Novel Algorithms for the Analysis and Manipulation of Short Genomic Sequences

---

A Dissertation

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

By

Otto Dobretsberger

May 2014

Novel Algorithms
for the Analysis and Manipulation
of Short Genomic Sequences

_____
Otto Dobretsberger

APPROVED:

_____
Dr. Ioannis Pavlidis

_____
Dr. Yuriy Fofanov

_____
Dr. Nikolaos V. Tsekos

_____
Dr. William Widger

_____
Dr. Jehan-François Pâris

_____
Dean, College of Natural Sciences and Mathematics

ACKNOWLEDGEMENTS:

I wish to express sincere appreciation to the Department of Computer Science for their extended long-term support and especially to Dr. Yuriy Fofanov for his vast reserve of patience and knowledge.

I would like to show my gratitude to Dr. William Widger, Dr. Ioannis Pavlidis, Dr. Nikolaos Tsekos, and Dr. Jehan-François Pâris for serving as members of the dissertation committee.

I am indebted to my numerous colleagues for their support and particularly to my teammates from the Institute for Pharmacology and Toxicology at UTMB Galveston.

This dissertation would have never been completed without the encouragement and devotion of my family and friends, especially my parents Otto Dobretsberger, MD, and Elfriede Dobretsberger, as well as my fiancée Kimberly Hartman.

Novel Algorithms
for the Analysis and Manipulation
of Short Genomic Sequences

An Abstract of a Dissertation

Presented to

the Faculty of the Department of Computer Science

University of Houston

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

By

Otto Dobretsberger

May 2014

ABSTRACT

The storage, manipulation, and transfer of the large amounts of data produced by high-throughput sequencing instruments represent major obstacles to realizing the full potential of this promising technology. To date, significant effort has been devoted to efficiently compressing these cumbersome sequencing data sets, which are produced in two main text formats: FASTQ and FASTA. As an alternative to the current standard of storing all data, we contend that only high quality data need be stored and propose several new file formats to effectively refine and efficiently store such data. The presented file formats are specifically designed to store only high quality sequencing reads in space efficient text and binary formats. Additionally, we address the quality and redundancy issues of genetic reference databases required for a variety of investigations in the field of genomics. Presented modifications of non-alignment based sequence comparison algorithms address this challenge and make it possible to cluster together dozens of millions of genomic sequences (genes): one of the key challenges to reduce redundancy of genomic databases.

Table of Contents

1. Introduction

1.1 Challenges in Genomics caused by Next Generation Sequencing Technology Advances

Recent successes in medicine [1], biotechnology, and agriculture [3], were enabled or facilitated by the quantum leaps of genomics during the last decades. New technologies were invented and better and faster ways to use them were developed. The first major attainment was the introduction of large-scale genomic sequencing in 1977 by Frederick Sanger [4], who developed the Sanger sequencing method [5]. For nearly 25 years it remained the most widely used method to perform sequencing. In 2004, 454 Life Sciences introduced their first commercial Next Generation Sequencing (NGS) instrument, which opened up a completely new era of DNA sequencing. Soon, the market was shared among a hand full of competitors offering different instruments and sequencing approaches. Over the last years, those NGS instruments were improved and refined, which opened the doors to cheaper, faster, and more reliable access to DNA data. The rapid piling up of more and more sequencing data brought along a variety of new challenges in this industry. New methods and tools for data analysis were needed, and timely hardware upgrades needed to be performed, in order to be able to deal with the exponential growth of available DNA data.

1.2 Motivation: High Throughput Sequencing Data Storage, Manipulation, and Analysis Challenges

There are five main vendors which offer sequencing instruments. Illumina Inc. currently holds around 66% of the total sequencing market [6]. Illumina has been able to dominate the high end of the market, because they offer faster and more cost effective instruments than their competitors [8]. Other providers are Pacific Biosciences (PacBio) [9], Roche (454 Life Sciences) [10], Life Technologies (Ion Torrent) [11], and Applied Biosystems (SOLiD) [12]. Each product uses a different approach to produce DNA data from samples. For example, all Illumina sequencing instruments use the technique of sequencing by synthesis to produce DNA sequences, up to 3 billion reads per run [13]. The length of the sequences may vary from run to run; however, all sequences produced by the same run on an Illumina instrument are of equal length. Ion Torrent, SOLiD, and 454 instruments produce sequences of different lengths. The accuracy of the sequenced nucleotides by Illumina is about 98%, which is slightly lower than what some competitive instruments produce, such as the sequencers from 454 (99.9%), SOLiD (99.9%), or Pacific Bio (99.999%) [14] [15]. Due to the larger amount of data Illumina instruments produce, this slight deviation in accuracy can confidently be neglected. An overview of the different sequencing instruments and their properties can be seen in Table 1 and Table 2 [16].

Table 1: Sequencing instruments and their properties;

Method	Pacific Bio (single molecule real-time sequencing)	Ion Torrent (Ion semiconductor)	454 (pyrosequencing)	Illumina (sequencing by synthesis)	SOLiD (sequencing by ligation)	Sanger (chain termination)
Read length in base pairs	5,500 - 8,500; maximum >30,000	<400	<700	50 - 300	<100	400-900
Accuracy	99.999%	98%	99.9%	98%	99.9%	99.9%
Reads per run	50,000	<80 million	<1 million	<3 billion	1.2 - 1.4 billion	N/A
Time per run	30min - 2h	2h	24h	1 - 10 days	1 - 2 weeks	20min - 3h
Cost per 1 million bases	$0.33-$1.00	$1	$10	$0.05 to $0.15	$0.13	$2400

Table 2: The four sequencers currently produced and sold by Illumina;

	Mi Seq	**Next Seq 500**	**Hi Seq 2500**	**Hi Seq X**
Applications	Small genome, amplicon, and targeted gen panel sequencing	Everyday genome, exome, transcriptome sequencing	Production-scale genome, exome, transcriptome sequencing	Population-scale human whole-genome sequencing
Output Range	0.3-15 Gb	20-120 Gb	10-1000 Gb	1.6 - 1.8 Tb
Run Time	5-65h	12-30h	7h - 6d	<3d
Maximum Read Length	300bp	150bp	125 - 150bp	150bp

The number of existing sequencing instruments continues to increase, and so does the amount of data produced every day. At the same time the cost for both, instruments and data production, decreases continuously [17]. One of the main challenges that arise due to these circumstances is finding an efficient way how one can store this new massive amount of data. Hard drive sizes do not grow at the same exponential rate as the DNA data production does. Thus, many institutions face the problem of insufficient storage capacities. Another major challenge is the limitation of computer resources and hardware to adequately handle and perform diverse data manipulations. In order to perform certain

functionalities, often the entire data set needs to be available in main memory. Applying algorithms that exhibit bad time and memory complexities makes the use thereof unfeasible. The analysis of NGS data also faces new challenges. Most analysis requires performing comparisons with some sort of references. Several years ago, the number of references was limited, and searches against references were trivial. Today, with often thousands or millions of potential references available, certain steps in the analysis process are very hard to perform. Many times there are several copies of the same reference sequence present in genomic databases. Reference sequences with only minor differences such as single mutations (SNPs) are present numerous times, increasing the redundancy of the entire database. Thus, processes with exponential time complexities such as sequence alignment techniques are often impractical. This manuscript presents an attempt to address this problem using non pair wise comparison based sequence alignment techniques to identify sequence dissimilarities, and preliminarily exclude distant sequences from the sequence alignment procedure.

2. Data Storage and Novel File Formats

2.1 Problem Description

The currently used data formats for NGS data were designed around 1985, and are legacies of the FASTA Software Package introduced by David J. Lipman and William R. Pearson [18]. Although the sequencing industry has experienced many improvements since then, the file format itself has not changed. The amount of data that is produced on a daily basis was not nearly as much as it is today, and the file formats were not specifically designed to save space. Today, with millions of reads being produced by a single sequencing run, those file formats seem quite inadequate due to their initial design [19]. Modern NGS instruments produce data ranging from hundreds of Megabytes up to hundreds Gigabytes, per run. Many experiments and research projects require sequencing to be performed on multiple samples, and the accumulated amount of data often goes beyond the capabilities of conventional hard drives. Hence, special storage systems, such as RAID servers, frequently come to use [20]. Naturally, such solutions tag along vast amounts of expenses: for initial purchase, as well as service and upkeep. While one trend goes towards building larger and faster storage devices to store the data, another solution is to limit the data that need to be saved by utilizing data compression techniques. We differentiate between lossy and lossless data compression. While lossless data compression allows the original data to be completely reconstructed, lossy compression methods discard certain data sets. Similar approaches are used in digital image

compression, e.g., in the JPEG file format [21]. NGS data requires the usage of lossless data compression, due to the fact that single alteration in DNA sequences can change the functionality of genes completely. However, the original files obtained from sequencing instruments do contain information that is redundant, or irrelevant for most analyses [22]. Sequence headers often occupy more space than the actual sequences themselves. Additionally, hundreds of thousands of duplicated sequences can be present in a single file. Sequence quality scores are redundant information once sequences are known to be of good quality, but they are still stored in the FASTQ files.

Another major challenge besides storage is the need to share data. One cannot simply send NGS data via email, but instead has to utilize some sort of online storage system to provide others with data. This can be as simple as to grant others online access to one's own storage systems or the data can be uploaded so certain cloud storage providers. Regardless of where the data is located, in order to use someone else's data, it has to be downloaded first, which is usually very time consuming. Compressing the data before sharing it with others can reduce the traffic needed for uploads and downloads. Assuming a modern internet connection with a download bandwidth of 50 Mbit/s (Mega Bit per second), a file of 3-4 Gigabyte, such as the human genome, would take around 10 minutes to download. This might not seem very much, but there are two factors that need to be considered, and which can change the required time drastically: First, one does not always have a 50 Mbit/s connection. In fact, especially in areas without modern infrastructure, average bandwidths are often

even below 5 Mbit/s, as it is in Central America, or on the African continent. Assuming a bandwidth of 3 Mbit/s for example, it would take more than three hours to download the same data set. Secondly, whenever data needs to be shared, it often exceeds the amount of just a few Gigabytes. Especially unprocessed data sets obtained from a sequencing instrument can easily exceed a few hundred Gigabytes. This tremendous amount of data in combination with a bad or slow connection will always result in hours, if not days needed for data transfers.

A solution to this problem is to design and develop novel file formats that are appropriate and efficient for today's volume of generated sequences, and which do not store any redundant information such as sequence headers, sequence duplicates, or quality scores. Applications can also benefit from loading compressed data, especially if the decompression is done while opening or loading the file into memory, instead of involving a separate decompression step. Efficient compression allows to massively decrease the time needed to open or read a file, which often portraits a massive bottleneck.

2.2 Current Data Formats to Store High-Throughput Sequencing Data

Data obtained from Illumina sequencing instruments initially is stored in qseq text files [23]. Each line within the file represents one record, and each record represents one read. Unknown bases are shown as '.' symbols. The files contain the following information, which is tab delimited within each row:

Instrument name, run number, lane number, tile number, X coordinate, Y coordinate, index, read number, sequence, quality, filter. An example is shown in Figure 1.

```
SOLEXA5 1      4      1      1137   6698   0      1
TAAATCAAAAGCACAATGAGATATCAATTTTCACCCACTGGAATGGCTATA
aa]a]WY]F]aWaZWa]a][a]^]aaaaa_]Y``QaaaUa\aa]]YU`^]P     1
```

Figure 1: Example of a qseq file;

The qseq files are specific to Illumina instruments. Since typical analysis software is not tailored to process solely Illumina data, the qseq files are usually converted into one of the two standard file formats for NGS data: FASTA and FASTQ files.

Data produced by the 454 sequencer, the Ion Torrent, or the SOLiD instrument generally produce sequences of variable length in each run. The resulting data sets are saved in one of the two standard file formats, FASTA or FASTQ.

FASTQ

The FASTQ file format has been around for decades, but yet it has remained one of the two dominant standards in this industry [23]. FASTQ files generally consist of four lines per sequence or record. Each record starts with the '@' symbol in its first line, which is followed by the sequence identifier or sequence header. This identifier is composed of information about the sequencing instrument that was used, the flow cell lane, and other run specific information. The second line holds the characters representing the nucleotides (for DNA/RNA sequences) or amino

acids (for protein/polypeptide sequences). The third line starts with the '+' symbol and again shows the sequence identifier. Additional information or sequence description can be added in this line. The fourth line holds the quality scores for each individual nucleotide or amino acid from the second line, hence the lengths of line number two and line number four are identical (Figure 2). The quality scores are ASCII characters within a certain range, depending on the scoring scheme that was used for sequencing [24].

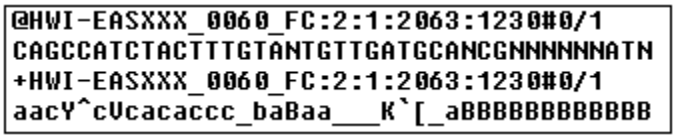

Figure 2: One record holding one sequence in FASTQ format;

Line two and four can span over several lines, especially for long sequences, genes, or whole genomes that are saved in the FASTQ format.

FASTA

The FASTA format has its origins in the FASTP software package (later FASTA software package), which was introduced in 1985 [25]. Together with the FASTQ format it is among the most widely accepted formats in the fields of genomics and bioinformatics. Many search and analysis tools accept sequences in FASTA format (several do also accept the FASTQ file format). The first line of each record (or sequence) in the FASTA format starts with the '>' symbol, followed by an identifier and an optional description (similar to the first line in the FASTQ

format). The second line contains the actual sequence, composed of nucleotides or amino acids. This sequence can span over several lines, depending on the length of the sequence (Figure 3).

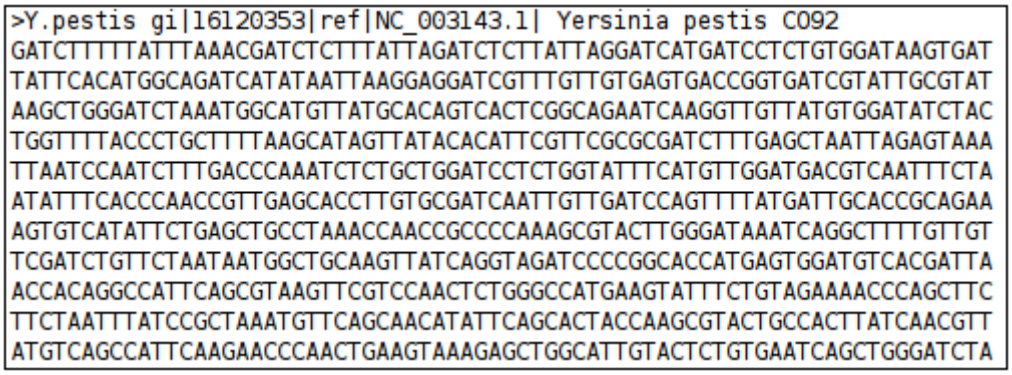

Figure 3: One record holding one sequence in FASTA format;

The format of the sequence identifiers is defined by the National Center of Biotechnology Information (NCBI).

SRA

The SRA file format was introduced as one of the International Nucleotide Sequence Databases Collaboration (INSDC) policies [26]. It is one of the currently supported file formats at the Sequence Read Archives (SRA) at NCBI, and the European Bioinformatics Institute (EBI). It supports data obtained from most current sequencing platforms: Roche's 454, Illumina's Genome Analyzer, Life Technologies' SOLiD, Helicos' Heliscope, Pacific Bio's PacBio, and Life Technologies' Ion Torrent. It performs a lossless compression and allows to

retrieve the original FASTQ files, including quality scores and sequence headers upon decompressing.

ZIP and GZIP Compressed Files

Compression algorithms didn't have much significance until the 1970s, when the first versions of the Internet became more popular. Although Huffman coding had already been invented in 1951 [27], the need to compress digital data wasn't justified until people started sharing more and larger amounts of data over the Internet. In 1977, Abraham Lempel and Jacob Ziv were pioneers in digital data compression, and they published their revolutionary LZ77 algorithm, which they refined again one year later [28]. Ever since then, Huffman Codes and Lempel-Ziv compression techniques have been the base for many compression algorithms or modifications and alterations thereof.

Huffman Codes

Huffman Code Algorithms take blocks of input characters of fixed length and produce output blocks of variable length. This algorithm is based on the probability of the appearances of short substrings. Short substrings are assigned to input blocks with high probabilities, and long substrings to those with low probabilities. This concept is similar to the Morse code. Table 3 and Figure 4 illustrate a simple example. A random sample text over the alphabet {A, B, C, D, E} is analyzed and the frequencies of each symbol present in the text are recorded.

Table 3: Frequencies for different symbols;

Symbol	Frequency
A	23
B	11
C	9
D	6
E	6

A tree is constructed according to the following scheme: First the two least common symbols are connected, which are 'D' and 'E'. Next, 'C' and 'B' are connected. The two resulting parent nodes are assigned the values 12 and 20, respectively. Those values are the sum of the frequencies of their child nodes. They are then connected in the next step to form another parent node with the value 32. In the final step, 'A' is added to the tree producing the root node with the value 55, which is the sum of all frequencies of all symbols. Each link branching to the right is assigned the value 1, and 0 is assigned to the ones branching to the left (Figure 4).

