation. Finally I would like to thank the Computer Science Department, in particular my supervisor Phillip Machanick for all his wisdom and guidance.

In addition I would like to acknowledge the financial and technical support of Telkom, Tellabs, Stortech, Genband, Easttel, Bright Ideas 39 and THRIP through the Telkom Centre of Excellence in the Department of Computer Science at Rhodes University.

ACM Computing Classification System Classification

Thesis classification under the ACM Computing Classification System (1998 version, valid through 2012)

J.3	[*Life and Medical Sciences*]:	Biology and Genetics
I.5.4	[*Applications*]:	Text Processing

General Terms: Bioinformatics, Genetic Sequence Analysis, Sequence Alignment, Phylogenetic Analysis, Approximate String Matching

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem Statement and Research Goals

The Rhodes University Department of Biochemistry, Microbiology & Biotechnology are undergoing research in the area of species identification which requires a large amount of computer processing. There is no existing tool that fulfills research requirements and they have reported that, using the tools they do have, it can take as long as ten days to process a data set. Reasons for this include the fact that they have to undergo an entire global sequence alignment and then count the differences for each sequence. The bioinformaticians have, however, identified assumptions and areas that can drastically reduce the processing time for a sample.

This projects aim is to understand how to best to utilise these assumptions and apply string matching theory to bioinformatic sequence analysis in order to aid the bioinformaticians in their research. In particular the objective is to create a tool for their specific problem that is able to process the large datasets in a timely manner.

1.2 Background Discussion

The Rhodes University Department of Biochemistry, Microbiology & Biotechnology have taken large aquatic samples of bacterial DNA on which they wish to do a phylogenetic analysis. They have found the gene samples have specific areas of highly conserved and other highly variable sections. By focusing on these sections it should be possible to greatly reduce the amount of time to process the samples. The samples have therefore been processed accordingly, which simplifies the problem as the sequences are assumed to be aligned to the same location. They require the sequences to be grouped with other sequences of no more than approximately three percent difference, although this difference is variable. The proposed method is to calculate the differences with the approximate string matching algorithm. There are two specific areas that can be made more efficient on this algorithm due to the assumptions mentioned above. Firstly, as similarities within a small percentage difference are being searched for, branches can be pruned from the search tree once the difference exceeds the threshold. Secondly, as the sequences are assumed to begin at the same location, and thus has no need for global alignment, this reduces the complexity saving processing time.

1.3 Structure of Thesis

This thesis is organised into the following chapters:

Chapter 2 is a discussion of the background literature in the fields of bioinformatics and approximate string matching.

Chapter 3 specifies the design and implementation details of creating the system.

Chapter 4 describes the results obtained from the system, both the output and performance are examined.

Chapter 5 summarises the aims and findings of this research and explores a few extensions.

Chapter 2

Background Literature

Since DNA sequences were stored in the first bioinformatics databases, computers have been aiding and speeding up the process of analysis for bioinformaticians. The Rhodes University Department of Biochemistry, Microbiology & Biotechnology has found need to identify the distinct species of bacteria given a large set of DNA sequences. However, they have identified a means to decrease the time taken in the processing of bacterial DNA for phylogenetic analysis. While the algorithms work well for small numbers of samples, increasing the number of samples into the tens of thousands has started to take an inordinate amount of time. After examining the broad area of bioinformatics focusing on framing the problem, this paper looks at how progression in the fundamentals of genetics along with the rise of computers began bioinformatics. It then focuses more specifically on sequence analysis including mentioning the problems of sequence alignment and phylogenetic analysis, then continuing with a discussion of the approximate string matching algorithm applied.

2.1 Bioinformatics

The Biomedical Information Science and Technology Initiative's Definition Committee which was chaired by Dr Huerta defined 'Bioinformatics' in the year 2000 as

> Research, development, or application of computational tools and approaches for expanding the use of biological, medical, behavioural or health data, including those to acquire, store, organize, archive, analyse, or visualise such data. [8]

A more concise definition is found in the Oxford English Dictionary:

> The branch of science concerned with information and information flow in biological systems, esp. the use of computational methods in genetics and genomics. [1]

While Biology, the study of life, has fascinated scientists and philosophers for hundreds of years, originating as the passing on of knowledge of plants and animals for survival purposes, it progressed into medicine, agriculture, botany and zoology to name a few. At this stage the classification of living organisms was done in a qualitative manner through observation. As an example from an observation that animals look the same it can be deduced that they are of the same species, or the grouping of all animals with mammary glands being called mammals. The development of the microscope allowed biologists to move beyond the reach of the naked eye, and the study of cells became paramount to all facets of biology. [9]

Further technological advances in the 19th century in genetics and cellular biology and 20th century advances in molecular biology as explained by Xu et al. [16] have made it possible to observe the fundamental units of biology and genetics, RNA and DNA. Being able to quantify this hereditary information allows the use of computers to aid in the analysis of the data.

2.1.1 Genetics

Biology is the science of life and genetics is the study of traits being passed through genes, the fundamental units of genetics. All life goes through a reproductive cycle and the DNA chain molecules hold the inherited information that passes from generation to generation. The macromolecules form a ladder-like helix from two strands, the runs of which are formed by pairs of a nitrogenous bases either adenine and thymine or guanine with cytosine.[1] Baldi and Brunak [3] name a fundamental feature of these chain molecules as their ability to be represented digitally. The representation most commonly used represents the bases with four letters: A, T, G and C [14, p. 18]. Reviewing this information from the bottom up, the bases together form genes, of which there are multiple in each chromosome, and which carry the information required for cell growth, division and function.

2.1.2 History of Bioinformatics

Now that some basic concepts have been discussed the history of where and when bioinformatics started can be examined. While the term 'bioinformatics' only came about in the 1990s, work which would now be classified under it began many years before. Watson, Crick, Wilkins and Franklin first discovered the structure of DNA in 1953 [6] [7, p. 163] and they quickly understood how it allowed a copying mechanism for the hereditary information. The structure of DNA enables an abstraction of the molecule down to just four base units, which allows for an easy digital representation, and this is where bioinformatics begins. The storing of this information in databases was the first problem posed to bioinformaticians, largely due to the sheer volume of data, [11, p. 7] the number of base pairs in a DNA molecule ranges from thousands to hundreds of thousands, but this, as pointed out by Tramontano, [14, p. 6] would hardly be useful without annotations of the knowledge of each sequence alongside it.

[1]There is an additional base, 'uracil', that forms part of RNA sequences, but for simplicity it is not in the scope of this project.

2.2 Sequence Analysis

The analytical study of DNA, RNA or any amino acid sequence is included under the umbrella of sequence analysis. Won, Park, Yoon and Kim identify the reason for DNA sequence analysis as being that often knowledge can be inferred about newly sequenced samples through knowledge of functions of other sequences [15]. Tramontano too describes how inference can be made from similar sequences to homology, as homologous sequences are expected to have higher similarity than unrelated sequences [14, p. 77]. Sequence analysis is more general than this. Gibas and Jambeck list the 5 main types of sequence analysis as :

- Knowledge-based single sequence analysis for sequence characteristics.
- Pairwise sequence comparison and sequence-based searching.
- Multiple sequence alignment.
- Sequence motif discovery in multiple alignments.
- Phylogenetic inference.

[7, pp. 159,160]

2.2.1 Sequence Alignment

One of the major issues in sequence analysis is the alignment of sequences, and while for the purpose of this project the sequences will be assumed to be globally aligned, it is still important to understand the concept. Brudno et al. recognise sequence alignment among the most successful applications of Computer Science in Bioinformatics [4]. Before any meaningful analysis into the genetic relationship between genes can take place, sequences must be aligned [13, p. 771]. The process of alignment is often simply matching the positions of sequences to their least different (or most similar) locations. This is particularly important with partial samples where one sample is being aligned within another with unknown starting locations. This can be very computationally expensive, as not only does each potential starting location need to be examined, but due to the nature of genes there is inherent fuzziness, bases can be inserted or deleted in mutations, so at each point there may be one or more bases extra or missing from either sequence. [14, p. 55] [12]

2.2.2 Phylogenetic Analysis and Species Identification

Phylogenetics is a branch of taxonomy, the study of homogeneity of organisms including determining their evolutionary relationships, for example their species, that deals with numerical data such as DNA sequences. One of the main applications of phylogenetics is the construction of phylogenetic trees through studying similarities between organisms [10]. These phylogenetic trees are a visualisation of sequences' genetic relationships. The steps in building a phylogenetic tree from a set of DNA sequences is first to determine the similarity of each of the sequences, then starting with the most similar pair, draw the branch of the tree and calculate the average distance between that branch and each of the remaining sequences [14, pp. 66 - 73]. This research focuses on the first step, the building of the similarity table, which, although trivial for small numbers of sequences, becomes more important when dealing with thousands due to its exponential nature. This can make computation very slow.