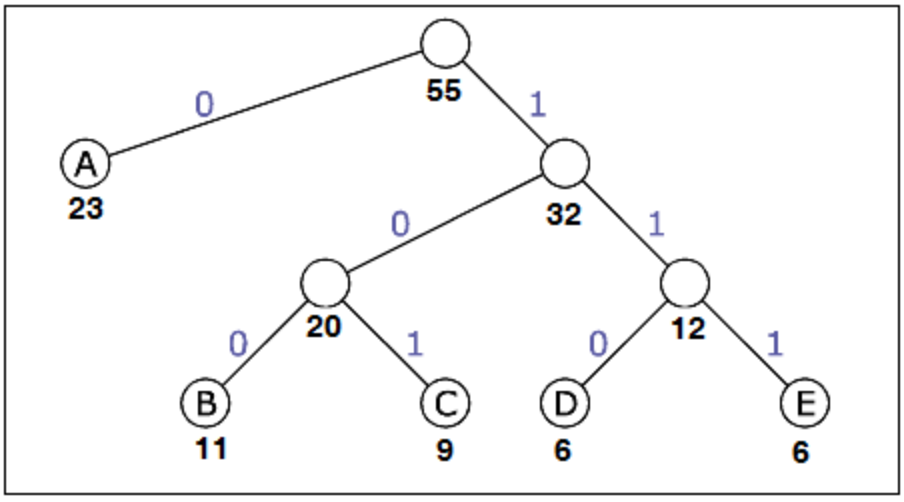

Figure 4: The resulting tree from the first steps in Huffman Coding;

Next, the code table is generated (Table 4). Starting from the root, each character's code corresponds to the sequence of 0s and 1s needed to reach the symbol while going through the tree. The code length corresponds to the depth of the node within the tree. The total length equals the symbol's frequency multiplied by its code length.

Table 4: Encoding table for the symbols present in the sample text;

Symbol	Frequency	Code	Code Length	Total Length
A	23	0	1	23
B	11	100	3	33
C	9	101	3	27
D	6	110	3	18
E	6	111	3	18

Using this table, each character can be encoded using a binary representation. The random sample text can be encoded using a total number of 119 bits, while the same text saved as a conventional text file (1 byte per character) would require 220 bits. The decoding works as follows: Starting from the root, the encoded bit-stream is passed through the constructed tree. Every time a node is encountered that does not have a child leaf, the appropriate symbol of that node is looked up in the table using the binary sequence gone though so far. The next decoding step starts at the root node again.

Huffman coding finds application in many data compression algorithms and formats, most worthy of mention in multimedia codecs such as JPEG [29] and MP3 [30].

Lempel Ziv Coding

The Lempel Ziv compression algorithm (LZ77) divides the input string into non overlapping blocks of different lengths, recording all encountered blocks in a dictionary of finite size. The algorithm has five distinct steps:

1. Dictionary initialization with all blocks of length 1
2. Look for the longest block B that has appeared in the dictionary
3. Encode B according to its dictionary index
4. Add B followed by the first character from the next block to the dictionary
5. Go back to Step 2 unless the end of the input string is reached

A simple example consisting of only two symbols {a, b} is shown in Table 5, using the dictionary shown in Table 6.

Table 5: The input string showing the corresponding value for each block;

Input String	a	b	b	a	a	b	b	a	a	b	a	b	b	a	a	a	a	b	a	a	b	b	a
Dictionary Index	0	1	1	0	2		4		2		6			5		5		7			3		0

Table 6: The dictionary created for the given input string;

Index	Entry
0	a
1	b
2	ab
3	bb
4	ba
5	aa
6	abb
7	baa
8	aba
9	abba
10	aaa
11	aab
12	baab
13	bba

Theoretically, an infinitely large dictionary is possible. In practice however, a maximum dictionary size must be defined. No additional words may be added to the dictionary once the limit is reached. Instead, the remaining input string is being encoded using the largest available words already present in the dictionary in each step. It is very obvious that this technique works significantly better for longer input strings than for shorter input strings. Decoding follows the very straight forward lookup process of the indices in the dictionary.

The most popular LZ77 application is the DEFLATE method [31], which combines LZ77 with Huffman Coding and finds use in various compression techniques like ZIP, GZIP, or PNG image files [32].

Burrows Wheeler

The Burrow Wheeler transform is not a compression algorithm in the conventional meaning. None of the characters present in the input change their values or are replaced, but instead a permutation of the order of characters is performed [33]. The advantage of this technique is that substrings with high repetition rates will be rearranged to occur next to each other. This circumstance can then be taken advantage of by another compression algorithm to actually reduce the size of the data [34]. Since the beginnings of high-throughput sequencing (HTS), the Burrows Wheeler transform has found many applications in bioinformatics, especially in alignment programs such as Bowtie [35], BWA, and SOAP2 [36]. It allows to greatly reduce the memory requirement.

Assume the sample input text: 'BACABBA'. The algorithm sorts all possible rotations of the input text in lexicographic order (Table 7).

Table 7: All rotations of the sample text, alphabetically sorted with the last character highlighted in bold font;

Index	**All rotations:**	**Alphabetically sorted:**
0	BACABBA	ABACAB**B**
1	ABACABB	ABBABA**C**
2	BABACAB	ACABBA**B**
3	BBABACA	BABACA**B**
4	ABBABAC	BACABB**A**
5	CABBABA	BBABAC**A**
6	ACABBAB	CABBAB**A**

The algorithm takes the very last character of each alphabetically sorted string to produce the output sequence: 'BCBBAAA'. The only additional information that needs to be saved is the index number of the row, in which the original sequence occurs, (4 in this case). Using this number, the original sequence can be deciphered. The decoding process works as follows: The encoded word is understood as the last column of the sorted table, as seen above. Alphabetically sorting the permutations of the encoded word will yield the first column of the table. The combination of the first and last columns of the table will yield all pairs of successive characters from the input (in a cyclical manner, so that the last and first character form a pair). Alphabetically sorting the pairs yields the first and second column. Following this method, the entire table can be reconstructed. There are two ways to find the original input sequence in this table. First, if the index number was saved, it can easily be looked up in the row of the given value. The second option is to put a unique character at the end of the sequence, before performing the Burrows Wheeler Transform. This unique character will appear at the very end of only one sequence in the reconstructed table, which will be the decoded input sequence.

2.3 Proposed Approach

The proposed approach specifically targets the challenges and problems discussed in chapter 2.1, and we present a time and memory efficient solution. We eliminate unnecessary data from the data sets, and keep only information that is known to be of value for further analysis. By removing duplicated

sequences, the redundancy is minimized. We keep track of the total number of copies of each sequence in a separate copy number array. We filter out sequences that do not pass a predefined quality threshold, which allows to discard all quality scores. Sequence headers are removed as well. Finally, after alphabetically sorting the nucleotide sequences, we can identify common prefixes among sequences, which can be substituted by single integer values, stored in a parallel array. This new data structure can be saved either as a text file, or as a binary file, and operates on both, sequences of equal length and sequences of variable length. The binary file format utilizes additional techniques and methods to minimize the space needed for a lossless sequence compression.

2.4 Design and Implementation

In this chapter we present the techniques and methods that were designed and developed to address the previously discussed problems. Our algorithms can be applied to sequencing data composed of sets of reads of equal length (Illumina reads), as well as sets of reads of variable length (reads from 454, Ion Torrent, SOLiD).

We have developed various functions to prepare the data for the actual compression steps. First, we added an integer array parallel to the sequence array, which holds the copy numbers for each sequence. This allows us to exclude duplicate sequences within the data set by simply incrementing the sequence's copy number for each duplicate that is present.

The next important step is to sort the sequences alphabetically. Much thought and many experiments were conducted to find the most appropriate sorting algorithm for this purpose. Eventually, we decided to implement a Most Significant Digit (MSD) Radix Sort [37]. Although a Least Significant Digit (LSD) Radix Sort might be easier to implement, the MSD version does come with some considerable advantages. While the LSD implementation starts at the position that makes the least difference, in the MSD algorithm we start with the position that makes the most difference. We divide all of the characters with an equal value into their own bucket, and continue to do the same thing with all buckets until the array is sorted. The recursive implementation allows performing the sort without the need to examine every single character in every single read. As soon as a difference in a more significant position is encountered, the sequences are separated into new buckets. This means that unique sequences residing in their own bucket do not have to be examined all the way to the end of the sequence. This saves a lot of time, especially if a very large amount of sequences is present in the data set.

Using these two techniques one can save the data in the so called AS, or ASF file format, which stands for *Array Subsequences* and *Array Sequences Flexible*, respectively. Both file formats start with an entry for the total number of sequences present in the data set. The AS format follows up with the read length of all sequences. Next, both formats list all sequences that are present in the following way: Each row begins with the actual sequence. If the sequence is unique, the copy number is not saved, and the next row shows the next

sequence. If the sequence is not unique, the copy number follows the sequence after a TAB key ('\t'). Then the next sequence follows in the next row (Figure 5).

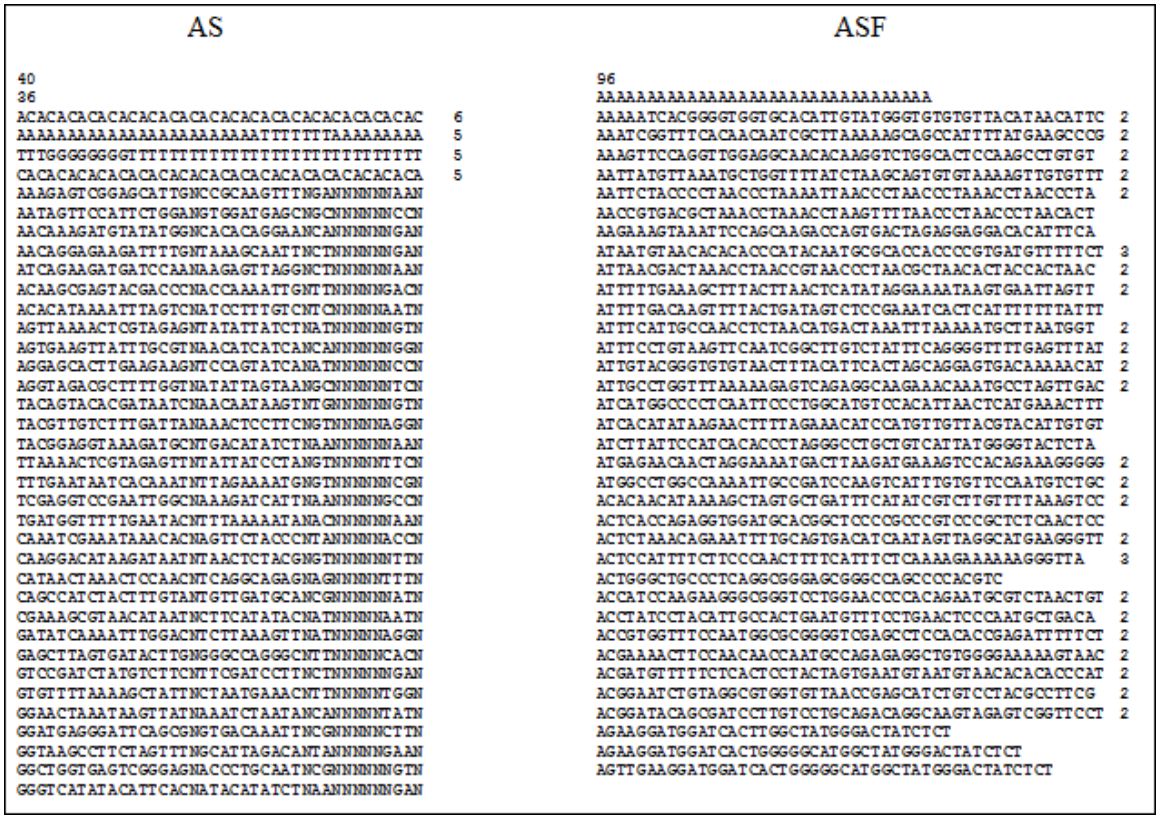

Figure 5: The AS (left) and the ASF format (right) with increased copy numbers for duplicate sequences, alphabetically sorted;

The next technique we applied to the sequences leads to the so called ASC or ASFC format, where the 'C' stands for "compressed". DNA sequences often share a common prefix, which can be present in hundreds of thousands of sequences. In order to reduce the total number of characters that are present in the data set, we substitute prefix sequences with integer numbers, which indicate the length of the prefix in respect to the previous sequence. The two file formats for the ASC and ASFC files are very similar to the AS and ASF formats, with the

only difference that each row (except the one holding the first sequence) has its prefix with the preceding sequence removed. The ASC file format does not require storing the length of the prefixes, since all sequences are of equal length. The ASFC file format has each row starting with the integer number representing the prefix length, and the only characters that are saved to the file are the characters producing the suffixes for the sequences (Figure 6).

AS

```
40
54
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGATCGATGAGCTAGAGCTAGCG 18
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGATCGAGATATACGACTAGCTA 9
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGATAGCGAGGCGAGAGCGGGCG 21
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGACCGAGCGGCGGCCCGAGCGA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGAGAAAATCGAGGCGAGCGGAG
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGAGAGAGGCGGAAGACGATCGA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGTTATTACTATCTATTTACTTA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGTCCCAGACTAGCTACGATCGA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGTGACTAGCTAGCTATCGAGCT 2
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGCCCAGATGAGCTAGGAGCTAG
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGCGAAGCACTATGAGGCTAGGA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGCGAAGGGGAGCTAGAGCTAGA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGCGATACTAGGAGCTTAGGCAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGCAAGGAGCATGCATTTTATTA 11
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGCCCCGATATTAAACGAGGCAG 2
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGCGAAAGCTCGGGGGCGAGACT
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGCGGACTAGAGCTAGCATCGAG
AAAAAAAAAAAAAAAAATATACTGACGATCAGCTAGCGATCGACTAGCGATCGA
AAAAAAAAAAAAAAAACGCTAGCATCAGCGATCAGGGGATAGCGATCAGGACT
AAAAAAAAAAAAAAAAGACTAGCATCAGAAAAAAAAAGCGACTAGCATGCTAG
AAAAAAAAAAAAAAAAGTAACATCGACATCAGCATGCACTAGCATGACTAGCG 66
AAAAAAAAAAAAAAAAGCCTACGACTAGCATCGACTAGCATGCATGACTGACT 3
AAAAAAAAAAAAAAAAGCCTAGCATCGACTAGCATGCATGCATCGATGACTAG
AAAAAAAAAAAAAAAAGGCTAGCATCGACTAGCATCGATGACTACGATCGATA
AAAAAAAAAAAAAAAACGACTAATCGAGGAGAGACTGACATCGATCAGCATAA 11
AAAAAAAAAAAAAAAAACGACTTGCATGAGAGCTAGCATAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAGGAGCTTTTTTTTTTTTTAGCATCGACTAGCAGCTA
```

ASC

```
40
54
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGATCGATGAGCTAGAGCTAGCG 18
GATATACGACTAGCTA 9
AGCGAGGCGAGAGCGGGCG 21
CCGAGCGGCGGCCCGAGCGA
GAAAATCGAGGCGAGCGGAG
GAGGCGGAAGACGATCGA
TTATTACTATCTATTTACTTA
CCCAGACTAGCTACGATCGA
GACTAGCTAGCTATCGAGCT 2
CCCAGATGAGCTAGGAGCTAG
GAAGCACTATGAGGCTAGGA
GGGAGCTAGAGCTAGA
TACTAGGAGCTTAGGCAA
GGAGCATGCATTTTATTA 11
CCCGATATTAAACGAGGCAG 2
GAAAGCTCGGGGGCGAGACT
GACTAGAGCTAGCATCGAG
TATACTGACGATCAGCTAGCGATCGACTAGCGATCGA
CGCTAGCATCAGCGATCAGGGGATAGCGATCAGGACT
GACTAGCATCAGAAAAAAAAAGCGACTAGCATGCTAG
GTAACATCGACATCAGCATGCACTAGCATGACTAGCG 66
GCCTACGACTAGCATCGACTAGCATGCATGACTGACT 3
GCATCGACTAGCATGCATGCATCGATGACTAG
GCTAGCATCGACTAGCATCGATGACTACGATCGATA
CGACTAATCGAGGAGAGACTGACATCGATCAGCATAA 11
TGCATGAGAGCTAGCATAAAAAAAAAAAAAAA
GGAGCTTTTTTTTTTTTTAGCATCGACTAGCAGCTA
```

ASF

```
40
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGATCGATGAGCTAGAGCTAGCG 18
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGATCGAGATATACGGCTA 9
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGATGGCGAGGCGAGAGAACGGCG 21
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGACCGAGCGGCGGCCCGAGCGA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGACGAAATCGAGGCCGGAG
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGACGGAGGCGGAAGACGATCGA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGAGATTACTATCTATTACTTA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGAGCCAGACTCGATCGA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGAGGATAGGAGCT 2
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGTCCAGATGAGCTAGGAGCTAG
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGTGAAGGACTATGGCTAGGA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGTGGAGCGGAGCTAGA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGCAGAACTAGGAGAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGGTAAAGGAGCATGCATTTATTA 11
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGGGACCCGATATTAAAGGCAG 2
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGGCAAAAAGCTCGGGGAGACT
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATCCGAAGACTAGAGCTAGCATCGAG
AAAAAAAAAAAAAAAATATACTGACGATCAGCTAGCGATCGACTGATCGA
AAAAAAAAAAAAAAAATGCTAGCATCAGCGATCAGGGGATAGCGATCAGGACT
AAAAAAAAAAAAAAAAGACTAGCATCAGAAAAAACATGCTAG
AAAAAAAAAAAAAAAACAAACATCGACATCAGCATGCACTAGCATGACTAGCG 66
AAAAAAAAAAAAAAAACGCTACGACTAGCATCGACTAGCATGCCTGACT 3
AAAAAAAAAAAAAAAATAGCTAGCATCGACTAGCATGCATGCATCGATGACTAG
AAAAAAAAAAAAAAAAGGACTAGCATCGACTAGCATCGATGACTGATCGATA
AAAAAAAAAAAAAAAACAGACTGATCGAGGAGAGACTGACATCGAAGCATAA 11
AAAAAAAAAAAAAAAATAAGACTAGCATGAGAGCTAGCATAAAAAAAAA
AAAAAAAAAAAAAACAACGAGCTTTTTTTTTTTAGCAGCTA
```

ASFC

```
40
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAATGCGATCGATGAGCTAGAGCTAGCG 18
38 GATATACGGCTA 9
35 GGCGAGGCGAGAGAACGGCG 21
34 CCGAGCGGCGGCCCGAGCGA
35 GAAATCGAGGCCGGAG
35 GGAGGCGGAAGACGATCGA
34 GATTACTATCTATTACTTA
35 CCAGACTCGATCGA
35 GATAGGAGCT2
33 TCCAGATGAGCTAGGAGCTAG
34 GAAGGACTATGGCTAGGA
35 GAGCGGAGCTAGA
33 CAGAACTAGGAGAA
31 GTAAAGGAGCATGCATTTATTA 11
32 GACCCGATATTAAAGGCAG 2
32 CAAAAAGCTCGGGGAGACT
30 CCGAAGACTAGAGCTAGCATCGAG
17 TATACTGACGATCAGCTAGCGATCGACTGATCGA
18 GCTAGCATCAGCGATCAGGGGATAGCGATCAGGACT
17 GACTAGCATCAGAAAAAACATGCTAG
17 CAAACATCGACATCAGCATGCACTAGCATGACTAGCG 66
17 CGCTACGACTAGCATCGACTAGCATGCCTGACT 3
16 TAGCTAGCATCGACTAGCATGCATGCATCGATGACTAG
16 GGACTAGCATCGACTAGCATCGATGACTGATCGATA
16 CAGACTGATCGAGGAGAGACTGACATCGAAGCATAA 11
15 TAAGACTAGCATGAGAGCTAGCATAAAAAAAAA
14 CAACGAGCTTTTTTTTTTTAGCAGCTA
```

Figure 6: The AS and ASF formats (left), and the ASC and ASFC formats (right) format showing the removal of prefixes for each sequence;

This format already shows a great potential to save on space while at the same time it remains readable to human eyes.

The compression resulting in the most significant reduction of file sizes is achieved using a binary file format to store the NGS data. The general file structure remains very similar for the so called ASCB and ASFB file formats, which contain reads of equal and flexible length, respectively. That is, first the total number of unique sequences is saved to the binary file as an unsigned integer value, followed by the sequence length in case of the ASCB file format. Then, prefix lengths, copy numbers, and suffix lengths (for ASFB), followed by the suffix sequences that represent the differences between sequential reads, are stored in binary format (Figure 7).

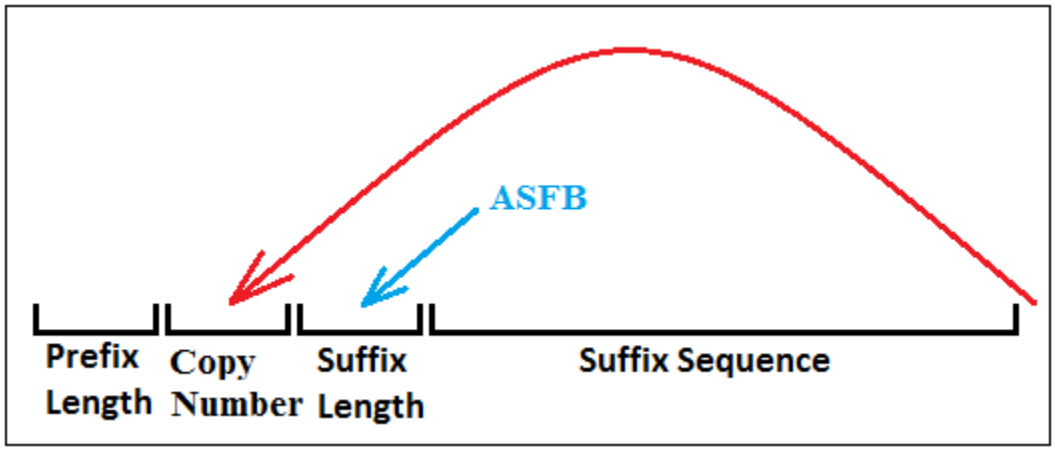

Figure 7: The design of the ASCB and ASFB formats; in contrary to the AS, ASF, ASC, and ASFC formats, the copy numbers are saved before the nucleotide sequences;

The algorithm takes advantage of the small size of the alphabet, NGS data is composed of. Instead of using one byte per character, as it is done when saving text files, we can fit four characters into one byte using the simple binary representation of nucleotides in a base-4 counting system (Table 8):

Table 8: Binary representation of the nucleotides A, T, C, G

A	00
T	01
C	10
G	11

This assumes that no unknown characters ('N') are present in the current data set, hence reads containing said characters have been excluded.

Depending on whether unknown nucleotides represented with the character 'N' are excluded or not, a base-4 or a base-5 counting system is used to transform blocks of nucleotides into integer values of the appropriate counting system. One so called block is the maximum amount of nucleotides the largest possible integer variable in C++ (unsigned long long int) can hold in its 8 bytes. Using the base-4 counting system, we can fit exactly 32 characters into such a variable, without losing any space to overhead. If reads containing unknown nucleotides have to be saved, the base-5 counting system allows a maximum of 27 characters to be saved in the 8 byte variable. This means each block also produces a very small overhead. For each sequence we first save one byte containing the following information:

The first bit of the byte indicates whether the copy number of the read equals one or not (0=false, 1=true). The second bit indicates whether the prefix length is less than 64 (0=false, 1=true). The remaining 6 bits are used to store the length of the prefix, as long as it is not more than 63 characters long (63 is the largest number one can represent using 6 bits, Figure 8).

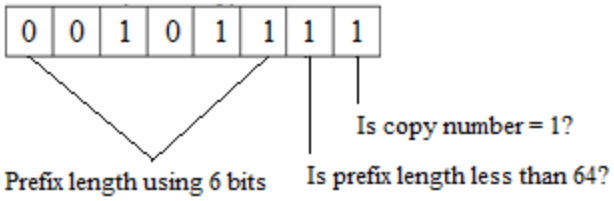

Figure 8: The first byte holding information about the Copy number and the prefix length;

If the prefix is longer than 63, an additional byte is used to store the prefix length, effectively giving this number 14 bits to allow a maximum value of 16,347 for the prefix (Figure 9).

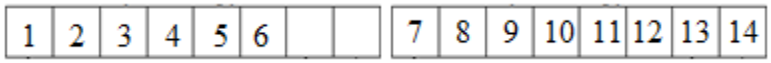

Figure 9: If 6 bits are not enough to store the length of the prefix, an additional byte is saved right after the first one, effectively giving the prefix length a maximum of 14 bits to be stored;

If the first bit indicated the copy number equals 1, meaning the sequence is unique, no additional byte to store the copy number must be saved to the file. If said bit was set to zero, meaning that there are several copies of the same sequence present, an additional byte will be saved next. This byte is divided as follows: the first bit indicates whether the copy number is less than 128. If this is true, the following 7 bits are used to store this copy number, allowing a maximum of 127 thereof. If the copy number exceeds this value, an additional two bytes are added, increasing the total number of bits to 23 (Figure 10). This allows the maximum value for the copy number to be 8,388,607. The reason behind adding two bytes instead of one is based on the presence of repeatable regions in the

sample. Most sequences are usually unique, meaning their copy number equals one. Then there are several sequences that are not unique, but have a few duplicates across the entire set of sequences. However, if a sequence is present in more than 127 copies, it is more than likely that this sequence originates from a repeatable region, and its copy number therefore is expected to also exceed the maximum value possible, if only one byte would have been added.

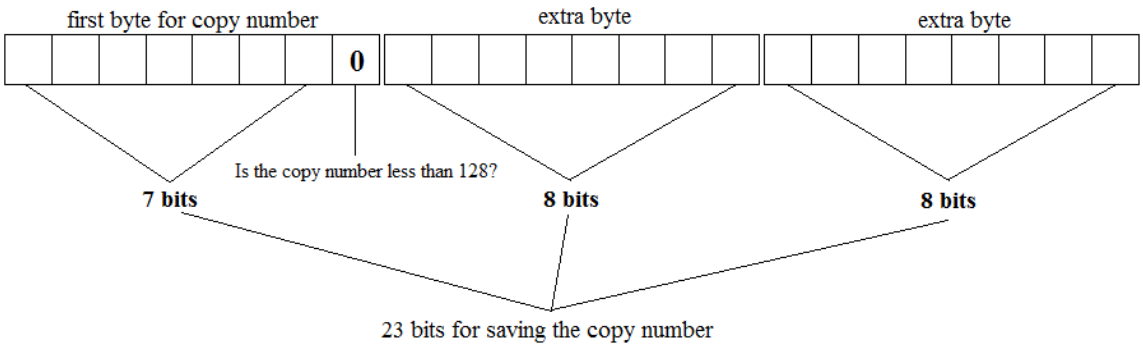

Figure 10: The first bite of the first byte for the copy number indicates that the copy number exceeds the value 127; two extra bytes are added, raising the new maximum for the copy number to 2^{23} - 1, which is 8,388,607;

Since all sequences are of the same length if they are produced with an Illumina instrument, no more additional information about the sequences has to be saved. However, if the reads are of different lengths, another byte is introduced before the actual sequences are saved to file. This additional byte contains on its first bit a one, if the suffix length is less than 128 and a zero otherwise. The following 7 bits are used to write the length of the suffix, as long as it does not exceed 128. If this limit is not enough, another byte is combined with the initial 7 bits to allow a maximum value of 32,767 for the suffix length (Figure 11).

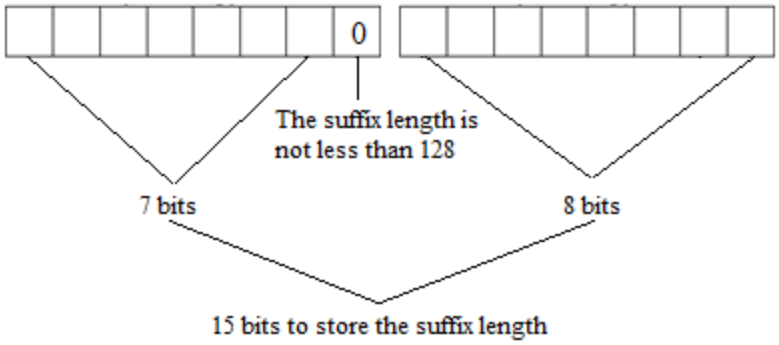

Figure 11: The first bit indicates the suffix length exceeds 127, hence an extra byte is added to increase the maximum suffix length to 2^{15} - 1, which is 32,767;

The algorithm then divides the suffix sequences into blocks. Depending on whether the base-4 or the base-5 counting system was used, the block size is either 32, or 27, respectively. This means every 32 or 27 characters are converted into an integer number in the appropriate counting system. This number, saved into an unsigned long long integer variable, is then written to the binary file using 8 bytes. In order to reduce the amount of overhead, the last block, which usually does not require 8 bytes to be saved, has a variable block size, which depends on the number of characters present in this last block. Following this scheme, the entire set of sequences of equal length is written to the binary file (Figure 12).

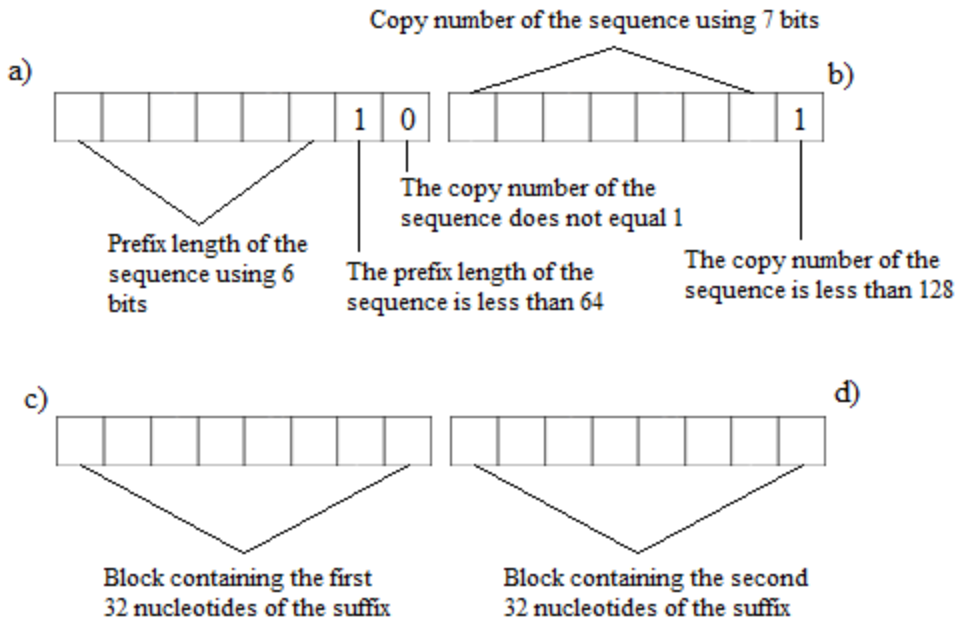

Figure 12: Visual representation of how the data for each individual sequence from an Illumina data set is stored in the binary file; the copy number in this example is bigger than 1 (0 bit in a)), but less than 127 (1 bit in b)); the prefix length is less than 64 (1 bit in a)); the first two blocks for the suffix sequence are depicted in the figure (c) and d)); there is no need to store the suffix length, since all sequences are of same length;

The file reading procedure can be seen as the reversed process of writing the binary file. After the number for the total amount of unique sequences is read, the sequence length is read in case the data set is composed only of sequences of equal length. Next, all the bytes that contain information about the sequences are read. The first byte contains the information about the prefix length, and whether the copy number of the sequence equals one. If necessary, additional bytes are read to calculate the total prefix length and the copy number. If the sequences are of variable length, an extra one or two bytes are read in order to retrieve the length of the suffix. The suffixes are extracted from the binary file the same way they are stored there: First, blocks of 8 bytes are read and stored in an unsigned

long long int variable. Those 8 bytes are then transformed back into 32 or 27 characters, depending on the underlying counting system. The total number of blocks can be determined by dividing the already read suffix length by the number of nucleotides that fit in one block. The last block can consist of less than 8 bytes in order to reduce the overhead. Once all blocks are successfully read, using the known prefix length, the entire sequence is reconstructed, and the first byte from the next record is read. This way, the original data set is rebuilt sequence by sequence.

The compression rates our approach is able to achieve are beyond the average compression rate one can expect. The analytical model is built as follows: Let *k* = *number of reads*, and let *n* = *length of the reads*; the total number of nucleotides is then *k* * *n*. In an ASC file, the number of possible nucleotides at position 1 is 4, and 16 at position 2, etc.; thus the number of nucleotides that are not being saved can hence be seen as k-4, k-16, etc. for position 1 and 2, respectively. A generalization of the model is shown in the following formula, which returns the number of nucleotides that are not stored in the ASC format:

$$\sum_{l=1}^{x} k - 4^{l}$$

Assuming $x = \max(k - 4^{l}) > 0$;

The number of nucleotides that need to be stored is therefore:

$$(k * n) - \sum_{l=1}^{x} k - 4^l$$

We have performed many experiments on both random data, as well as actual DNA data from sequencing instruments. While the results using the randomly generated data very closely resemble what the analytical model predicts, the experiments using real data showed very different results.

The file size reduction depends strongly on the properties of the sequencing data. Using the presented formats, the binary file sizes can be as low as 0.7% - 5% of the original FASTQ file. We compared our results with compression techniques that are commonly used in the bioinformatics and genomics industry: ZIP, GZIP, and SRA.

Important to mention is that especially in old data sets, many reads are of very poor quality. Our algorithm also checks the quality for each individual nucleotide, and excludes reads which do not pass the minimum quality threshold, which is set to 10 by default. This led to another important observation: Especially databases like the Small Reads Archive have the potential to massively free up space, if they were to only accept and store reads of satisfactory quality.

The time and space complexity of the sorting algorithm we implemented is linear, i.e. O(n). This implies that the file conversion time is linearly dependent only on the total number of reads.

We have chosen four different data sets to illustrate the advantages of our data compression over other techniques. The four data sets contain reads from:

1. Bacteria
2. Human Transcriptome
3. Human Genome
4. Groudwater Metagenome

Generally, converting the original FASTQ files into the SRA file format yields just a slightly better result than a ZIP or GZIP compression produces. In all cases, the binary compressed file we produced outperformed all other methods by a large margin, even if no low quality reads were excluded. A second comparison which also included the exclusion of low quality reads resulted in an even more significant difference (Table 9).

Table 9: Our four data sets compared in file sizes (MB) using different file formats;

Data Set	SRA	zip	gzip	FASTQ	FASTA	All reads			Low quality reads excluded		
						AS	ASC	ASCB	AS	ASC	ASCB
Bacteria	726	910	910	2,819	1,246	1,055	864	271	871	718	191
Human transcriptome	2,237	2,786	2,786	14,520	5,814	293	158	59	141	99	30
Human genome	14,301	20,916	20,916	105,142	51,153	43,408	38,548	11,870	15340	13,885	3,635
Groundwater metagenome	4,888	6,566	6,566	20,902	9,264	7,578	6,901	2,134	4670	4,286	1,130

If we take the original FASTQ file sizes as references, we can observe that both ZIP and GZIP compression tend to yield files in the range of 19% - 30% of the original file sizes. The SRA file conversion delivers files ranging from 15% - 30%, depending on the composition of the DNA sequences (Figure 13).