2.2.3 FASTA format

The FASTA format is the data format chosen by the Rhodes University Department of Biochemistry, Microbiology & Biotechnology to represent DNA sequence data, and it is one of the most commonly used and simple formats [7, p.180]. A file in the FASTA format can represent many DNA sequences,where each has a single header line followed by multiple lines of sequence data. The header line requires a leading 'greater than' symbol '>' and a single word which is the name of the sequence, the rest of the line is a comment or description of the sequence. The data can contain multiple lines, new line characters are ignored, and whitespace, periods or underscores can have application-specific meaning. Below is an example of a single sequence in the FASTA format.

```
>SequenceName description of the sequence
CCGGAATACCTAGGACATAGCAGAGGCGTCTTGCCTATACAG
TGTTTTTCTCCGAGACGCCTGATTACCTGCTAGTCGGGATGA
TAACCAAGAATTTGTGTCTGCTGCGCGCCATTTGCCAACCGA
GCCTTCATCCCCCGCCGGTCTGTGATGTCCCAATGGACCGGA
```

2.3 String Matching

String matching in computer science is applied to many fields, largely in text analysis, but also in speech or character recognition and many others, as patterns can be similarly found in the binary data. Approximate string matching algorithms are used when an exact copy is expected, such as with determining the similarity of sequences. Baase [2, pp. 504 - 508] discusses an algorithm for building a difference table between two strings taking into account insertions and deletions as they may occur in DNA sequences. The approach uses a dynamic programing technique to speed up the process of building a two dimensional matrix where each point D[i][j] has the minimum number of differences between two string segments P and T, each ending at p_i and t_j respectively. The matrix is built up column by column where D[i][j] is calculated as the minimum of three possible numbers either a 'matchCost' (if $p_i = t_j$) or a 'reviseCost'(if $p_i \neq t_j$), a 'insertCost' and a 'deleteCost' where each is defined as the following:

$$\textbf{matchCost} = D[i-1][j-1] \qquad , \text{if } p_i = t_j \text{ or}$$

$$\textbf{reviseCost} = D[i-1][j-1] + 1 \qquad , \text{if } p_i \neq t_j$$

$$\textbf{insertCost} = D[i-1][j] + 1$$

$$\textbf{deleteCost} = D[i][j-1] + 1$$

2.4 Problem Discussion

Firstly, as mentioned previously, we are searching for similarities within a small percentage difference, branches can be pruned from the search tree once D[i][j] exceeds the threshold, as it is the minimum difference at that point in the matrix no further processing is needed. Secondly, as the sequences are assumed to begin at the same location, the first row can include a dependency on the previous column's first row such that D[0][j] = D[0][j-1] if $p_i = t_j$ or D[0][j-1]+1 if $p_i \neq t_j$, which will reduce the possible 'starting columns' down to only one percentage of the shortest gene rather than its entire length, saving processing but more significantly space. Another method to make this more efficient could be explored by noting that given two similar sequences a third sequence significantly different from the first need not be compared at all with the second.

Chapter 3

Design and Implementation

3.1 Analysis/Requirements

The development requirements that were outlined by the biology department are as follows. They require a system that takes its input files in the FASTA file format and compares all the sequences found therein to each other following this it groups them into species based on a given difference threshold. The system needs to run in a timely manner on up to about 10000 sequences of 400 to 800 bases each.

3.1.1 Assumptions

Pre-aligned Data

The information supplied by the biology department was that the current system that they use takes ten days to process a sample. The bioinformaticians, however, have made an assumption based on the fact that the sequences have been pre-processed. The sequences are not entire genomes but sections that have been targeted for this processing, thus they have already been globally aligned. This assumption is what allows the use of the approximate string matching algorithm as opposed to time consuming global alignment algorithms. In addition, while the local alignment needs to be taken into account, aligned sequences are not the result of this processing, this allows advantage to be taken of branch pruning.

3.1.2 Limitations

Single Threaded

While it would appear that this algorithm could achieve a large speed-up by taking advantage of parallelisation, the computer available to the bioinformaticians for this research has eight cores and as they are collecting many sets of samples at a time they have expressed that they would like each to run on a single thread. This would allow them to set up the processing of eight difference sample sets concurrently, as opposed to doing each sample set as quickly as possible if it could use all the resources. Further discussion on parallelisation is included in the future extension section of this paper.

Working Directly with DNA Sequences

When working with DNA it is often possible to convert from DNA sequences to the protein sequences [11]. This is due to the fact that while there are only four bases in DNA sequences, protein sequences have 20. The resulting protein sequences are, as a result, shorter and more unique, which not only allows for faster processing but can also be more accurate. While this is the preferred way of analysing sequences, this approach is not always applicable, and as a result the bioinformaticians have made a decision to use DNA directly in their research .

Gene Similarity vs Homogeneity

Yet another limitation is that similarity does not imply homology. Homology in DNA refers to the traits that are inherited from ancestors and therefore, given two homologous sequences, it follows that organisms from which the samples came from are of the same species. However the lines around bacterial and microbial species are blurred [5] and as such the bioinfomaticians have decided to correlate similarity and homogeneity for the purpose of their research.

3.2 Implementation

The algorithm used in this work has been adapted from the approximate string matching algorithm of Sara Baase[2, pp. 504 - 508]. Her algorithm uses a dynamic programing approach, building a difference table in an attempt to find the first instance of a k-th approximate match between a pattern and the text. The mismatches found in the algorithm are described in a similar way to the mutations found in genetic material. They include revise, delete and insert, where revise is where corresponding characters in the pattern and the text are different; delete is the scenario where the text contains a character missing from the pattern; and insert is the converse where the text is missing a character present in the pattern.

3.2.1 Language Choice

As two important considerations are performance and portability C has been chosen as the development language. In addition to the powerful performance C has the advantage of low level flexibility.

3.3 Changes to Baase's algorithm

The main changes to Baase's algorithm deal with the differences between finding a pattern in a text and counting the differences between two sequences.

3.3.1 Value Initialisations

Baase's algorithm, which finds a pattern within a text, can have points of interest starting at any point throughout the text. For this it assumes there is a null string which contains zero matches and thus initialises the first row to zeros. Another approach to this is that down the first column a null string will have exactly i differences from the pattern at that point in order to include the possibility of a partial match on the tail end of the pattern with the initial letters of the text. As this research assumes that the sequences are pre-aligned at the beginning of the sequence, the null string comparison is unnecessary. In addition, the fact that we are not searching for a pattern in a string but comparing the two sequences means that the initial rows and columns will emulate the initial row in the original algorithm. Instead of the first row being initialised to zeros and the first column to its index, there are two cases, in the case where the character at the current index matches the corresponding sequence's initial character, where all previous are treated as insert or delete mismatches, or in the case of a mismatch between these two characters, index plus one, where there are index number of inserts or deletes plus a revision at the first character.

```
Relevant code:
D: the two dimensional difference table
minLen: the length of the smaller of the two sequences.
seq0: char array containing the first sequence
seq1: char array containing the other sequence

for(i = 0; i < minLen; i++) {//initial data
        D[0][i] = i+(seq0[i]!=seq1[0]?1:0);
        D[i][0] = i+(seq0[0]!=seq1[i]?1:0);
}
```

3.3.2 Stop Case and Branch Pruning

Another change to Baase's algorithm is the stop case. Because there is no longer a pattern being matched to a text but rather two sequences being compared, both the last row and last column need to be taken into account for stop cases. As the calculation happens row by row we should just assume the similar case for the approximate string matching algorithm. The difference comes in the fact that if any value in the last row is below the threshold, it, too, counts as a match within the threshold. In addition, a branch pruning heuristic can be implemented as if an entire row's results are above the threshold as the numbers found in the table can never fall again later in the table. This can be seen in figure 3.2. If the case was that the threshold limited only single difference then we could conclude that no match has been obtained. This is only possible in this implementation due to the changes to the value initialisation based on the assumption that the sequences are aligned to the same place. A similar heuristic would be possible with the original algorithm but it is of use as each column's initial value is zero.