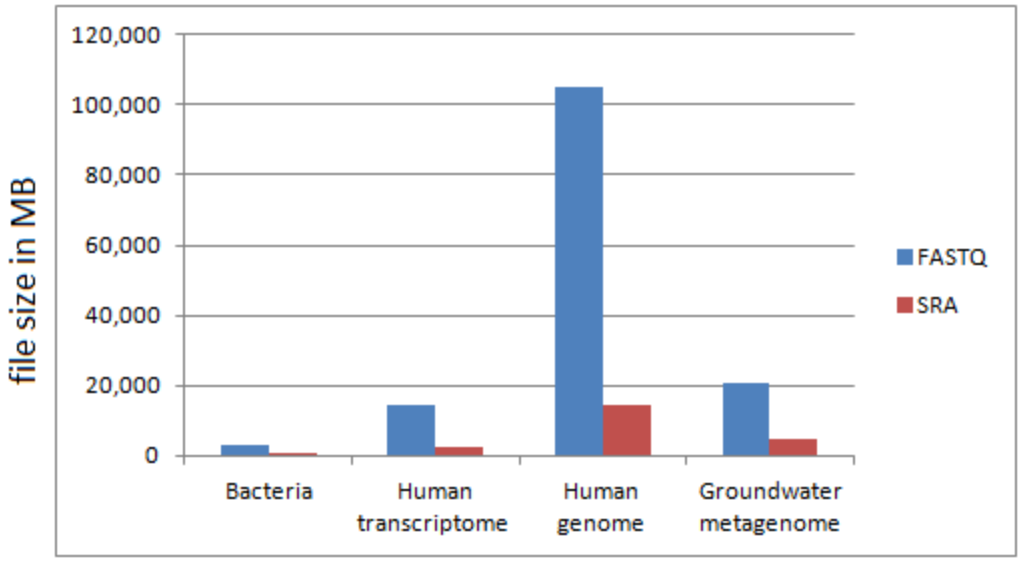

Figure 13: FASTQ to SRA compression of the four test files;

The compressed binary files of the nucleotide sequences take up only 10% - 0.4% of the original FASTQ file sizes. Especially data sets that contain many repeated sequences have the potential to produce a significantly smaller binary file. If many sequences share a long common prefix, even smaller file sizes can be achieved. Since nucleotides with very low quality scores should never be included in any type of bioinformatics analysis, we also compared the original files to the data sets, which resulted after excluding low quality reads. The file sizes again decreased significantly for data sets that contained many such low quality reads. For example, while the binary file of the human genome sample was almost 12 Gigabytes in size, the cleaned data set was only 3.6 Gigabytes, i.e. about 70% smaller. A direct comparison of how much the cleaning of the data sets yields can be seen in Figure 14.

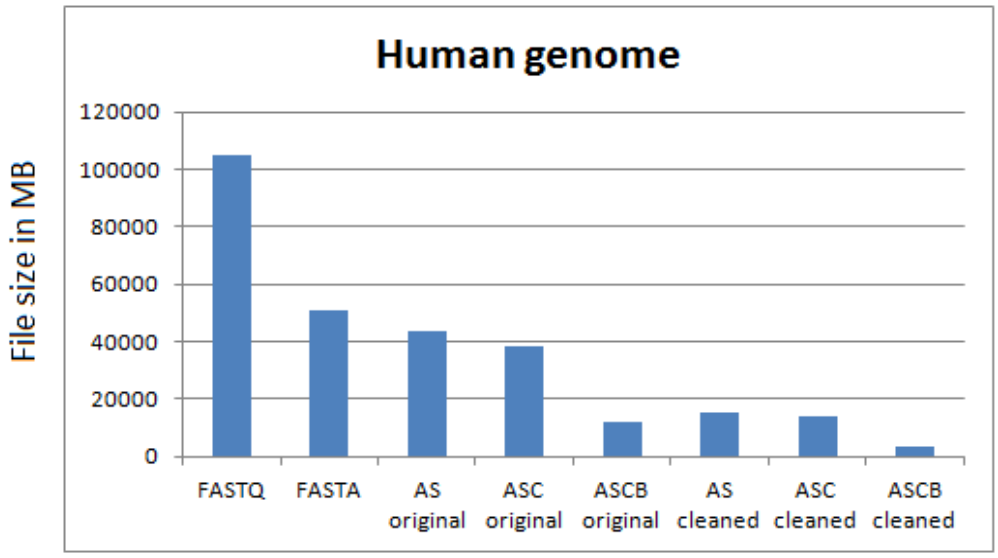

Figure 14: Human Genome data set, depicting the reduction of file sizes when cleaning the data set from duplicate sequences and sequences with quality scores below the threshold 10;

Not only do we compare our approach in terms of resulting file sizes. An almost equally important aspect that needs to be considered is the amount of time such file conversions take to efficiently convert files from one format to another. The sorting algorithms we implemented are a variation of a most-significant digit Radix Sort in combination with a Counting Sort. These algorithms run in linear time, i.e. O(n), where *n* is the total number of objects to be sorted. This benefits the total runtime enormously. In fact, especially for very large files, the main bottleneck in time complexity drifted from execution and calculations to I/O.

We again compared the times needed to perform various file conversions. Not only did we look at the time it takes to decompress SRA files into the common FASTQ format, but we also examined the times to compress and decompress

files using the GZIP approach. We opposed these times to the results obtained from our implementation, which you can see in Tables 10, 11, and 12.

Table 10: Time comparisons for SRA and GZIP and our implemented file formats;

Data Set	SRA to FASTQ	FASTQ to .GZ	.GZ to FASTQ
Bacteria Genome	61.8	478.3	56.0
Human transcriptome	404.3	817.1	176.7
Human genome	7,901.3	8,366.6	1,141.4
Groundwater metagenome	347.2	3,322.7	257.0

Table 11: Time comparisons for our implemented file formats (all reads);

FASTQ to AS	FASTQ to ASC	AS to ASCB	ASCB to FASTA	ASCB to AS	ASCB to ASC	ASCB (read/write)
34.0	61.9	41.1	50.5	42.5	69.3	22.0/ 14.7
299.7	302.3	11.8	157.4	14.3	20.4	4.7/3.4
1,434.5	2,670.3	1,632.4	1,962.7	1,692.3	1,791.2	1,002.0/620.9
263.0	480.7	256.6	560.0	502.5	500.7	181.5/115.2

Table 12: Time comparisons for our implemented file formats (low quality reads excluded);

FASTQ to AS	FASTQ to ASC	AS to ASCB	ASCB to FASTA	ASCB to AS	ASCB to ASC	ASCB (read/write)
35.4	55.9	35.9	43.6	36.1	58.7	19.4/12.1
276.8	279.3	6.0	122.8	7.5	10.6	2.6/1.9
1,003.3	1,497.9	631.5	722.2	625.4	1,000.2	336.9/226.0
228.1	549.6	194.5	252.9	199.6	525.8	115.1/71.9

Figure 15 depicts the times needed using conventional file compression techniques to compress and decompress the NGS data, such as SRA and GZIP.

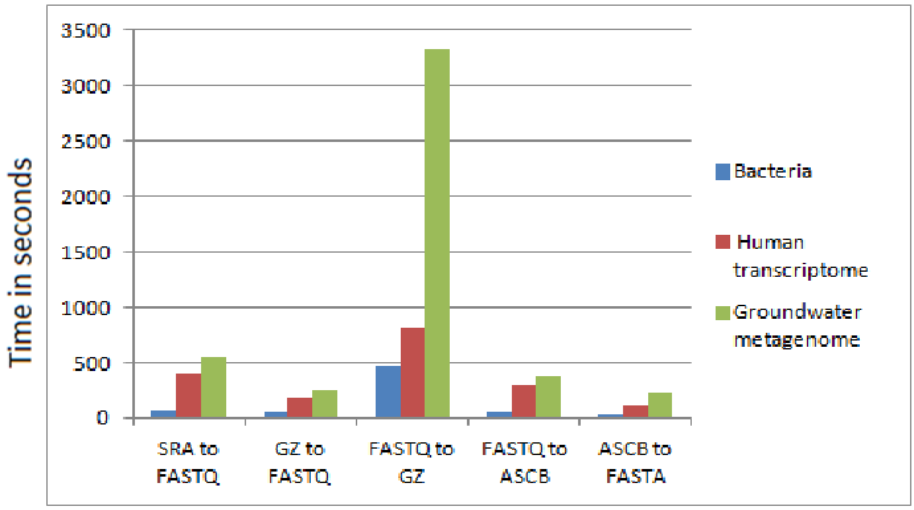

Figure 15: Times for compressing and decompressing data sets using ASCB, SRA, and GZ formats;

Clearly, the SRA file format shows by far the worst performance. We then recorded the times of our solution again in two different ways: First we did not exclude any reads from the data sets, but then we converted the files again and excluded all reads with quality scores below our set threshold of 10. The results indicate that the most significant differences are between conventional compression techniques and our approach, rather than between the two runs with and without the exclusion of sequences. This finding implies that our algorithms are generally faster and much more time efficient, and the total number of sequences plays a much less significant role. While for example the decompressing process of the human transcriptome SRA file to a usable FASTQ takes 404 seconds (i.e., 6.7 minutes), the same file can be read within 5 seconds

if it were compressed in our binary format. This would allow bioinformatics applications to read much smaller files into memory in much less time, while at the same time the file reading and decompressing is performed within the same process (as opposed to first decompressing a file using SRA / GZIP and then separately reading the file into memory using conventional application I/O).

All experiments were performed on a single computer running on 4xAMD Opteron 6300 Series, 2.8GHz 16 Core Socket G34 CPU, 512GB DDR3 1600Mhz E/R, 1TB SATA II Enterprise Hard Drive 3.0Gb/s, 7200 RPM.

2.5 Discussion and conclusion

The proposed algorithms and file structures are expected to have a huge impact on how HTS data will be managed in the future. For example, instead of having to download all the sequenced data from a metagenomics related project onto 100 hard drives, the same amount of information can now be saved on just three hard drives of the same type. This removes the necessity of large data storage devices such as RAID servers. Instead, the three hard drives can directly be plugged the workstation the analysis is performed on.

This advantage in return enables the analysis of HTS data to be done on regular computers or workstations, instead of on big data clusters. It takes us closer to the goal of bringing analysis and data generation together.

3. Gene Cluster Approach to Reduce Redundancy of Reference Sequences for High Throughput Sequencing Data Analysis

3.1 Problem Description

Data redundancy is an issue many researchers have to deal with when obtaining or downloading data from public storages such as SRA or GenBank [38]. Often the sequencing data is outdated or very old. The quality of the sequences produced several years ago is not nearly close to today's standards, and thus the sequencing data itself many times is of poor quality as well. Many data sets are incomplete, are missing annotations, or contain sequences that are too short to be used. The redundancy lies not only in the sequences though. A large portion of unnecessary data resides in the headers of FASTQ and FASTA files. Many times the header lines are much longer than the actual nucleotide sequences, thus reducing the total amount of useful information in a FASTA or FASTQ file far below 50% of its total file size. Additionally, in several cases the headers are even several magnitudes greater than the actual sequences (Figure 16).

>gi|30262378|ref|NP_844755.1| mbtH-like protein [Bacillus anthracis str. Ames]gi|47527668|ref|YP_019017.1| balhimycin biosynthetic protein MbtH [Bacillus anthracis str. 'Ames Ancestor']gi|49185218|ref|YP_028470.1| balhimycin biosynthetic protein MbtH [Bacillus anthracis str. Sterne]gi|49479960|ref|YP_036475.1| balhimycin biosynthetic protein MbtH [Bacillus thuringiensis serovar konkukian str. 97-27]gi|65319670|ref|ZP_00392629.1| COG3251: Uncharacterized protein conserved in bacteria [Bacillus anthracis str. A2012]gi|118477774|ref|YP_894925.1| mbtH-like protein [Bacillus thuringiensis str. Al Hakam]gi|165870487|ref|ZP_02215141.1| mbtH-like protein [Bacillus anthracis str. A0488]gi|167632767|ref|ZP_02391093.1| mbtH-like protein [Bacillus anthracis str. A0442]gi|167639636|ref|ZP_02397906.1| mbtH-like protein [Bacillus anthracis str. A0193]gi|170686910|ref|ZP_02878129.1| mbtH-like protein [Bacillus anthracis str. A0465]gi|170706688|ref|ZP_02897147.1| mbtH-like protein [Bacillus anthracis str. A0389]gi|177649522|ref|ZP_02932524.1| mbtH-like protein [Bacillus anthracis str. A0174]gi|190565680|ref|ZP_03018600.1| mbtH-like protein [Bacillus anthracis str. Tsiankovskii-I]gi|196033814|ref|ZP_03101225.1| mbtH-like protein [Bacillus cereus W]gi|196039934|ref|ZP_03107237.1| MbtH-like protein [Bacillus cereus NVH0597-99]gi|196043531|ref|ZP_03110769.1| mbtH-like protein [Bacillus cereus 03BB108]gi|218903507|ref|YP_002451341.1| mbtH-like protein [Bacillus cereus AH820]gi|227814816|ref|YP_002814825.1| mbtH-like protein [Bacillus anthracis str. CDC 684]gi|229603791|ref|YP_002866710.1| mbtH-like protein [Bacillus anthracis str. A0248]gi|254684950|ref|ZP_05148810.1| mbtH-like protein [Bacillus anthracis str. CNEVA-9066]gi|254722357|ref|ZP_05184145.1| mbtH-like protein [Bacillus anthracis str. A1055]gi|254737398|ref|ZP_05195102.1| mbtH-like protein [Bacillus anthracis str. Western North America USA6153]gi|254743418|ref|ZP_05201103.1| mbtH-like protein [Bacillus anthracis str. Kruger B]gi|254751713|ref|ZP_05203750.1| mbtH-like protein [Bacillus anthracis str. Vollum]gi|254760232|ref|ZP_05212256.1| mbtH-like protein [Bacillus anthracis str. Australia 94]gi|301053892|ref|YP_003792103.1| balhimycin biosynthetic protein MbtH [Bacillus cereus biovar anthracis str. CI]gi|376266284|ref|YP_005118996.1| Polymyxin synthetase PmxB [Bacillus cereus F837/76]gi|386736125|ref|YP_006209306.1| MbtH-like protein [Bacillus anthracis str. H9401]gi|421510129|ref|ZP_15957027.1| MbtH-like protein [Bacillus anthracis str. UR-1]gi|421636180|ref|ZP_16076779.1| MbtH-like protein [Bacillus anthracis str. BF1]gi|30257009|gb|AAP26241.1| mbtH-like protein [Bacillus anthracis str. Ames]gi|47502816|gb|AAT31492.1| mbtH-like protein [Bacillus anthracis str. 'Ames Ancestor']gi|49179145|gb|AAT54521.1| mbtH-like protein [Bacillus anthracis str. Sterne]gi|49331516|gb|AAT62162.1| MbtH protein [Bacillus thuringiensis serovar konkukian str. 97-27]gi|118416999|gb|ABK85418.1| mbtH-like protein [Bacillus thuringiensis str. Al Hakam]gi|164713642|gb|EDR19165.1| mbtH-like protein [Bacillus anthracis str. A0488]gi|167512345|gb|EDR87721.1| mbtH-like protein [Bacillus anthracis str. A0193]gi|167531579|gb|EDR94244.1| mbtH-like protein [Bacillus anthracis str. A0442]gi|170128419|gb|EDS97287.1| mbtH-like protein [Bacillus anthracis str. A0389]gi|170668961|gb|EDT19705.1| mbtH-like protein [Bacillus anthracis str. A0465]gi|172084596|gb|EDT69654.1| mbtH-like protein [Bacillus anthracis str. A0174]gi|190563707|gb|EDV17672.1| mbtH-like protein [Bacillus anthracis str. Tsiankovskii-I]gi|195993494|gb|EDX57451.1| mbtH-like protein [Bacillus cereus W]gi|196025840|gb|EDX64509.1| mbtH-like protein [Bacillus cereus 03BB108]gi|196029193|gb|EDX67797.1| MbtH-like protein [Bacillus cereus NVH0597-99]gi|218538120|gb|ACK90518.1| mbtH-like protein [Bacillus cereus AH820]gi|227003692|gb|ACP13435.1| MbtH-like protein [Bacillus anthracis str. CDC 684]gi|229268199|gb|ACQ49836.1| mbtH-like protein [Bacillus anthracis str. A0248]gi|300376061|gb|ADK04965.1| MbtH-like protein [Bacillus cereus biovar anthracis str. CI]gi|364512084|gb|AEW55483.1| Polymyxin synthetase PmxB [Bacillus cereus F837/76]gi|384385977|gb|AFH83638.1| MbtH-like protein [Bacillus anthracis str. H9401]gi|401187706|gb|EJQ94779.1| hypothetical protein IGW_02499 [Bacillus cereus ISP3191]gi|401819842|gb|EJT19014.1| MbtH-like protein [Bacillus anthracis str. UR-1]gi|403396708|gb|EJY93945.1| MbtH-like protein [Bacillus anthracis str. BF1]

MTNPFENDNYTYKVLKNEEGQYSLWPAFLDVPIGWNVVHKEASRNDCLQYVENNWEDLNPKSNQVGKKILVGKR

Figure 16: 36 header lines for one amino acid sequence of length 74, as found in GenBank;

Another major issue is the redundancy of references in reference databases. Exact copies of same sequences might be present multiple times, each time with different annotation. Sequences with just minor variations between one another such as very few or even single SNPs are present in the hundreds of thousands. Finding and selecting appropriate references is therefore very difficult, sometimes even impossible. The interpretation of results hence is another

challenge, especially if many different reference sequences were involved in the analysis. For example, assuming one has a 6 Gigabases dataset, which is the resulting output from one NGS experiment. Searching for each individual sequence in a reference sequence data base would require an immense amount of time and resources.