3.4 Sequence by Sequence

The general case for building the difference table is implemented the same way that Baase describes. There are clear similarities when understanding the mechanics of the both the natural mutations of genes and the approximate string matching algorithm. The algorithm's costs (MatchCost, ReviseCost, InsertCost, DeleteCost) can be mapped to the four alignment options for sequences (Match, Replacement, Insert, Delete). Figure 3.1 shows an example of the approximate string matching algorithm in use, halfway through building up the difference table.

By comparing this with figure 3.2 the changes mentioned can be observed.

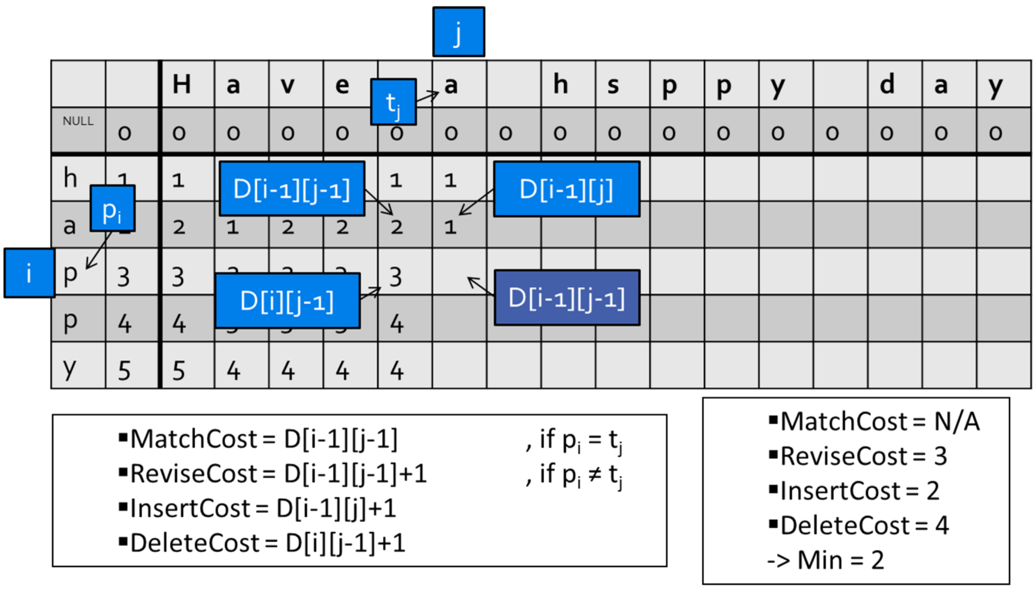
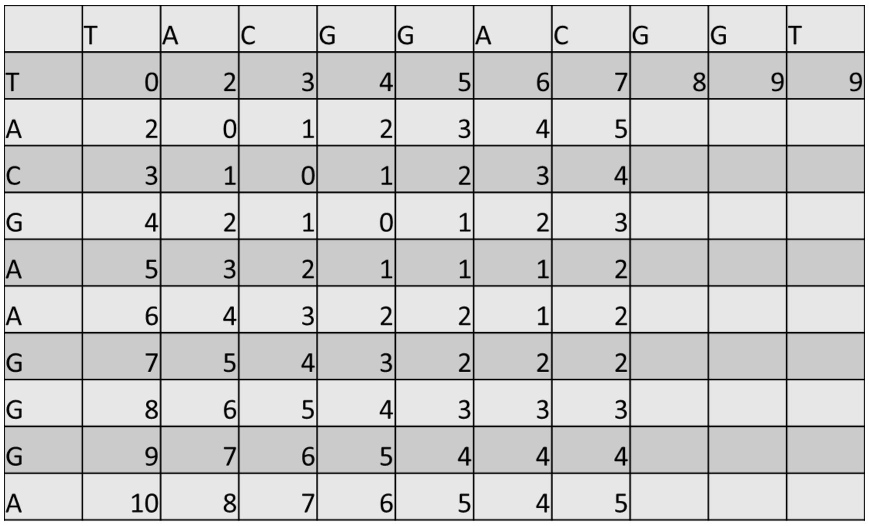

j
H a v e a h s p p y d a y
NULL
t_j
p_i
i
h
a
p
p
y
D[i-1][j-1]
D[i-1][j]
D[i][j-1]
D[i-1][j-1]
▪MatchCost = D[i-1][j-1] , if p_i = t_j
▪ReviseCost = D[i-1][j-1]+1 , if p_i ≠ t_j
▪InsertCost = D[i-1][j]+1
▪DeleteCost = D[i][j-1]+1
▪MatchCost = N/A
▪ReviseCost = 3
▪InsertCost = 2
▪DeleteCost = 4
-> Min = 2

Figure 3.1: Approximate String Matching Example

3.4.1 Reverse Compliment

As noted in the previous chapter on DNA sequences, while being represented by a single string of letters that represent the bases, each character actually represents a base pair, and due to difficulties in obtaining the data there are no guarantees on the ‘direction’ it is represented. In the case of a reversed gene the reverse compliment can easily be calculated by reading the sequence in reverse and in place of the base being read, the base’s pair is substituted. It is possible to apply heuristics to determine when to select the reverse compliments, such looking at the number of matches attained thus far, but as it is uncertain which of the sequences might need to be reversed, it is necessary to check this on any mismatch. However, in the case that a sequence is reversed, it is likely to very quickly fall through onto a branch pruning as it will have significantly more difference than the small threshold.



	T	A	C	G	G	A	C	G	G	T
T	0	2	3	4	5	6	7	8	9	9
A	2	0	1	2	3	4	5			
C	3	1	0	1	2	3	4			
G	4	2	1	0	1	2	3			
A	5	3	2	1	1	1	2			
A	6	4	3	2	2	1	2			
G	7	5	4	3	2	2	2			
G	8	6	5	4	3	3	3			
G	9	7	6	5	4	4	4			
A	10	8	7	6	5	4	5			

Figure 3.2: Sequence by Sequence Example

3.5 Grouping Algorithm

Once all the sequences have been compared to each other and it determined if they fall within the similarity threshold of each other they need to be grouped into similar species. The initial algorithm used for the grouping of the samples was a greedy algorithm that first takes sequence with the most matches within the threshold and grouped all those sequences with it. This however tended to yeild less than optimal results as the sequence with the most matches tended to be amongst the shortest sequences in the set. This meant that longer sequences that were grouped with this one often where not themselves matches with each other. So instead of using the heuristic of sorting by the sequence with the most matches it was found that sorting by the length of the sequence worked significantly better. It follows that the resulting groups are no longer displayed with the largest group first running down to the outliers, but the sequences are grouped in better matching sets.

3.6 Hardware Requirements

The hardware requirements to calculate a sample of 10,000 sequences of 400-800 bases long are as follow: The majority of the memory is consumed by the difference table containing the difference between each of the sequences. This falls in the order of

```
n(n-1)/2
```

bytes. In addition, with the sequence's data and the difference table for the current sequence under examination, this totals approximately 200MB.