Most analysis performed on HTS data, such as searching, mapping, or assembly, requires performing comparisons to already known data. This pool of known data can be tremendously huge, and algorithms with very high time complexities, most significantly sequence alignment algorithms, cannot efficiently be applied to this large amount of references.

3.2 Gene Cluster Database - Proposed approach

One of the possible solutions to this problem would be the creation of a gene cluster database [39]. In this approach gene sequences have to be clustered together using a certain degree of maximum dissimilarity. As long as two or more sequences are similar to each other within the boundaries of this limit, they form their own cluster and a *single representative sequence* is chosen to correspond to this new cluster. Applying this idea to the entire set of known reference gene sequences, many almost identical sequences can be eliminated. This clustering process can be performed for several dissimilarity thresholds. This allows creating different sets of clusters. Depending on the target application, the

smallest, most appropriate cluster can be chosen to serve as a reference data base. If the analysis requires a higher order of dissimilarity the cluster set can easily be exchanged with the one representing the next higher dissimilarity.

3.3 Large Scale Sequence Clustering Challenge

The biggest problem one would face trying to design and implement mentioned sequence clusters is to find the longest common subsequences within a set of references [40]. This can be done using a variety of different sequence alignment algorithms. Sequence Alignment is the primary technique used to identify regions of similarity among multiple DNA, RNA, or protein sequences. Those similarities usually imply an evolutionary or structural relation among the analyzed sequences. Alignment methods typically produce matrices, with the aligned sequences representing the rows thereof. Identical or similar residues ideally appear in the same column, enabled by displaying gaps as dashes ("-") (Figure 17).

```
AGCCTTGTCATCCGTATC-TTTCAA----
AGCCTTGTCATCCGTATC-TTTCA-----
-GCCTTGTCATCCGTATC-TTTCAACG--
--CCTTGTCATCCGTATC-TTTCAACGTG
--CCTTGTCATCCGTATC-TTTCAAC---
---CTTGTCATCCGTATCTTTTCAAC---
---CTTGTCATCCGTATC-T---------
----TTGTCATCCGTATC-TT--------
------GTCATCCGTATC-TTTCAACGTG
------GTCATCCGTATC-TTTCAACGTG
--------CATCCGTATC-TTTCAA----
--------CATCCGTATC-TTTCA-----
---------ATCCGTATC-TTTCAACGTG
```

Figure 17: Sequence alignment utilizing gap insertions ("-");

Sequence alignment applications result in certain alignment scores by summing up the scores for:

- Aligned pairs of letters
- Letters aligned with gaps
- Aligned mismatches [41]

The higher the alignment score, the more similar two or more sequences are said to be.

Sequence alignment can be performed locally [42] or globally [43]. Global sequence alignment requires that all letters and gaps in each sequence must be aligned. Hence global alignment works best for sequences of same or comparable length that are expected to be similar. Local sequence alignment works best on sequences of different lengths, which are expected to have certain regions of high similarity within the sequences. See Figure 18 for an example of global and local alignment, applied to the same sequences.

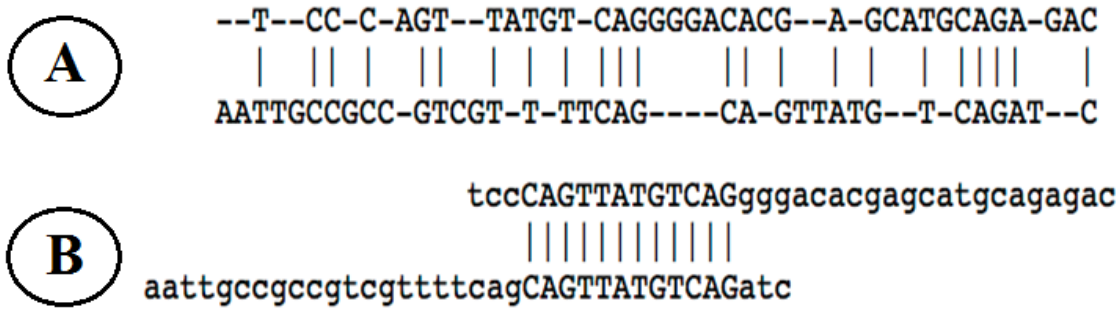

Figure 18: Global alignment (A) and local alignment (B) performed on the same set of sequences;

Another simple approach to visualize the results of alignment is the Dot-matrix [44]. Here, each sequence is placed on the edge of a matrix, the first one on the top row, the second one along the leftmost column of the matrix. A dot is printed at each location where the characters of the two sequences match. Regions of similarity are revealed by diagonal lines within the matrix, while isolated dots apart from mentioned lines represent random matches (Figure 19).

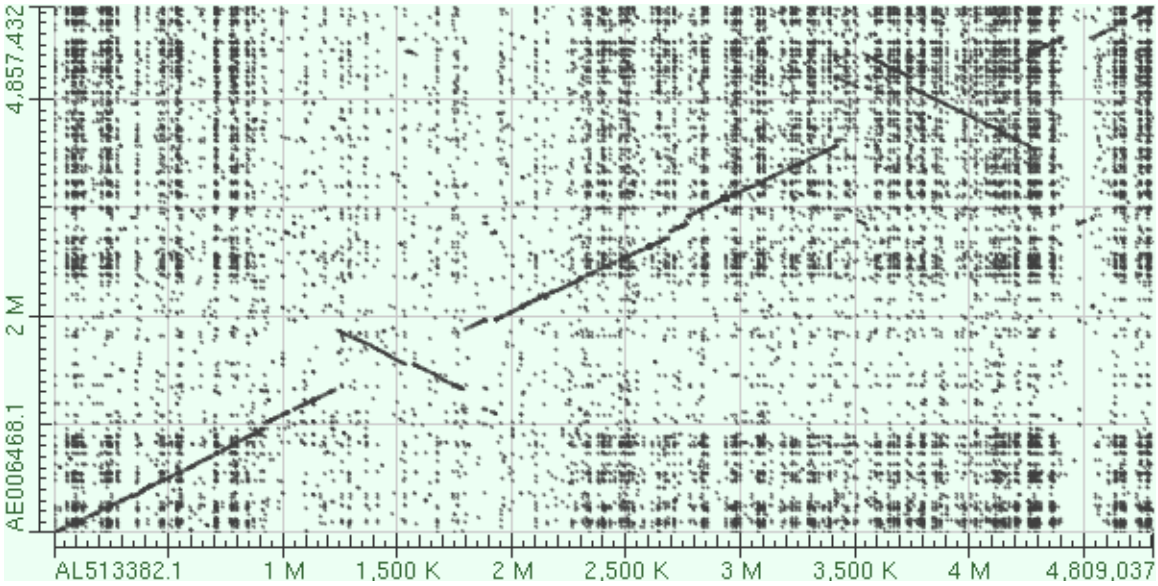

Figure 19: Dot-matrix for the complete genomes of Salmonella str. LT2 [45] vs. Salmonella str. CT18 [46];

There are three techniques commonly used for sequence alignment: The Needleman-Wunsch algorithm for global alignment [47], the Smith-Waterman algorithm for local alignment [48], and the Longest Common Subsequence (LCS) approach [40].

Needleman-Wunsch Algorithm

The Needleman Wunsch algorithm was designed to compute global alignment. It was introduced in 1970 by Saul Needleman and Christian Wunsch. The algorithm is an example of dynamic programming [49] and finds the best alignment by using optimal alignments of smaller subsequences. It's time complexity for two sequences that are both n letters long initially was n^3, but the algorithm soon was improved to only require n^2 [50]. At first, the scoring function σ for matches and mismatches, as well as the gap penalty must be defined. A simple scoring function would be +1 for matches and -1 for mismatches. Then, there are two steps that need to be performed in order to compute the best alignment:

First, fill a scoring matrix *T* according to the following function:

$$T\ (i,j) = max \begin{cases} T(i-1,j-1) + \ \sigma(S1(i), S2(j)) \\ \quad T(i-1,j) + \ gap\ penalty \\ \quad T(i,j-1) + \ gap\ penalty \end{cases}$$

S1 and *S2* are sequence 1 and sequence 2, respectively, while *i* and *j* represent the indices for the matrix and sequence positions. Secondly, use the completed matrix *T* to trace back and find the best alignment.

Smith-Waterman Algorithm

The Smith Waterman algorithm was designed to perform local alignment. Temple F. Smith and Michael S. Waterman first proposed the variation of the Needleman Wunsch algorithm in 1981. Just like the Needleman Wunsch algorithm, it is a

dynamic programming algorithm, and it guarantees to find the optimal local alignment considering a given scoring system. Unlike the Needleman Wunsch Algorithm, negative scores are not allowed, but are instead set to zero. The backtracking algorithm starts at the cell with the highest score and ends upon reaching a cell with a zero score, which produces the optimal local alignment for that location.

Longest Common Subsequence Algorithm

The LCS algorithm is also based on dynamic programming. The problem it aims to solve is as follows: Given two strings, X and Y, the goal is to produce the longest common subsequence, which appears left to right in both strings (but not necessarily in a contiguous block). For example, for the two strings:

- X = AEIOUZ
- Y = AOUSTZ

The LCS would be AOUZ. A naive solution would generate all subsequences for each of the given sequences using all present characters, and look for the longest subsequences appearing in both sets. This solution would be classified as NP-hard [51], and require an exponential time complexity, which is unacceptable especially for very long sequences. The remedy is found in dynamic programming, which allows solving the problem in polynomial time, as long as the number of sequences remains constant.

A solution to this problem cannot be heuristically approached, but must be found exhaustively instead. However, all techniques used to perform sequence alignment share the common problem of immense time, memory, and processing power needed, especially for very large sequences. To align two sequences of length *m* and *n*, O(*mn*) time is needed. In order to align a set of *k* sequences, the time complexity increases to O(*kmn*).

3.4 Basic Idea of Proposed Approach

The following process serves as a preliminary step for pair wise sequence alignment in order to exclude sequences that are too distant from each other from the process. First, Markov Chain profiles [52] [53] are created for all sequences present in the data set. This process is only dependent on the total number of sequences and possesses linear time complexity, because each profile has to be generated only once. Next follows the pair wise comparison of the newly created profiles, in order to find the distance between two sequences. For lower order Markov Chain profiles this can be done very fast, and allows identifying sequences, which are very distant from each other. One can then exclude the flagged sequences from any further alignment process, since it would not yield a satisfying result. Although this process requires additional time to execute, the overall time will be reduced due to the enormous time complexity of pair wise sequence alignment algorithms.

Markov Chain profiles are generally used to probabilistically model state transitions where the succeeding states only depend on *k* number of states. All states preceding the last *k* are irrelevant. In other words, a Markov Chain model of order 1 means that every next state is dependent solely on the current one. In a Markov Chain model of order 2, every state is dependent on the 2 immediate preceding sates. A transition matrix *P* can be created to illustrate the likelihoods to transition from one (current) state into each of the possible following states [53].

We can model our application the following way: We have 4 distinct states, each nucleotide being one of them: A, T, C, G. Each nucleotide has a 25% chance to be followed by any of the four nucleotides again. First we can create a transition graph (Figure 20).

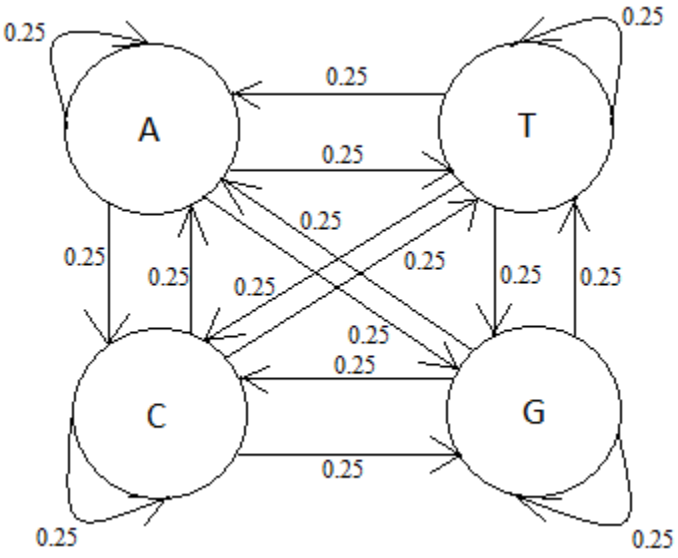

Figure 20: Transition Graph depicting the chances to change from one state (nucleotide) to another;

The transition matrix *P* created suing these four distinct states looks as follows:

$$P = \begin{bmatrix} 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \end{bmatrix}$$

If we are looking at the nucleotide Adenine, and we want to calculate the probability of encountering Thymine 2 locations away from the current, we need to proceed as follows:

Pr[A -> ? -> T] =

Pr[A -> A -> T] + Pr[A -> T -> T] + Pr[A -> C -> T] + Pr[A -> G -> T] =

0.25 * 0.25 + 0.25 * 0.25 + 0.25 * 0.25 + 0.25 * 0.25 = **0.25**

Now this might seem obvious assuming the probabilities never change, but instead remain 0.25 for each and every transition throughout the sequence. In reality however, these probabilities differ a lot and depend on the nature of the DNA sequence. In an Adenine / Thymine rich region for example, the chances to encounter an A or T are much higher than for C and G. For instance, if at the current position resides an A, the probabilities can state that with a 45% chance, the next nucleotide will also be an A, and a T with 35%. With a chance of only 15%, the current A is followed by a C, and a G with 5%. On the other hand, if the current position shows a C, with the chance of 55%, it is followed by an A, and a T with 30%, while with the chance of 5% it is followed by another C, and a G with

10%. The probabilities of transitions for current positions T and G can be seen in the following new transition graph:

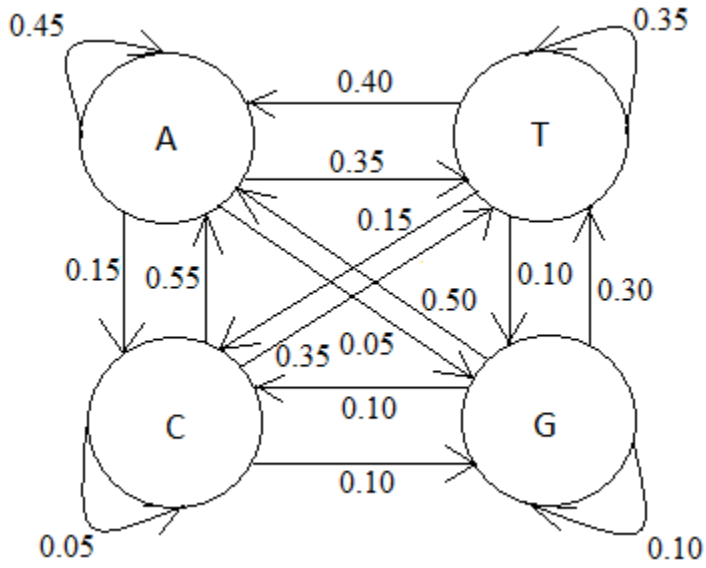

Figure 21: Possible transition graph for the new probabilities in an A/T rich region of the DNA sequence;

The new transition matrix looks as follows:

$$P = \begin{bmatrix} 0.45 & 0.35 & 0.15 & 0.05 \\ 0.40 & 0.35 & 0.15 & 0.10 \\ 0.55 & 0.35 & 0.05 & 0.10 \\ 0.50 & 0.30 & 0.10 & 0.10 \end{bmatrix}$$

Trying to answer the same question as before, namely the chance to encounter a Thymine 2 locations from the current location, which is an Adenine, yields the following result:

Pr[A -> ? -> T] =

Pr[A -> A -> T] + Pr[A -> T -> T] + Pr[A -> C -> T] + Pr[A -> G -> T] =

0.45 * 0.35 + 0.35 * 0.35 + 0.15 * 0.35 + 0.05 * 0.30 = **0.3475**

Not surprisingly, the value has increased. This way the likelihood of an arbitrarily far away nucleotide can be estimated using the shown Markov Chains.

Before the Markov Chain Profiles are created, we sort the sequences by length. This allows removing all sequences which due to their length differences do not allow successful alignment to begin with. For example, one sequence is of length 2000, a second sequence is of length 1000, and the maximum dissimilarity threshold is set to 5%; then it is impossible to meet the threshold of 5% (Figure 22).

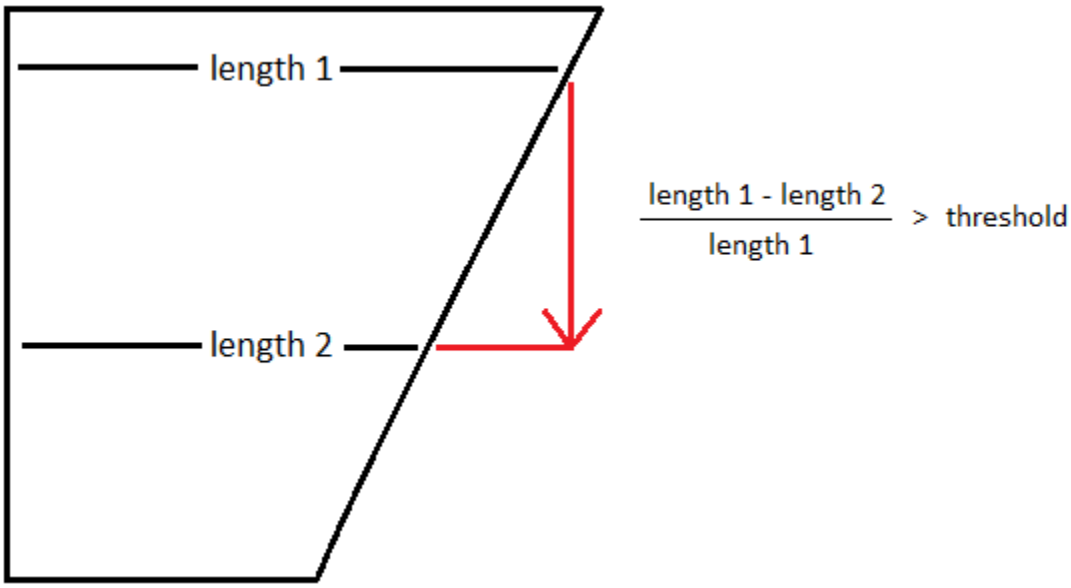

Figure 22: All sequences are sorted by length and all sequences which differ beyond a certain threshold in their lengths, are excluded;

Once all the distances are computed and the set of the remaining sequence has been reduced, the sequence alignment can be performed on only a small subset of the prior very large data set.

3.5 Implementation

The way we can compare two sequences using Markov Chain profiles is to view the cumulative differences as state values. Then the probability of each nucleotide depends on *k* previous nucleotides, where *k* is the order of the Markov Chain profiles. Depending on the highest Markov Chain order, the number of states varies. For example, in Markov Chain order 1 (M_1), there are only four states: A, T, C, G. Proceeding to Markov Chain order 2, the number of states increases to 16 (AA, AT, AC, AG, TA, TT, TC, TG, CA, CT, CC, CG, GA, GT, GC, GG). The total number of states can be written as 4^n, where *n* is the highest order of the Markov Chain profiles.

A mismatch in one sequence will affect a different number of states, dependent on the set order of Markov Chains (Figure 23 and 24).

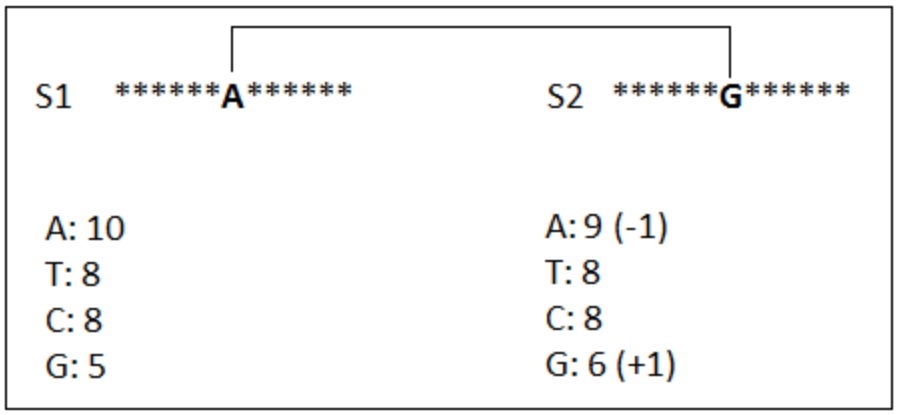

Figure 23: Markov Chain profiles of order 1 with changing states upon encountering a nucleotide mismatch;

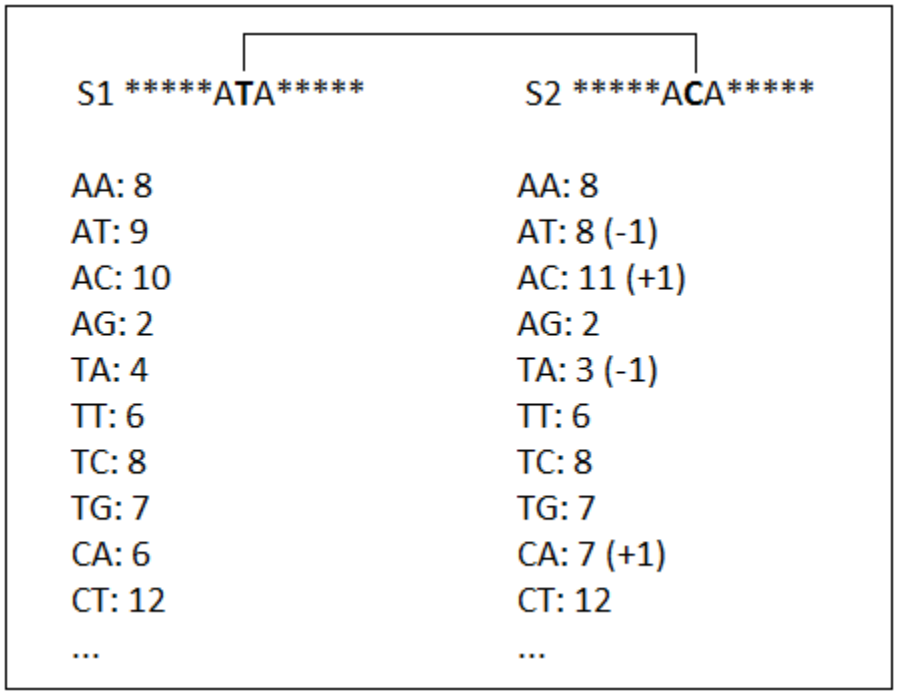

Figure 24: Markov Chain profiles of order 2 with changing states upon encountering a nucleotide mismatch;

While the substitution of an A to a G only increases the score for G and decreases the score for A in M_1, this single substitution affects 4 states in M_2,

and would affect 8 states in M_3. Hence the number of affected states in defined to be 2^n, where *n* equals the highest Markov Chain order.

The worst case cumulative differences can be calculated using the formula:

$$2 * n * m$$

Where *n* - Markov Order and *m* - number of mismatches.

3.6 Higher Order Markov Chain Models for the Long Sequence Comparison Problem

We face the following problem: The longer the sequence is, the more deviation from the worst case scenario occurs. This happens due to the fact that the same n-mers will be hit repeatedly in very long sequences (Figure 25).

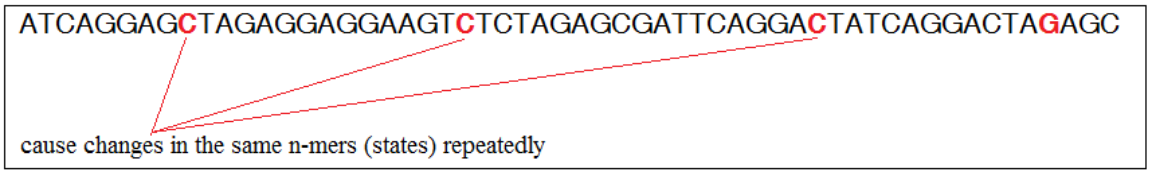

Figure 25: The longer the sequences, the higher the probability for a mismatch to alter the state of several states repeatedly;

We have developed two data structures in order to provide the necessary functionality: *Markov_Chains_Profile* and *Susbequences_Frequency_Array* (Figure 26).

Figure 26: The two classes *Markov_Chains_Profile* and *Subsequences_Frequency_Array*, which are used to compute the Markov Chain profiles for the sequences and to calculate their distances;

The class *Markov_Chains_Profile* has two data members and provides several methods (Figure 27).

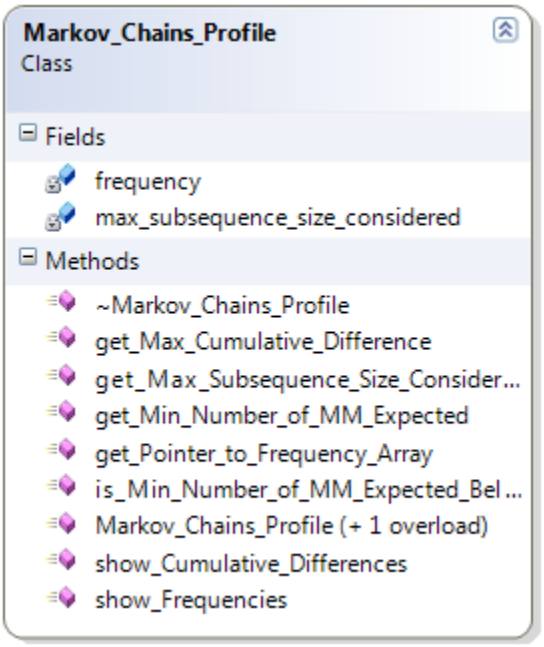

Figure 27: The class *Markov_Chains_Profile*, its data members, and its methods;

The variable *max_subsequence_size_considered* represents the highest Markov Chain order the object is supposed to implement. The field *frequency* is an array of instances of *Subsequences_Frequency_Array*. The size of this array is the

highest Markov Chain order supported by the object, stored in the variable *max_subsequence_size_considered.*

The function *get_Pointer_to_Frequency_Array* provides access to the frequency arrays of the object and is used in the process of comparing two profiles.

The second class we have developed is the *Subsequences_Frequency_Array*. It has three data members: *array_size*, *frequency*, and *subsequence_length* (Figure 28).

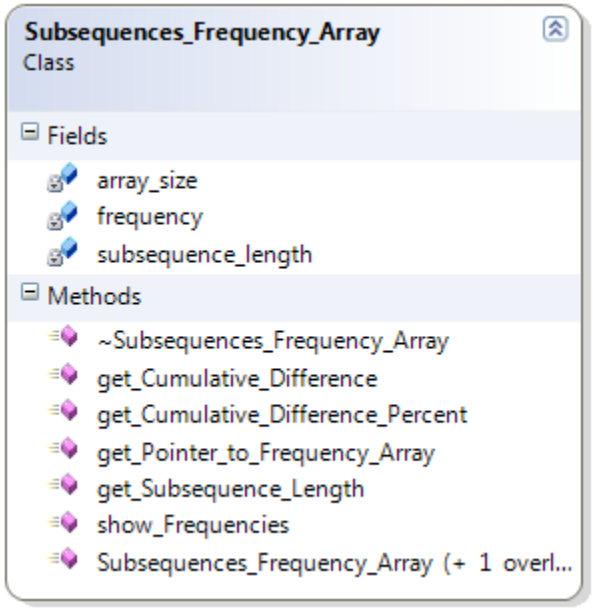

Figure 28: The class *Subsequences_Frequency_Array*, its data members, and its methods;

The field *subsequence_length* represents the highest Markov Chain order an instance of this object is expected to support. Its value is passed as an argument to the constructor when creating the array within the *Markov_Chains_Profile* object. The field *array_size* represents to total length of this frequency array, and

is calculated by 4^l, where l = *subsequence_length*. This means with increasing Markov Chain orders, the array will grow exponentially in size. The field *frequency* is an unsigned integer array of size *array_size*. It serves as a counting array to store the frequencies of occurrences of all possible n-mers.

The frequency array is built by converting every occurrence of a subsequence of length *n* (*n* = Markov Chain order) into an integer number. This integer number represents a unique index within the frequency array. Every occurrence of a certain n-mer will increase the value located at the index in the frequency array corresponding to the n-mer's converted integer value. Figure 29 will exemplify this procedure:

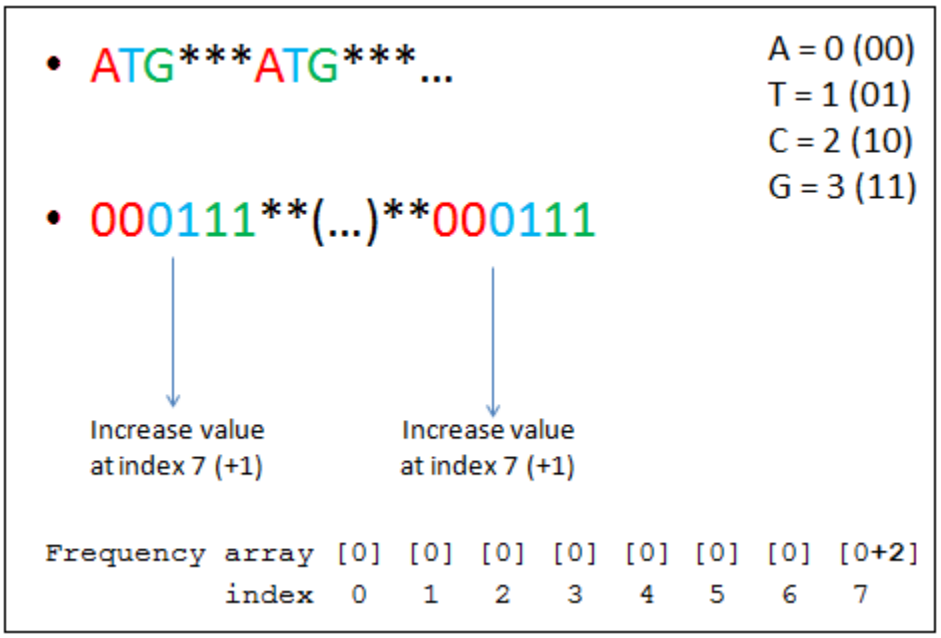

Figure 29: The procedure of converting of n-mers into integer numbers;

Each nucleotide corresponds to a 2-bit number: A=00, T=01, C=10, and G=11. All n-mers are investigated converted to their appropriate integer values. These integer values then serve as indices for the frequency array, and each occurrence of a certain n-mer increases the value present at its representing index in the frequency array.

Once all frequency arrays for all sequences are created, the Markov Chain profiles can be compared in order to obtain the cumulative differences among them, using the method *get_Cumulative_Difference* (Figure 30).

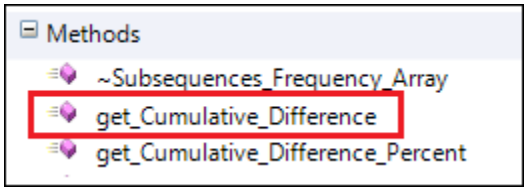

Figure 30: The function *get_Cumulative_Difference*, used to compare two frequency arrays;

The cumulative differences are calculated by taking the sum of all absolute values of the differences per location within the frequency arrays $frequency_1$ and $frequency_2$:

$$\sum_{0}^{array\_size-1} |frequency_1[i] - frequency_2[i]|$$

With *i* going from 0 to array_size-1;

As an example, assume the frequency arrays $frequency_1$ and $frequency_2$ of two Markov Chain profiles to be as follows:

frequency$_1$: [1] [3] [7] [0] [9] [5] [3] [0] [0] [1] [0] [3]

frequency$_2$: [6] [2] [2] [0] [0] [1] [1] [0] [9] [4] [4] [0]

Then the following absolute differences prevail among them:

Differences: 7 1 5 0 9 6 2 0 9 3 4 3

The cumulative difference of Markov Chain profile 1 and Markov Chain profile 2 is therefore: 7 + 1 + 5 + 0 + 9 + 6 + 2 + 0 + 9 + 3 + 4 + 3 = **49**; this will be seen as the distance value between the corresponding sequence 1 and sequence 2.

If we were to use the worst case scenario value as a threshold to filter out distant sequences, almost all sequences would be passing this restriction, hence being not very effective. To illustrate this problem we performed several simulations using randomly generated test sequences of sizes between 100 and 10,000 nucleotides, which is the typical range of gene lengths. We then computed the cumulative differences by inserting random Single Nucleotide Polymorphisms (SNPs) in a copy of that same sequence, and compared the results with the worst case cumulative differences. This experiment was conducted for each Markov Chain order from one to seven. Setting the threshold for dissimilarity to 10% (which depicts the amount of mismatches between two sequences), we compared the average dissimilarities with the worst case dissimilarities. The results projected precisely our prediction: For long sequences, the differences between the predicted and the actual values are much greater than for short sequences (Figure 31).

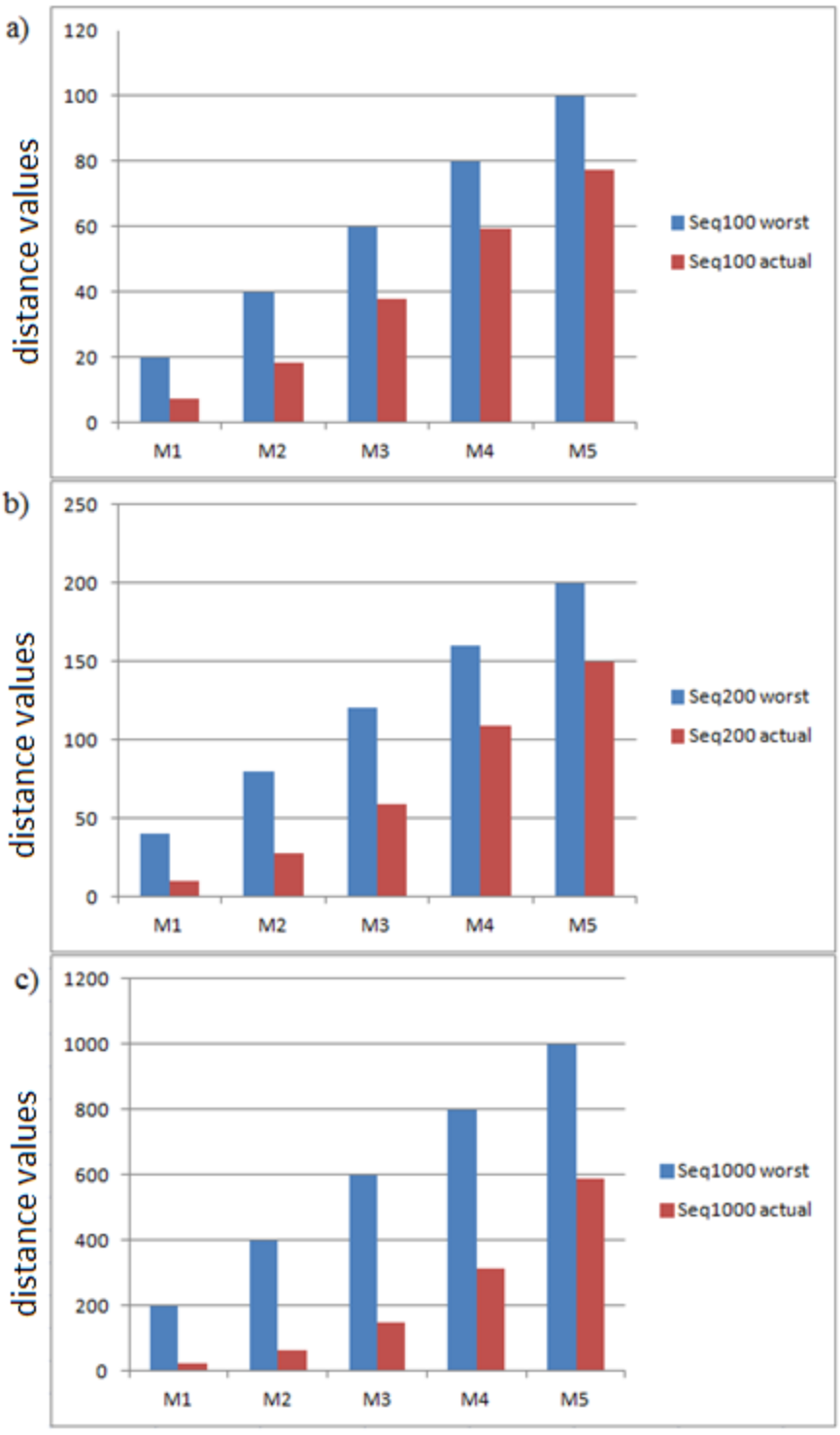

Figure 31: Markov Chain Profile M_1 - M_5 predictions of cumulative differences (blue columns) versus actual cumulative differences (red columns);
a) illustrates the findings for random sequences of length 100, b) of length 200 and c) of length 1000;

While this difference between the worst cases and the average actual cases are decreasing with higher Markov Chain orders, one cannot simply choose an arbitrarily large Markov Chain order for this purpose. The reason therefore is that the higher the Markov Chain order, the larger the number of n-mers that are present in the Markov Chain models. At the point where $O(n) < O(m^4)$, with *n* being the total number of sequences and m being the highest Markov Chain order, this exponentially growing number of n-mers rises to the peak where the time needed for computations will not compensate sufficiently for the exclusion of sequences from the alignment process.