Chapter 4

Results

4.1 Output

The output of the system as requested by the bioinformaticians is to be sequences grouped with their best matched group. They also requested the names of all sequences in the group be available so that they may later be inspected individually. In the meeting the above was briefly discussed. It was expected that another meeting would take place with the current output as an examples so that the bioinformaticians could examine and have further input into how they would like to see the results. Thus the format of the results is simply the output of the groupings in .csv file format, with the longest sequence heading the first row, followed by all sequences that fall within the difference threshold with it and finally a count thereof. Each following line undergoes the same process on the remaining sequences. In this case, the longest sequence *HK2QS7R01BEEV6* happens to be the only one in its group, but the grouping can be observed in the cases that follow. The output in table 4.1 is a sample from the output from the processing of the sample *Kowie_W1_gDNA.trim.fasta.*

Table 4.1: Output File Example

Processed:Kowie_W1_gDNA.trim.fasta at 3.0000 %threshold

Seq[969]	HK2QS7R01BEEV6		1	
Seq[1827]	HK2QS7R01BMIU5	Seq[1020]:HK2QS7R01BM4U3	Seq[1697]:HK2QS7R01AB41M	...
Seq[1633]	HK2QS7R01BADMH	Seq[1958]:HK2QS7R01ASAZ4	Seq[2788]:HK2QS7R01BXDXQ	...
Seq[1616]	HK2QS7R01A0MHC	Seq[2404]:HK2QS7R01AN3O6	Seq[1456]:HK2QS7R01BAHY0	...
Seq[512]	HK2QS7R01BI86J	Seq[436]:HK2QS7R01BZQ6V	Seq[1904]:HK2QS7R01AQWBB	...
...				

4.2 Performance Results

Testing was done on a Intel Core i7 870 with 4GB of RAM on Ubuntu 11.10. In order to obtain statistically accurate results the program was run multiple times on a range of sample sets of sequences: 25 to 1250 sequences in increments of 25. Four of these tests were run concurrently on separate data files. The program was modified to limit the number of sequences to this number rather than considering the entire file, even when the largest file was used the overhead was still negligible — a fraction of a second compared to the hours to take to run a sample of that size. In addition, testing was done on the largest data set available, which contained 11764 sequences the results of which were:

```
Sequences: 11764
Comparisons: 69189966
Elapsed Time: 38201.69922
Comparisons per second: 1811.175089
```

This test was run separate from other processing and as a result suffered from less low level cache swapping. The four larger tests were run concurrently as it is the intention of the biology department to run multiple samples at the same time rather than utilizing multiple cores for parallelisation. It is important to first note that in a set of n sequences, (n-1)n/2 comparisons are required, this can be clearly seen on Figure 4.1. We should therefore expect that the program run in at least order n squared time. This is supported by Figure 4.2 which shows that the number of comparisons achieved per second is independent of the scale of the problem. Each of T1 to T4 represent different sample files. The difference in speed between the fastest, T3, and the slowest, T1, can be explained by each sample managing to take advantage of the branch pruning at different rates. There might appear to be a slight downward trend to this graph but it should be noted that the smallest samples were completing in fractions of a second and therefore their accuracy can not be compared to that of the larger samples. However the larger samples could be subject to a higher low level cache miss rate. The majority of the processing time is done on sequence by sequence comparisons. Thus the amount of memory required is proportional to the length of the sequences and not the total number of sequences being compared. This can be seen from the fact that the large sample still managed to run at a high speed.

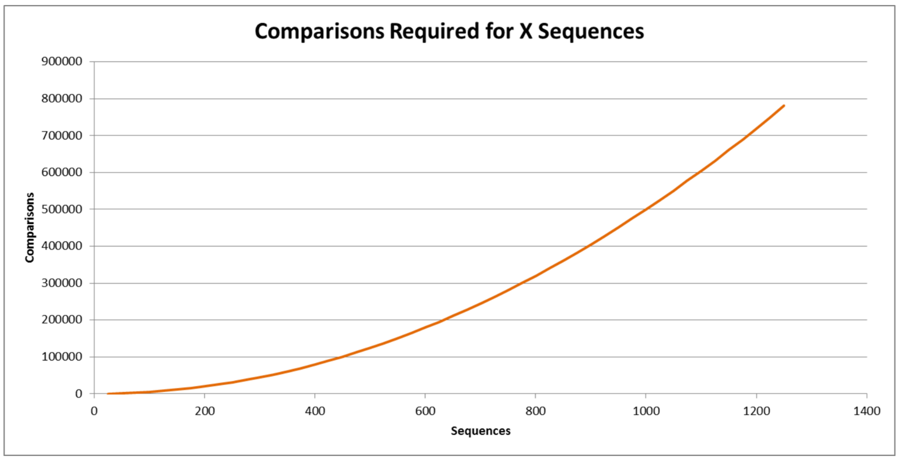


Figure 4.1: Comparisons Required for X Sequences

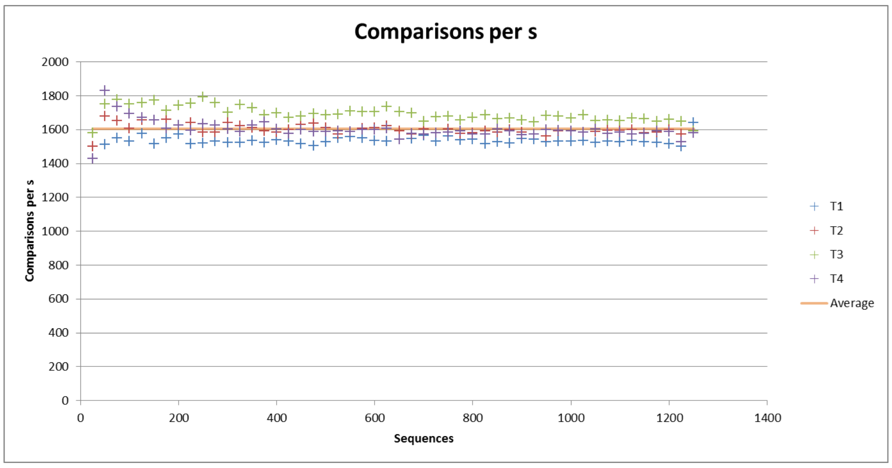


Figure 4.2: Comparisons per Second

It can thus be seen from figure 4.3 that there is a linear relationship between the number of comparisons and processing time. This is further supported by observing that overhead, Table B.5 and Table B.4, everything other than calculating the sequence matches falls to less than 1/100th of a percentage of the processing time.

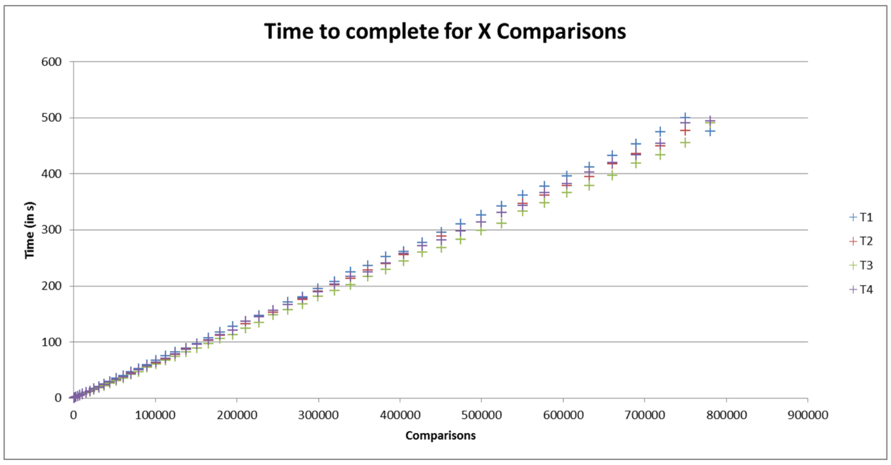


Figure 4.3: Overal Time to Complete for X Comparisons

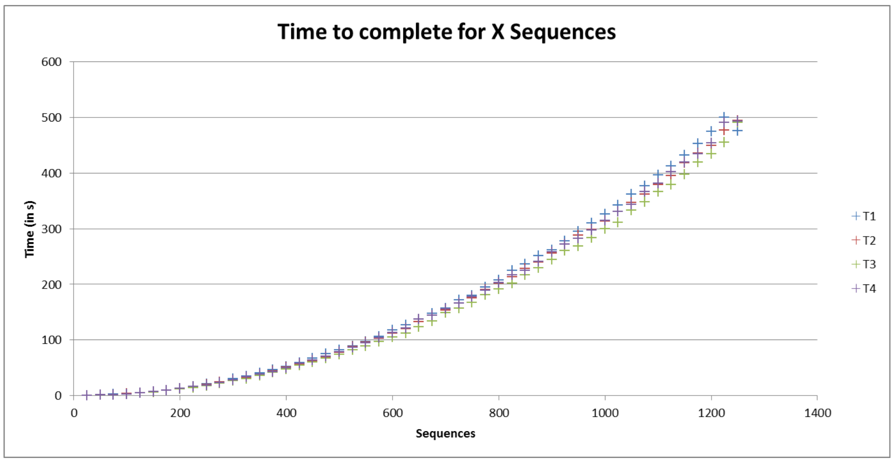


Figure 4.4: Overall Time to Execute for X Sequences

Figure 4.4 confirms these results as a very close relationship between it and the order n squared plot in Figure 4.1 can be seen.