3.7 Proposed Approach

It is possible to identify alternative thresholds for different sequence lengths and different Markov Chain orders to eliminate sequences. These new thresholds are found at distance values, which allow 95% or 99% of all sequences to pass, while the few remaining outliers will be excluded. Thus, these new boundaries serve as the criteria to indicate whether sequences, due to their distances, can be excluded from the alignment process. Using three steps of exhaustive analysis, those boundaries can be predicted for different sequence lengths, and different Markov Chain orders, which can be increased until a 95% or 99% threshold that is sufficient enough has been found.

Those three steps include analysis with random sequences and random mutations (SNPs), real sequences with randomly inserted SNPs, and finally real sequences with real mutations.

3.8 Computational Modeling and Validation

3.8.1 Random Sequences and Random Mutations

In the first analytical modeling we produced randomly generated sequences of lengths: 100, 200, 300, 500, 1,000, 2,000, 3,000, 5,000, and 10,000. We then introduced mutations at random locations with the following mutation rates: 1%, 2%, 3%, 5%, 10%, and 15%. We calculated the Markov Chain profile distances for all Markov Chain orders from one to six. Each possible combination of parameters was executed 1,000 times in order to obtain representative averages and standard deviations (Figures 32, 33).

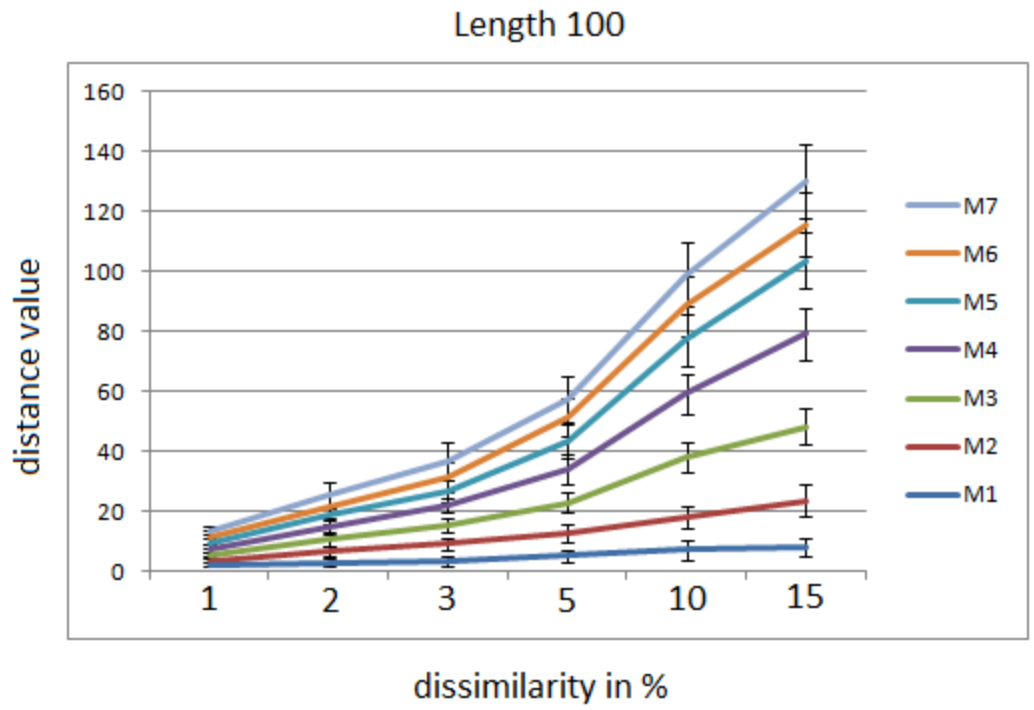

Figure 32: Average distance values and standard deviations for random sequences of length 100 for different Markov Chain orders and increasing dissimilarities caused by introduced SNPs;

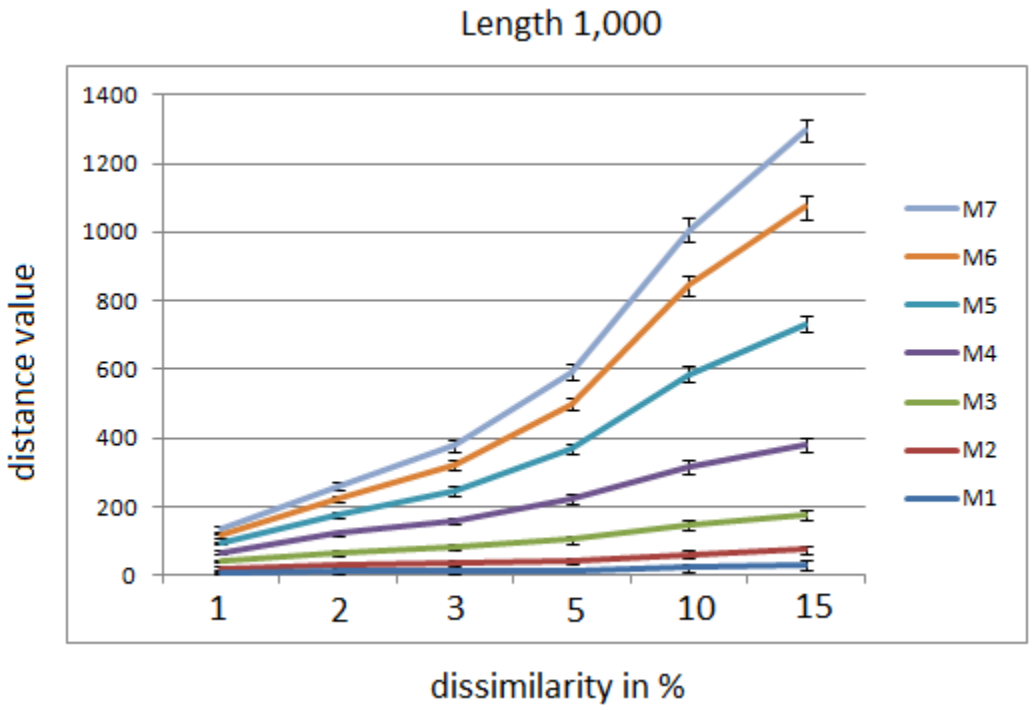

Figure 33: Average distance values and standard deviations for random sequences of length 1,000 for different Markov Chain orders and increasing dissimilarities caused by introduced SNPs;

At the length of 200, with an introduced dissimilarity of 3% and Markov order 3, the worst-case cumulative dissimilarity value is 36. However, our calculations have shown that setting the threshold to 30 will already include 95% of the executions (Figure 34).

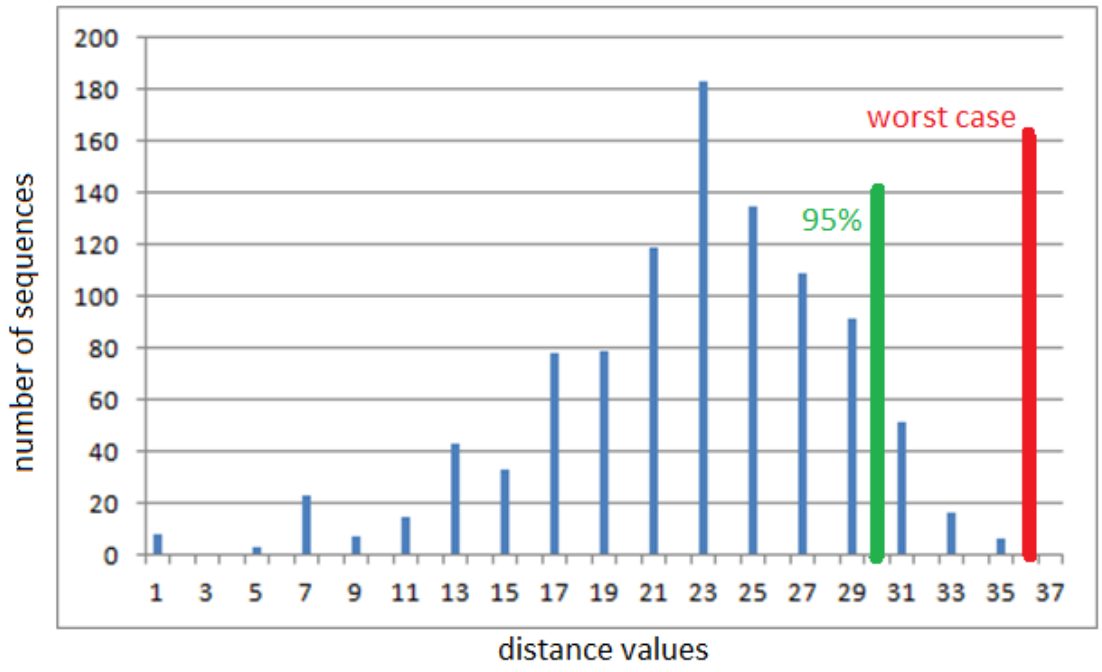

Figure 34: Random sequence length: 200; Dissimilarity: 3%; Markov Order: 3; 95% threshold at 30; worst case cumulative difference at 36;

Very similar was the result from a different experiment with longer sequences (length 1,000), 1% dissimilarity, and Markov order 4. While the worst case cumulative difference is 80, the 95% threshold is at 68 (Figure 35).

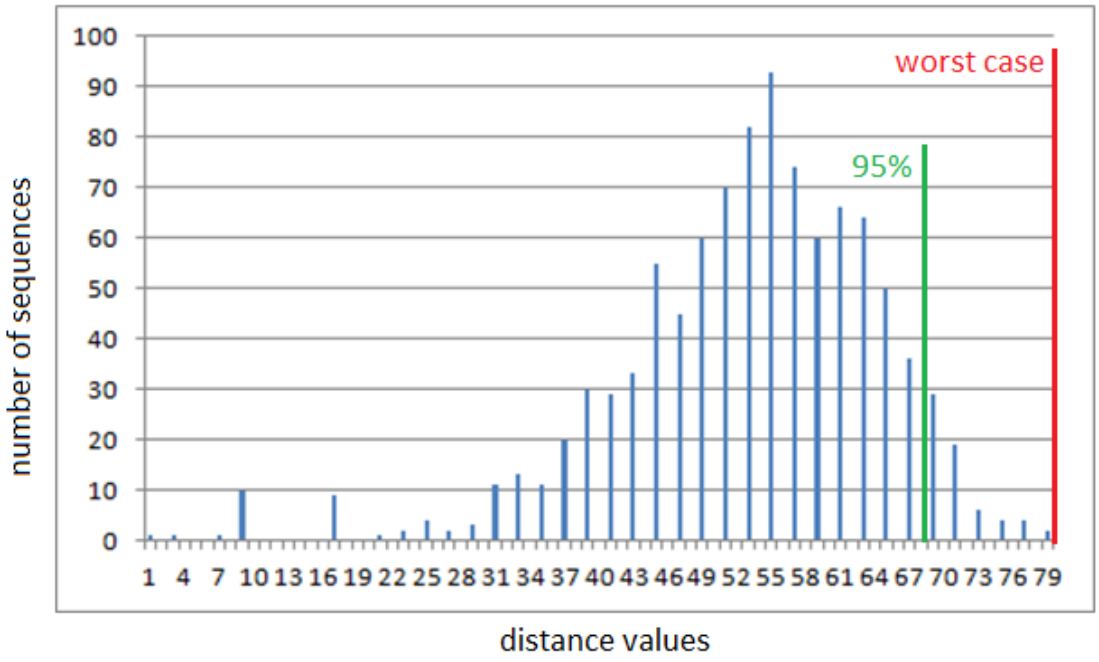

Figure 35: Random sequence length: 1,000; Dissimilarity: 1%; Markov order: 4; 95% threshold at 68; worst case cumulative difference at 80;

As we increase the sequence length, our observations reflect exactly what we predicted in our analysis: The worst-case cumulative difference value is far from the actual cumulative differences, which we observed during the simulations. For the next example we then further increased the sequence length to 2,000, introduced 3% dissimilarity and applied Markov Chain order 3. While the worst-case cumulative difference value for this situation is at 360, at 114 we already included 95% of all executions, 99% at 124 (Figure 36).

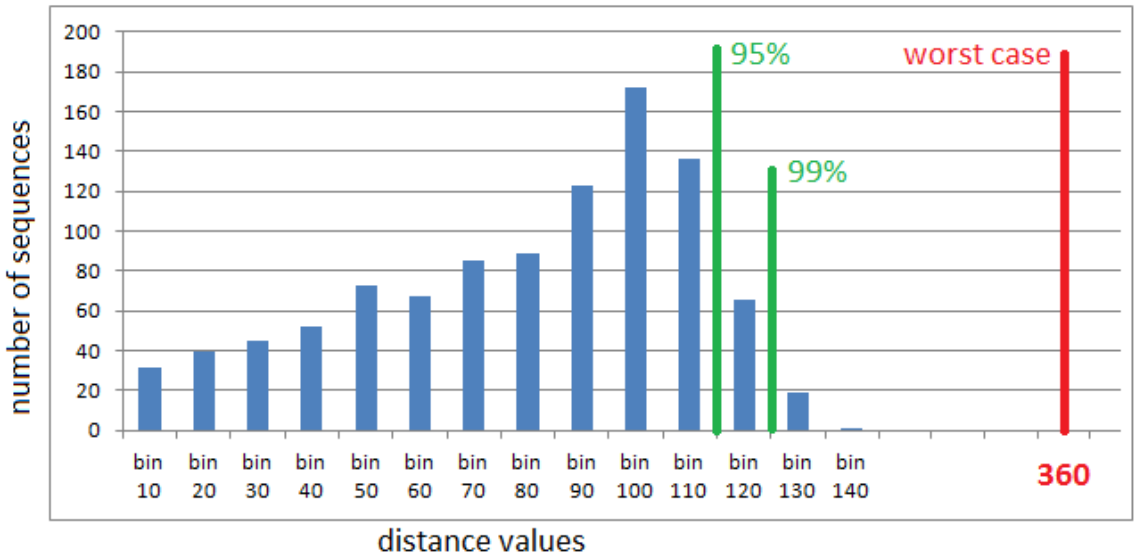

Figure 36: Random sequence length: 2,000; Dissimilarity: 3%; Markov order: 3; bin size: 10;

Another property that was expected to have significant influence on the 95% threshold is order of dissimilarity percentage. To observe this property we ran the experiment on sequences of length 1,000 and used Markov order 4. The worst-case cumulative difference value is 80, with an introduced dissimilarity of 1%. Our 95% threshold was at 68 (Figure 37).