Chapter 5

Conclusion

5.1 Discussion

The goal of this project is to seek an understanding of bioinformatic sequence analysis and combine it with string matching algorithms in order to create a specific tool to aid bioinformaticians in their research. Their research involves an analysis of large sets of bacterial DNA in order to group them into inferred species groups by assuming that samples whose genes are similar within a small threshold are of the same species. Their requirement came from the fact that there is no existing tool to solve this problem and using the tools available resulted in them having to do entire alignments of each of the combinations of sequences. They reported that this took as long as ten days to process. However, taking into account the fact that an entire alignment is not needed and the assumption that the sequences are already globally aligned, a variation of Sara Baase's approximate string matching algorithm can be applied, which can take into account maximum advantage of dynamic programming and branch pruning to minimise the required processing at the critical point of this problem. This is supported by the results showing that the overhead time for every other part of the program accounted for less than one hundredth of a percentage of processing time for all but the smallest of samples. In addition the results showed that concurrent processing of the data achieved a stable rate of 1600 sequences per second which means that processing time for the large data sets reported to take ten days has been reduced to eight to ten hours.

5.2 Future Extensions

5.2.1 Comparative Analysis

One main section that is lacking from this paper is a comparative analysis with other similar systems. I could not find one with which to compare results and examine algorithms. There are a number of possible reasons for a lack of similar systems primarily, I would put this to the fact that similarity does not imply homogeneity. The bioinformaticians, however, have chosen in their research to assume that it does for this specific gene under study. In addition, it has been noted that, where possible, DNA strings should be converted to their corresponding protein sequence before processing, as, due to the increased complexity of the sequence, they are both shorter and more statically accurate — meaning that there is less likely to be a chance case of matching bases.

5.2.2 Graphical User Interface

The bioinformaticians mentioned in a discussion that they are not always as computer literate as those that develop these kinds of systems and as such they would like to interface with tools like this one through a GUI. Apart from making a GUI for this specific tool it could be possible to make a GUI that can aid biologists with many tools such as this.

5.2.3 Parallelisation

Noted above as one of the limitations of this project as per the requirements from the bioinformaticians, parallelisation of this project could yield near linear speed–ups. This is due to the large amount of comparisons that can each be viewed as an independent sub-problem. In fact, as part of the masters research of Dale Tristrim on GPU arrays, another student in the Rhodes University Computer Science Department, he has implemented a similar algorithm and has reported being able to complete processing on a large sample of sequence data in as little as 12 minutes. He has asked me to check his algorithm and results to ensure they are correct as this is not the focus of his research and I intend to do this using the algorithm developed above in the coming month.

Bibliography

[1] Oxford English Dictionary: Bioinformatics. [online]. Accessed on 2 April 2012. Available from: `http://www.oed.com/view/Entry/255935`.

[2] Baase, S., and Van Gelder, A. *Computer Algorithms: Introduction to Design and Analysis.* Addison-Wesley, 2000.

[3] Baldi, P., and Brunak, S. *Bioinformatics: The Machine Learning Approach.* Adaptive Computation and Machine Learning. Mit Press, 2001.

[4] Brudno, M., Do, C. B., Cooper, G. M., Kim, M. F., Davydov, E., Program, N. C. S., Green, E. D., Sidow, A., and Batzoglou, S. Lagan and multi-lagan: Efficient tools for large-scale multiple alignment of genomic dna. *Genome Research 13*, 4 (2003), 721–731.

[5] Buckley, M., and Roberts, R. Reconciling microbial systematics and genomics, 2006. American Academy of Microbiology.

[6] Crick, F., and Watson, J. Molecular structure of nucleic acids: A structure for dna. *Nature 171* (April 1953), 737 – 738.

[7] Gibas, C., and Jambeck, P. *Developing Bioinformatics Computer Skills.* O'Reilly Series. O'Reilly, 2001.

[8] Huerta, M., Downing, G., Haseltine, F., Seto, B., and Lie, Y. NIH Working Definition of Bioinformatics and Computational Biology. [online], July 2000. Accessed on 2 April 2012. Available from: `http://bisti.nih.gov/docs/CompuBioDef.pdf`.

[9] Jones, N., and Pevzner, P. *An Introduction To Bioinformatics Algorithms.* Computational Molecular Biology. Mit Press, 2004.

[10] Kanehisa, M. *Post-Genome Informatics.* Post-genome Informatics. Oxford University Press, 2000.

[11] Krawetz, S., and Womble, D. *Introduction to Bioinformatics: A Theoretical And Practical Approach.* Humana Press, 2003.

[12] Li, H., and Homer, N. A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in Bioinformatics 11*, 5 (2010), 473–483.

[13] Nardone, J., Lee, D. U., Ansel, K. M., and Rao, A. Bioinformatics for the 'bench biologist': how to find regulatory regions in genomic dna. *Nature Immunology 5*, 8 (Aug. 2004), 768–774.

[14] Tramontano, A. *Introduction to Bioinformatics.* Chapman and Hall mathematics series. Chapman & Hall/CRC, 2007.

[15] Won, J.-I., Park, S., Yoon, J.-H., and Kim, S.-W. An efficient approach for sequence matching in large dna databases. *Journal of Information Science 32*, 1 (2006), 88–104.

[16] Xu, D., Keller, J., Popescu, M., and Bondugula, R. *Applications of Fuzzy Logic in Bioinformatics.* Series on Advances in Bioinformatics and Computational Biology. Imperial College Press, 2008.

Appendix A

Code

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <time.h>

typedef struct{
  int index;
  char *sequenceName;
  char *sequenceData;
  char *revComp;
  int sequencelen;
  int count;
  int used;

  //could add an array of pointers to all the sequences that fall within
  // the threshold. this enabling any sequence to be selected as a
  // group at the choice of the bioinformaticians
} SequenceStruct;
```

```
int LoadFile(char *filename, char* data[], long *filesize);
int CountSequences(char* data, long length);
int LocateSequences(char* data, long length, SequenceStruct *sequences[]);
short int Match(char* seq0, char* seq1, int minLen, float t);
int RevComp(char* sequenceData,char* revComp,int length);

//the two cmp methods for the above struct
int struct_cmp_count(const void *a, const void *b) {
    const SequenceStruct *ia = *(const SequenceStruct **) a;
    const SequenceStruct *ib = *(const SequenceStruct **) b;
    return ib->count - ia->count;
}
int struct_cmp_len(const void *a, const void *b) {
    const SequenceStruct *ia = *(const SequenceStruct **) a;
    const SequenceStruct *ib = *(const SequenceStruct **) b;
    return ib->sequencelen - ia->sequencelen;
}

int main(int argc, char *argv[])
{
clock_t c0, c1;//clocks used for timings
int i,j;//variables for loops

long length; // the length of the data file.
char *data; // a variable to hold the file data - this data will be
//manipulated and the Sequence Array will have pointers into this data.
SequenceStruct **s;//array to hold all information that
                            //pertains to the DNA sequences
float threshold = 0.03f;// matches will be considered less than this %
```

```
if (argc == 3) {
    if (0 == sscanf(argv[2], "%f", &threshold)
         ||threshold>1||threshold<0) {
      printf("Threshold incorrect format\n");
      return 0;
    }
}
printf("Threshold set at %f %%\n",threshold*100);

char* filename = argv[1];
if (0 != LoadFile(filename, &data, &length)) { return 1; }
// memory has been allocated to data, remember to free it up
//File Loaded.

int numSequences = CountSequences(data, length);
if (0==numSequences) {
printf("No Sequences found. Incorrect input file or bad format?\n");
return 0;
}
//we now know how many sequences we are dealing with.

s = malloc(numSequences*sizeof(SequenceStruct*));
//allocated space for the sequence array

for(i = 0; i < numSequences; i++) {
  s[i] = malloc(sizeof(SequenceStruct));
  s[i]->index = i;
  //s[i]->sequencelen = strlen(s[i]->sequenceData);
//not calculated yet line moved below
  s[i]->count = 0;
  s[i]->used = 0;
  s[i]->revComp = NULL;
}
printf("%d sequences found in the file.\n",numSequences);
if (0!=LocateSequences(data,length,s)) {
free(s);
return 1;
}
// now we have found the sequences with no problems.
// build the difference table and we can also now allocate
// the sequence lengths as they have been calculated now
short int **D;
D = malloc(numSequences*sizeof(int*));
for (i=0;i<numSequences;i++) {
D[i] = malloc(numSequences*sizeof(int));
//this line moved here as after LocateSequences
s[i]->sequencelen = strlen(s[i]->sequenceData);
s[i]->revComp=malloc(s[i]->sequencelen*sizeof(char));
sprintf(s[i]->revComp,"X\0");
}
//--------------------------------------------------------------
printf("-- Matching Loop --\n");
// this is where the bulk of the processing occurs
int totalcomparisons = (numSequences-1)*numSequences/2;
//the number of comparisons required for this number of sequences
c0 = clock();//start the clock to time the comparisons
for(i = 0; i < numSequences; i++) {

for(j = 0; j < numSequences; j++) {
if (j==i) { D[i][j] = -1; }
else if (j<i) { D[i][j] = D[j][i]; }
else {
    int minLen = s[i]->sequencelen<s[j]->sequencelen?
s[i]->sequencelen:s[j]->sequencelen;
    D[i][j] = Match(s[i]->sequenceData,s[j]->sequenceData,minLen,threshold);
    if (0==D[i][j]) {
      if (s[j]->revComp[0] == 'X')
RevComp(s[j]->sequenceData,s[j]->revComp,s[j]->sequencelen);
//lazily evaluate if we need the revComp or not.
      D[i][j] = Match(s[i]->sequenceData,s[j]->revComp,minLen,threshold);
    }
    if (1==D[i][j]) {
      s[i]->count++;
      s[j]->count++;
    }
}
}
c1 = clock();
float elapsedtime = (float) (c1 - c0) / CLOCKS_PER_SEC;
int remainingsequences = (numSequences - i - 1);//improve these calculations
int remainingcomparisons = (remainingsequences - 1) * remainingsequences / 2;
float estimated = elapsedtime/(totalcomparisons - remainingcomparisons)
* remainingcomparisons;
//console output
printf("\rSequence[%.5d]:%s     %4.d matches within threshold.
Elapsed: %.3f     ~Remaining: %.3f     ~Total:%.2f      "
, i, s[i]->sequenceName,s[i]->count,
elapsedtime, estimated, elapsedtime + estimated);
fflush(stdout);
}
```

```
c1 = clock();
float sec = (float) (c1 - c0)/CLOCKS_PER_SEC;
printf("\n\nSequences:%d\tComparisons:%d\tElapsed Time: %f
\tComparisons per second: %f\n"
,numSequences,totalcomparisons,sec,totalcomparisons/sec);

// at this point we have the lists of all sequences and if they meet the
//  threshold match to each other and a count of how many matches they make
//------------------------------------------------------------
/*printf("Writing Difference Table To File\n");
FILE *f;//Result file to view number of differences between all sequences.
f = fopen("Output/DifferenceTable.csv","w");
for(i = -1; i < numSequences; i++) {
for(j = -1; j < numSequences; j++) {
if (i == -1) {
if(j == -1) { fprintf(f,",,"); }
else { fprintf(f,"Seq[%d],",s[j]->index); }
}
else {
if(j == -1) { fprintf(f,"%d,Seq[%d],",s[i]->sequencelen,s[i]->index); }
else { fprintf(f,"%4d,",D[s[i]->index][s[j]->index]); }
}
}
if (i == -1) { fprintf(f,",Matches,\n"); }
else { fprintf(f,",,%d\n",s[i]->count); }
}
fclose(f);*/
//------------------------------------------------------------
//qsort(s, numSequences, sizeof(SequenceStruct*), struct_cmp_count);
qsort(s, numSequences, sizeof(SequenceStruct*), struct_cmp_len);
```

```
//make and arg to allow for the choice between
//the longest and the best matches?
FILE *f2;
//Result file to view the sequences that match within the threshold
//   - the species groupings
int filenamelen = strlen(filename);
char *outputfilename =  malloc((18+filenamelen)*sizeof(char));
sprintf(outputfilename,"MatchesFound-%s.csv\0",filename);
f2 = fopen(outputfilename,"w");
fprintf(f2,"Processed:%s at %f %%threshold\n",filename,threshold*100);
int current_index;
for(i = 0; i < numSequences; i++) {
  if (s[i]->used == 0)
  {
    int count = 1;
    current_index = s[i]->index;
    s[i]->used = 1;
    fprintf(f2,"Seq[%d],%s,",s[i]->index,s[i]->sequenceName);
    for(j = i+1; j < numSequences; j++) {
      if (s[j]->used == 0 && D[current_index][s[j]->index]==1) {
fprintf(f2,"Seq[%d]:%s,",s[j]->index,s[j]->sequenceName);
count++;
s[j]->used = 1;
      }
    }
    fprintf(f2,",%d\n",count);
//would be nice to have this at the begining of
// the line following the initial sequence
  }
}
```

```
fclose(f2);
printf("Done writing file: %s\n",outputfilename);
//------------------------------------------------------------
for (i = 0;i<numSequences;i++) {//free all data
free(D[i]);
free(s[i]->revComp);
free(s[i]);
}
free(outputfilename);
free(D);
free(data);
free(s);
return 0;
}
//return of 0 means mismatch return of 1 means match
short int Match(char* seq0, char* seq1, int minLen, float t){
//include threshold for branch pruning
int i,j;
int threshold = (int)(t*minLen);//same difference as floorf

int **D;// the difference table
D = malloc(minLen*sizeof(int*));
for (i=0;i<minLen;i++) {
  D[i] = malloc(minLen*sizeof(int));
  for (j=0;j<minLen;j++) {
    D[i][j] = threshold+2;
  }
}
for(i = 0; i < minLen; i++) {//initial data
D[0][i] = i+(seq0[i]!=seq1[0]?1:0);
```

```
D[i][0] = i+(seq0[0]!=seq1[i]?1:0);
}
//for(i = 0; i < minLen; i++) D[i][0] = i+(seq0[0]!=seq1[i]?1:0);
//incorperated these two loops into a single one
//for(j = 0; j < minLen; j++) D[0][j] = j+(seq0[j]!=seq1[0]?1:0);
short int match = -1;
int minmatches;
for(i = 1; i < minLen; i++) {
minmatches = minLen;
for(j = 1; j < minLen; j++) {
//MatchCost or ReviseCost
int x = D[i-1][j-1]+(seq0[i]!=seq1[j]?1:0);
//min of the InsertCost and DeleteCosts
int y = (D[i][j-1]<D[i-1][j]?D[i][j-1]:D[i-1][j])+1;
//Min
D[i][j] = y<x?y:x;
if (D[i][j] < minmatches) minmatches = D[i][j];
}
if (minmatches>threshold) {
  //as the algorithm can never yeild a result below this value
  // we can break out of the loop knowing this is a mismatch.
  for (i = 0;i<minLen;i++) free(D[i]);
  free(D);
  return 0;
}
if(D[i][minLen-1]<=threshold) {
  //successful match found
  for (i = 0;i<minLen;i++) free(D[i]);
  free(D);
  return 1;
}
}
//just a note it is unlikely to reach here
for (i = 0;i<minLen;i++) free(D[i]);
free(D);
if (match == -1 && minmatches<=threshold) return 1;
return 0;
}
int LoadFile(char* filename, char* data[], long *filesize) {
//Check Valid filename? File exists etc...
printf("Loading file: %s\n",filename);

FILE* file = fopen(filename,"rb");
if (NULL == file) {//ERROR: Cannot open file
printf("Error: Cannot open file '%s'\n\n",filename);
return 1;
}
if (-1 == fseek(file, 0L, SEEK_END)) {//Error: Problem Reading File
fclose(file);
printf("Error: Problem Reading File '%s'\nfseek(end)\n",filename);
return 2;
}
long fs = ftell(file);
*filesize = fs;
if (*filesize == -1) {//Error: Problem Reading File
fclose(file);
printf("Error: Problem Reading File '%s'\nftell\n",filename);
return 2;
}
if (-1 == fseek(file, 0L, SEEK_SET)) {//Error: Problem Reading File
```

```
fclose(file);
printf("Error: Problem Reading File '%s'\nfseek(set)\n",filename);
return 2;
}
//Alternate code which would be more condenced but encountered an error
//if (-1 == fseek(file, 0L, SEEK_END)||
// -1 == *(filesize = ftell(file))||
// -1 == fseek(file, 0L, SEEK_SET)) {//error
// fclose(file);
// printf("Error: Problem Reading File '%s'\n\n",filename);
// return 2;
//}
printf("Reading file data...");
*data = malloc(sizeof(char)*(*filesize+1));
if (NULL == data) {//Error: Encountered a problem allocating memory
printf("Error: Encountered a problem allocating memory\n\n");
return 3;
}
size_t n = fread(*data,sizeof(char),*filesize,file);
fclose(file);
if (n<1) {//Error: Problem Reading File
free(data);
printf("Error: Problem Reading File '%s'\n\n",filename);
return 4;
}
*(*data+n) = '\0';// string terminating null character

printf("Done.\n");
return 0;
}
```

```
int CountSequences(char* data, long length) {
//search for the sequences... ASSUMES '>' is reserved for the
//begining of the header line and nowhere else (not in the comments etc.)
int i = 0, c = 0;
for (; i < length; i++) { if (data[i] == '>') c++; }
return c;
}

//int LocateSequences(char* data, long length, SequenceStruct sequences[]);
int LocateSequences(char* data, //Data from the FASTA File Format
      long length, //the length of the file data
      SequenceStruct *sequences[]) {
      // returns 0 if no errors are encountered
printf("Locating Sequences...");
int i = 0,  //the index of the character in the file under examination
  j = 0,  //the index for the destination of the current
              //character in the resulting data.
  c = 0; //the sequence under examination
//ensure first character is a '>'
if (data[i]!='>') {//ERROR: File does not begin with header line
printf("Error: Bad file format - Does not being with a header line\n\n");
return 1;
}
for (; i < length; i++) {
switch(data[i])
{
case '>':
data[j++] = '\0';
sequences[c]->sequenceName=&data[i+1];
```

```
while (i<length && !(data[i]=='\n'||data[i]=='\r'))
{
  if (data[i]==' ')
  {
    data[i] = '\0';
  }
  i++;
}
data[i] = '\0';
i++;
if (i >= length){//ERROR: We should not expect to find the end of file here
printf("Error: Unexpected end of file\n");
return 2;
}
//we are looking at the beinging of the next sequence's data
//printf("Found sequence[%d] at %d\n",c,i);
sequences[c++]->sequenceData=&data[i];
j=i+1;//Abnormal details might cause a problem here,
//such as sequences of 1 or 0 bases which really should never occur.
break;
case '\n':
case '\r':
break;//skip newlines
case 'A'://Expected or Allowed Data
case 'C':
case 'G':
case 'T':
//here we could add more cases for things like 'N' which means any
//- but special allowances need to be made in the Match method
//as an N matching an N would result in a mismatch instead of a match.
```

```
*(data+j++) = *(data+i);
break;
default:
printf("Error: Bad file format or format not supported.
                       \nFound character %c at byte %d\n\n",data[i],i);
return 3;
break;
}
}
*(data+j++) = '\0';//string terminating null character
printf("Done!\n");
return 0;
}
int RevComp(char* sequenceData,char* revComp,int length){
  int i;
  for (i = length;i>=0;--i)
  {
    char c = sequenceData[i];
    if (c == 'T') c = 'A';
    else if (c == 'A') c = 'T';
    else if (c == 'C') c = 'G';
    else if (c == 'G') c = 'C';
    revComp[length-i-1]=c;
  }
  return 0;
}
```

Appendix B

Data Results

Table B.1: Large Dataset Runtime

Sequences	11764
Comparisons	69189966
Elapsed Time (s)	38201.69922
Comparisons per second	1811.175089

Table B.2: Time to complete for x sequences

sequences	comparisons	T1	T2	T3	T4
25	300	0.21	0.2	0.19	0.21
50	1225	0.81	0.73	0.7	0.67
75	2775	1.79	1.68	1.56	1.6
100	4950	3.23	3.08	2.83	2.92
125	7750	4.92	4.68	4.41	4.64
150	11175	7.38	6.75	6.3	6.75
175	15225	9.82	9.17	8.88	9.47
200	19900	12.66	12.24	11.42	12.25
225	25200	16.64	15.36	14.35	15.78
250	31125	20.48	19.64	17.37	19.04
275	37675	24.59	23.8	21.44	23.2
300	44850	29.44	27.33	26.35	27.99
325	52650	34.57	32.45	30.13	33.14
350	61075	39.8	37.91	35.38	37.59
375	70125	46.04	44.01	41.59	42.63
400	79800	51.82	50.43	47.01	49.8
425	90100	58.84	56.34	53.91	57.12
450	101025	66.59	61.98	60.21	63.18
475	112575	74.88	68.81	66.46	70.85
500	124750	81.69	77.5	73.94	78.49
525	137550	88.72	87.48	81.43	86.5
550	150975	96.85	95.1	88.41	95.07
575	165025	106.48	102.61	96.72	103.29
600	179700	117.11	111.55	105.39	112.39
625	195000	127.23	120.28	112.35	121.23
650	210925	136.66	132.39	123.71	136.67
675	227475	147.22	144.56	133.89	144.35
700	244650	156.26	152.75	148.32	155.63
725	262450	171.27	165.96	156.69	166.05
750	280875	179.83	175.13	167.15	177.26
775	299925	194.92	190.3	181.02	188.64
800	319600	207.08	202.2	191.18	203.2
825	339900	224.51	213.39	201.67	216.14
850	360825	236.26	227.82	216.67	224.98
875	382375	251.62	239.4	229.2	240.33
900	404550	261.53	255.28	244.43	258
925	427350	277.35	271.5	259.82	271.6
950	450775	295.05	288.41	268.18	281.88
975	474825	310.45	298.19	283.1	297.53
1000	499500	326.11	313.61	299.28	314.05
1025	524800	341.72	330.99	311.45	331.2
1050	550725	361.78	346.54	333.36	343.27
1075	577275	377.12	362.08	348.31	366.37
1100	604450	396.28	379.15	365.9	381.84
1125	632250	412.14	395	378.85	402.18
1150	660675	432.17	417.73	397.05	419.37
1175	689725	452.79	435.86	418.8	433.74
1200	719400	474.19	449.43	433.81	453.54
1225	749700	499.45	476.79	455.08	490.96
1250	780625	476.11	494	490.19	493.61

Table B.3: Comparisons Achieved per Second

sequences	comparisons	T1	T2	T3	T4	Average
25	300	1428.57	1500	1578.95	1428.57	
50	1225	1512.35	1678.08	1750	1828.36	
75	2775	1550.28	1651.79	1778.85	1734.38	
100	4950	1532.51	1607.14	1749.12	1695.21	
125	7750	1575.2	1655.98	1757.37	1670.26	
150	11175	1514.23	1655.56	1773.81	1655.56	
175	15225	1550.41	1660.31	1714.53	1607.71	
200	19900	1571.88	1625.82	1742.56	1624.49	
225	25200	1514.42	1640.63	1756.1	1596.96	
250	31125	1519.78	1584.78	1791.88	1634.72	
275	37675	1532.13	1582.98	1757.23	1623.92	
300	44850	1523.44	1641.05	1702.09	1602.36	
325	52650	1523	1622.5	1747.43	1588.71	
350	61075	1534.55	1611.05	1726.26	1624.77	
375	70125	1523.13	1593.39	1686.1	1644.97	
400	79800	1539.95	1582.39	1697.51	1602.41	
425	90100	1531.27	1599.22	1671.3	1577.38	
450	101025	1517.12	1629.96	1677.88	1599	
475	112575	1503.41	1636.03	1693.88	1588.92	
500	124750	1527.11	1609.68	1687.18	1589.37	
525	137550	1550.38	1572.36	1689.18	1590.17	
550	150975	1558.85	1587.54	1707.67	1588.04	
575	165025	1549.82	1608.27	1706.21	1597.69	
600	179700	1534.45	1610.94	1705.1	1598.9	
625	195000	1532.66	1621.22	1735.65	1608.51	
650	210925	1543.43	1593.21	1705	1543.32	
675	227475	1545.14	1573.57	1698.97	1575.86	
700	244650	1565.66	1601.64	1649.47	1572	
725	262450	1532.38	1581.41	1674.96	1580.55	
750	280875	1561.89	1603.81	1680.38	1584.54	
775	299925	1538.71	1576.06	1656.86	1589.93	
800	319600	1543.36	1580.61	1671.72	1572.83	
825	339900	1513.96	1592.86	1685.43	1572.59	
850	360825	1527.24	1583.82	1665.32	1603.81	
875	382375	1519.65	1597.22	1668.3	1591.04	
900	404550	1546.86	1584.73	1655.08	1568.02	
925	427350	1540.83	1574.03	1644.79	1573.45	
950	450775	1527.79	1562.97	1680.87	1599.17	
975	474825	1529.47	1592.36	1677.23	1595.89	
1000	499500	1531.69	1592.74	1669.01	1590.51	
1025	524800	1535.76	1585.55	1685.02	1584.54	
1050	550725	1522.26	1589.21	1652.04	1604.35	
1075	577275	1530.75	1594.33	1657.36	1575.66	
1100	604450	1525.31	1594.22	1651.95	1582.99	
1125	632250	1534.07	1600.63	1668.87	1572.06	
1150	660675	1528.74	1581.58	1663.96	1575.4	
1175	689725	1523.28	1582.45	1646.91	1590.18	
1200	719400	1517.11	1600.69	1658.33	1586.19	
1225	749700	1501.05	1572.39	1647.4	1527.01	
1250	780625	1639.59	1580.21	1592.49	1581.46	1606.52

Table B.4: Overhead Time

sequences	comparisons	T1	T2	T3	T4
25	300	0.01	0.01	0.04	0.04
50	1225	0.01	0.02	0.04	0.05
75	2775	0	0.02	0.04	0.04
100	4950	0	0.02	0.04	0.04
125	7750	0	0.02	0.04	0.04
150	11175	0.01	0.02	0.04	0.04
175	15225	0.01	0.01	0.04	0.05
200	19900	0.01	0.01	0.04	0.04
225	25200	0.01	0.01	0.04	0.04
250	31125	0.01	0.02	0.04	0.04
275	37675	0.01	0.03	0.04	0.04
300	44850	0.01	0.03	0.04	0.05
325	52650	0	0.02	0.04	0.05
350	61075	0	0.03	0.04	0.04
375	70125	0.01	0.02	0.04	0.04
400	79800	0.01	0.02	0.04	0.04
425	90100	0.01	0.02	0.05	0.05
450	101025	0.01	0.01	0.05	0.04
475	112575	0.01	0.02	0.04	0.04
500	124750	0.01	0.02	0.05	0.04
525	137550	0	0.02	0.04	0.04
550	150975	0.01	0.02	0.04	0.04
575	165025	0	0.02	0.04	0.04
600	179700	0.01	0.02	0.04	0.04
625	195000	0.01	0.01	0.04	0.04
650	210925	0.01	0.02	0.04	0.05
675	227475	0.01	0.01	0.04	0.04
700	244650	0.01	0.02	0.06	0.04
725	262450	0	0.02	0.04	0.04
750	280875	0	0.02	0.04	0.05
775	299925	0.01	0.02	0.05	0.04
800	319600	0.01	0.03	0.04	0.06
825	339900	0.01	0.02	0.04	0.05
850	360825	0	0.01	0.04	0.06
875	382375	0.01	0.01	0.05	0.05
900	404550	0.01	0.02	0.05	0.04
925	427350	0.02	0.01	0.04	0.05
950	450775	0.01	0.01	0.04	0.05
975	474825	0	0.02	0.04	0.04
1000	499500	0.01	0.02	0.05	0.04
1025	524800	0.01	0.03	0.05	0.04
1050	550725	0.01	0.01	0.04	0.05
1075	577275	0.01	0.02	0.04	0.04
1100	604450	0	0.02	0.04	0.05
1125	632250	0.01	0.02	0.04	0.06
1150	660675	0.02	0.02	0.04	0.04
1175	689725	0	0.03	0.05	0.04
1200	719400	0	0.02	0.05	0.04
1225	749700	0.01	0.03	0.04	0.04
1250	780625	0	0.02	0.04	0.04

Table B.5: Overhead Percentage

sequences	comparisons	T1	T2	T3	T4
25	300	4.76%	5%	21.05%	19.05%
50	1225	1.23%	2.74%	5.71%	7.46%
75	2775	0%	1.19%	2.56%	2.5%
100	4950	0%	0.65%	1.41%	1.37%
125	7750	0%	0.43%	0.91%	0.86%
150	11175	0.14%	0.3%	0.63%	0.59%
175	15225	0.1%	0.11%	0.45%	0.53%
200	19900	0.08%	0.08%	0.35%	0.33%
225	25200	0.06%	0.07%	0.28%	0.25%
250	31125	0.05%	0.1%	0.23%	0.21%
275	37675	0.04%	0.13%	0.19%	0.17%
300	44850	0.03%	0.11%	0.15%	0.18%
325	52650	0%	0.06%	0.13%	0.15%
350	61075	0%	0.08%	0.11%	0.11%
375	70125	0.02%	0.05%	0.1%	0.09%
400	79800	0.02%	0.04%	0.09%	0.08%
425	90100	0.02%	0.04%	0.09%	0.09%
450	101025	0.02%	0.02%	0.08%	0.06%
475	112575	0.01%	0.03%	0.06%	0.06%
500	124750	0.01%	0.03%	0.07%	0.05%
525	137550	0%	0.02%	0.05%	0.05%
550	150975	0.01%	0.02%	0.05%	0.04%
575	165025	0%	0.02%	0.04%	0.04%
600	179700	0.01%	0.02%	0.04%	0.04%
625	195000	0.01%	0.01%	0.04%	0.03%
650	210925	0.01%	0.02%	0.03%	0.04%
675	227475	0.01%	0.01%	0.03%	0.03%
700	244650	0.01%	0.01%	0.04%	0.03%
725	262450	0%	0.01%	0.03%	0.02%
750	280875	0%	0.01%	0.02%	0.03%
775	299925	0.01%	0.01%	0.03%	0.02%
800	319600	0%	0.01%	0.02%	0.03%
825	339900	0%	0.01%	0.02%	0.02%
850	360825	0%	0%	0.02%	0.03%
875	382375	0%	0%	0.02%	0.02%
900	404550	0%	0.01%	0.02%	0.02%
925	427350	0.01%	0%	0.02%	0.02%
950	450775	0%	0%	0.01%	0.02%
975	474825	0%	0.01%	0.01%	0.01%
1000	499500	0%	0.01%	0.02%	0.01%
1025	524800	0%	0.01%	0.02%	0.01%
1050	550725	0%	0%	0.01%	0.01%
1075	577275	0%	0.01%	0.01%	0.01%
1100	604450	0%	0.01%	0.01%	0.01%
1125	632250	0%	0.01%	0.01%	0.01%
1150	660675	0%	0%	0.01%	0.01%
1175	689725	0%	0.01%	0.01%	0.01%
1200	719400	0%	0%	0.01%	0.01%
1225	749700	0%	0.01%	0.01%	0.01%
1250	780625	0%	0%	0.01%	0.01%