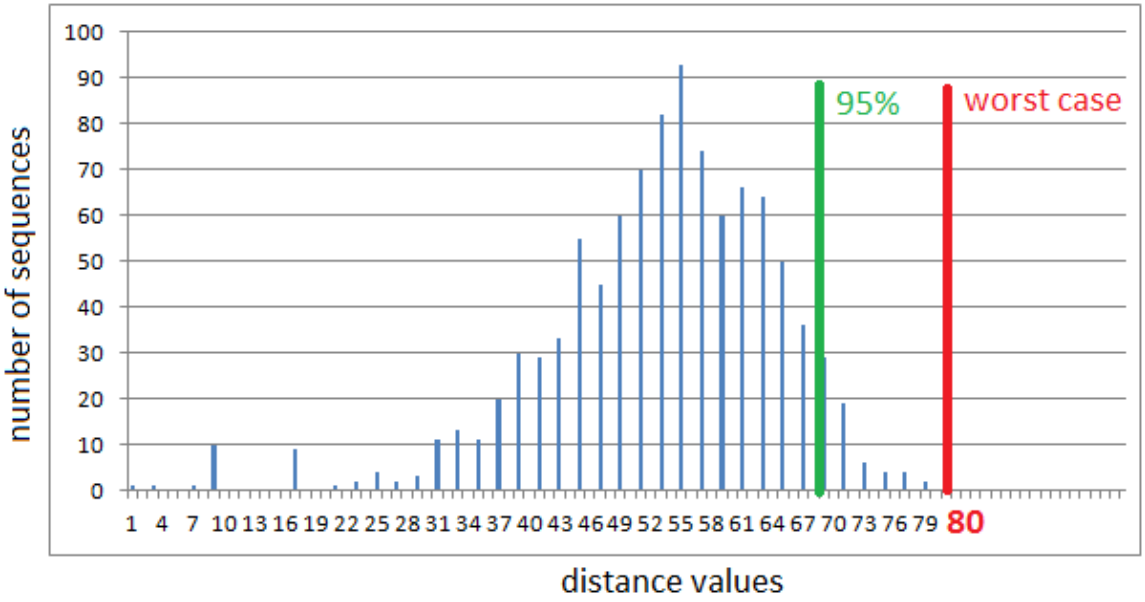

Figure 37: Random sequence length: 1,000; Dissimilarity: 1%; Markov order: 4;

The result showed us that the worst-case assumption (80) was quite close to the actual worst-case cumulative difference (68), especially when compared with the next simulation, where only the dissimilarity was increased to 5%. Now, the expected worst-case cumulative difference was at 400, while the actual 95% threshold was at exactly half of that value, at 200. Even the 99% mark (212) was not significantly closer to the worst-case expectation (Figure 38).

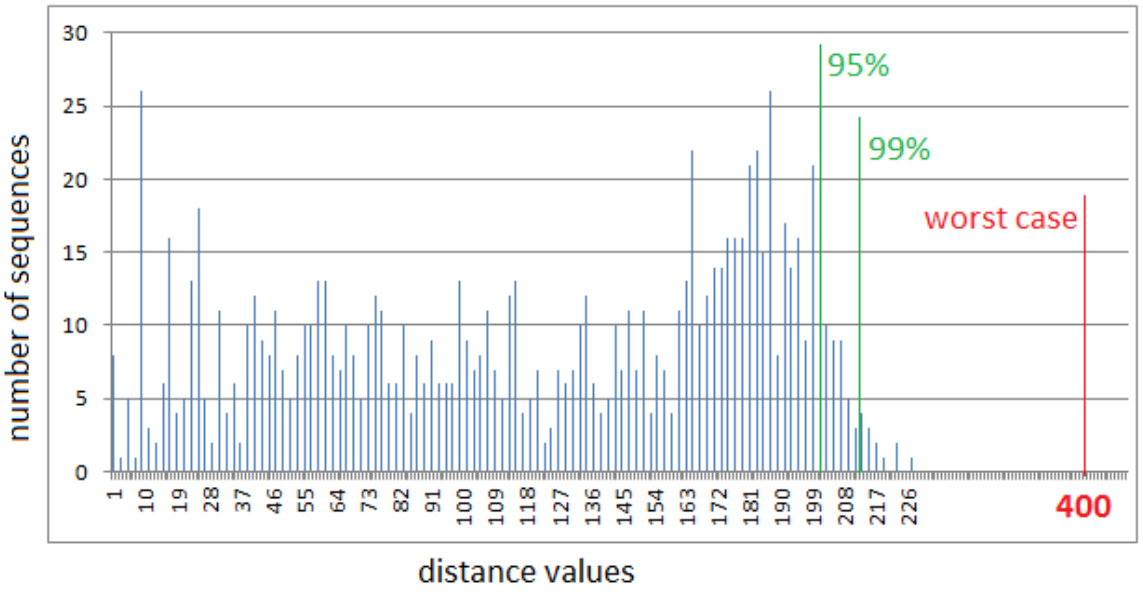

Figure 38: Random sequence length: 1,000; Dissimilarity: 5%; Markov order: 4;

This graph can be smoothened out by increasing the bin sizes of the distance values on the x-axis from 1 to 10 (Figure 39).

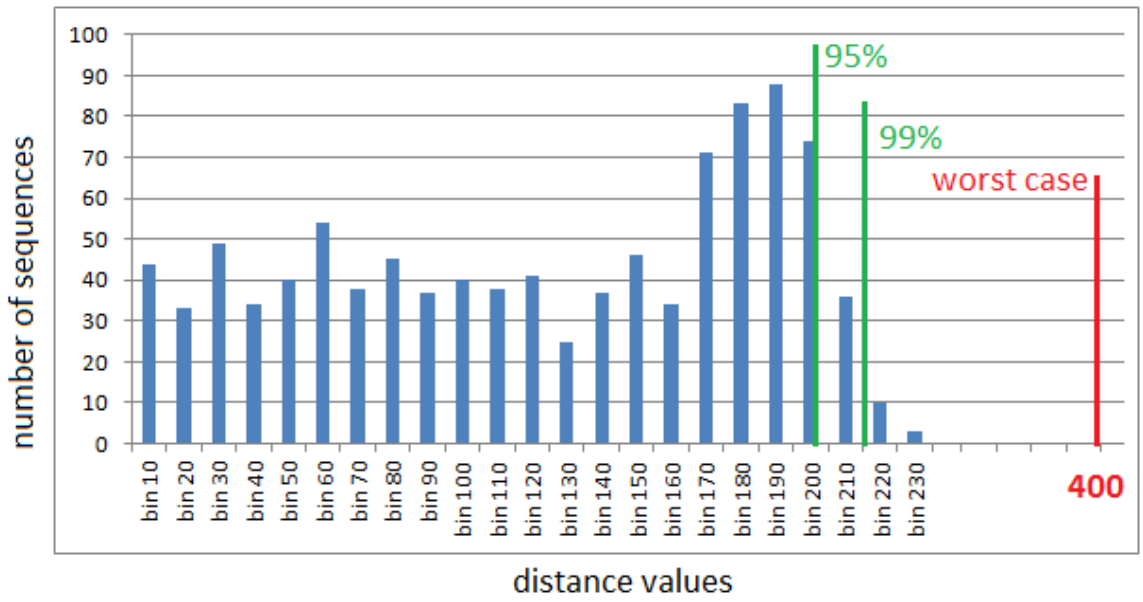

Figure 39: Random sequence length: 1,000; Dissimilarity: 5%; Markov order: 4; bin size increased to 10;

Our analysis showed that with increasing sequence length, even a small increase in dissimilarity has a major impact on how close the expected worst-case cumulative difference is to the actual measurements. The simulations confirmed said predictions.

3.8.2 Real Sequences and Random Mutations

After the demonstrated simulations had verified the predictions of our analysis, we have performed similar applications on real DNA data, instead of randomly generated sequences. For this purpose the gene thrA from Escherichia coli K-12 [54] was taken, which has a length of 2463 nucleotides. We introduced mutations of various rates (1%, 2%, 3%, 5%, 10%) at random locations within this nucleotide sequence and performed the distance calculations on all Markov Chain profiles for the Markov orders from one to six. With an introduced dissimilarity of 1%, and a Markov Chain order 1, the estimated worst case distance value is 49.26. However, our model has shown that setting the threshold to 16 will already result in an inclusion of 95% of all sequences, while a threshold of 20 will include 99% (Figure 40).

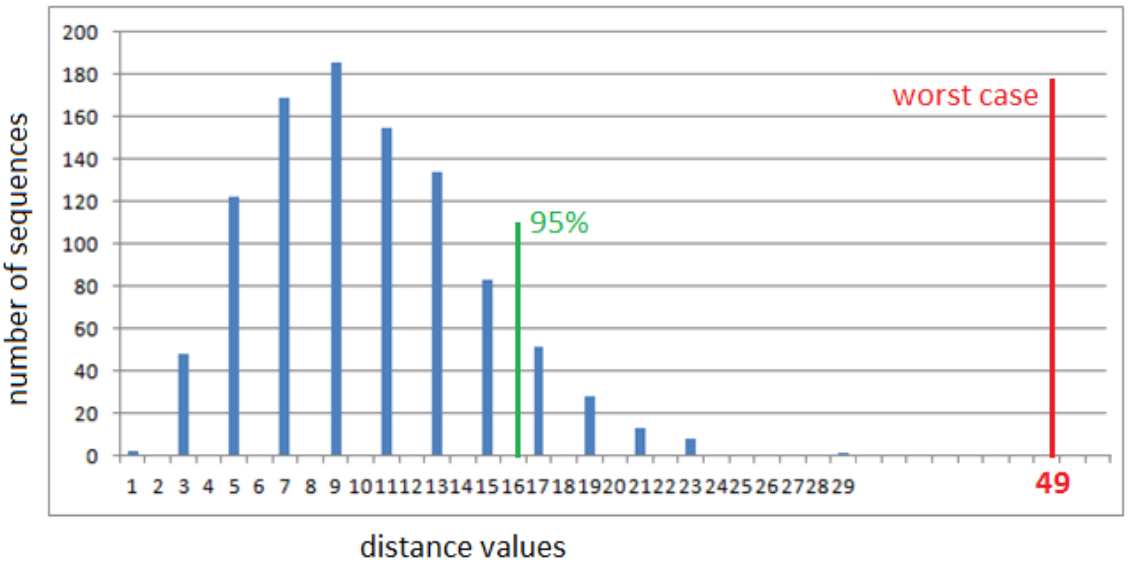

Figure 40: Gene length: 2,463; Dissimilarity: 1%; Markov order: 1;

After increasing the Markov order to 5 with the other parameters unaltered, the 95% and 99% thresholds yielded a distance value of 196 and 203, respectively, while the worst case distance increased to 246.3 (Figure 41).

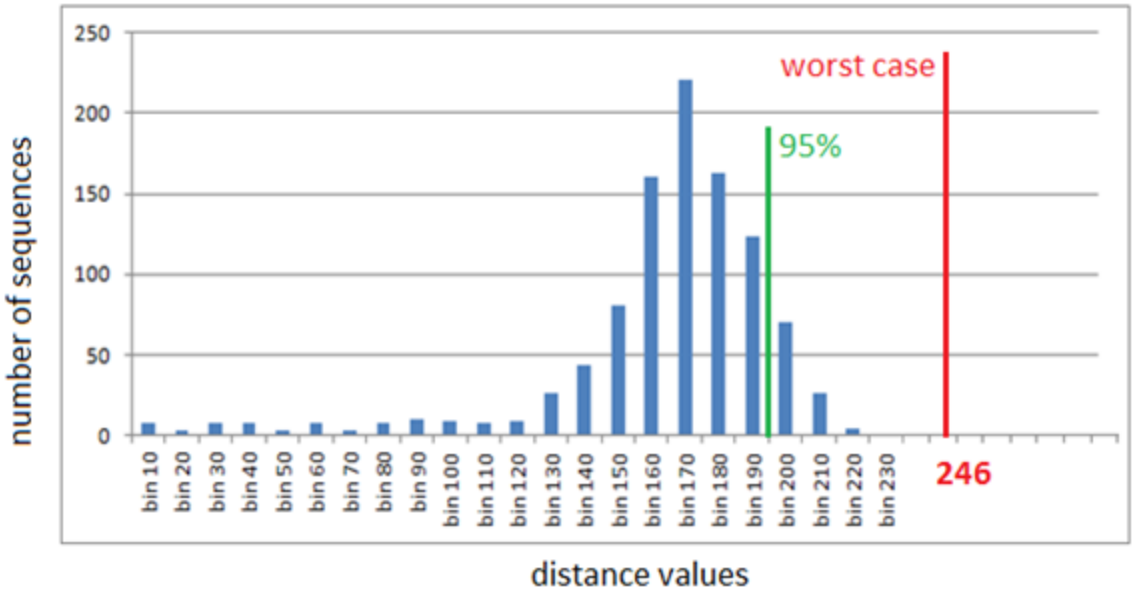

Figure 41: Gene length: 2,463; Dissimilarity: 1%; Markov order: 5; bin size increased to 10;

The same gene was again mutated at random locations with a dissimilarity rate of 10%. Using both Markov Chain orders 1 and 3, a major difference between the worst case sequence distances and the 95% threshold for each were observed (Figures 42, 43).

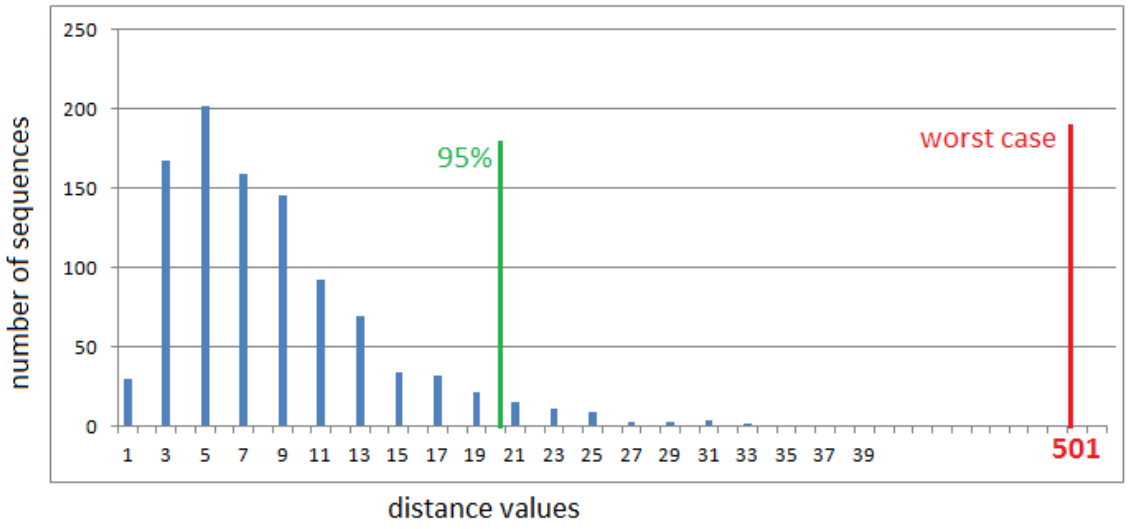

Figure 42: Gene length: 2,463; Dissimilarity: 10%; Markov order: 1;

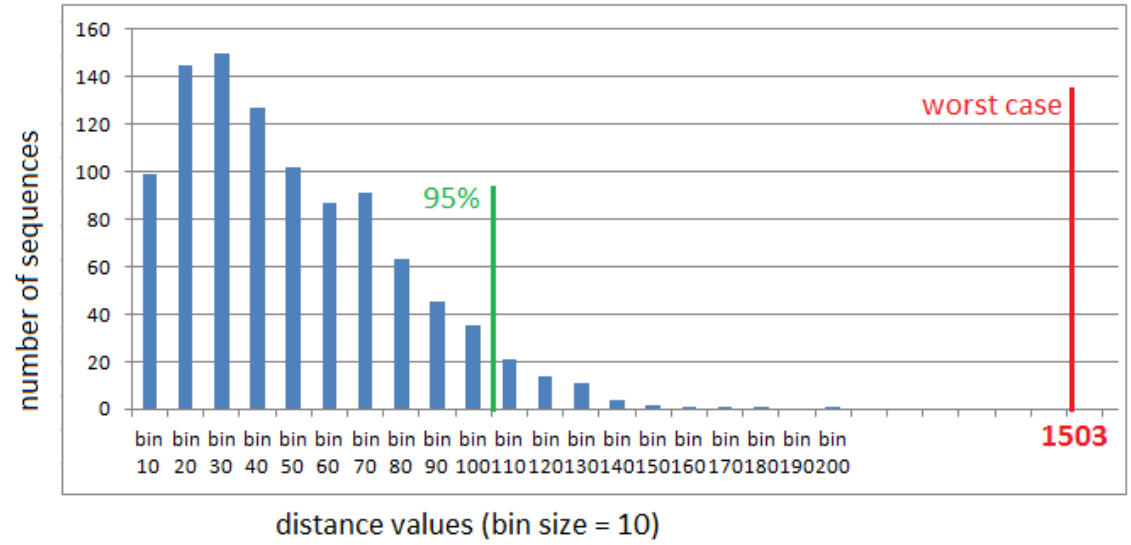

Figure 43: Gene length: 2,463; Dissimilarity: 10%; Markov Order: 3;

These observations are in line with what the simulations using randomly generated sequences were suggesting, as Figures 44 and 45 illustrate.

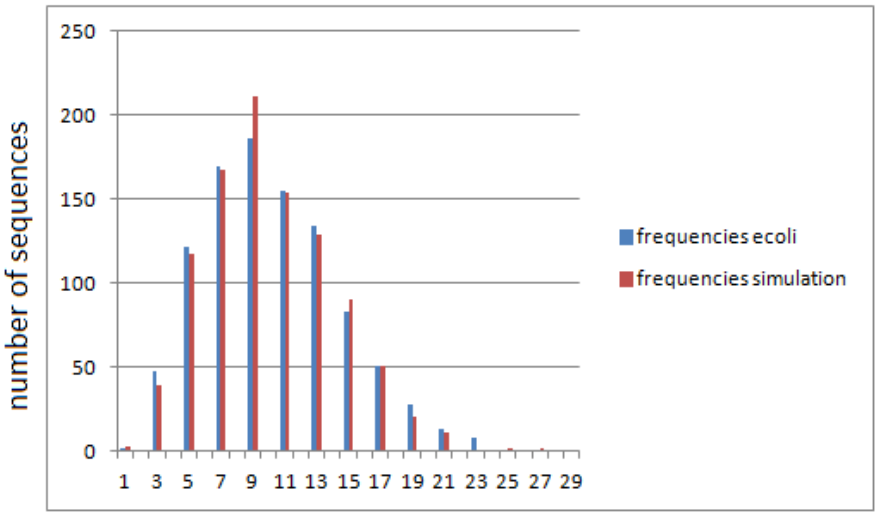

Figure 44: Sequence length (gene and random simulation): 2,463; Dissimilarity: 1%; Markov order: 1;

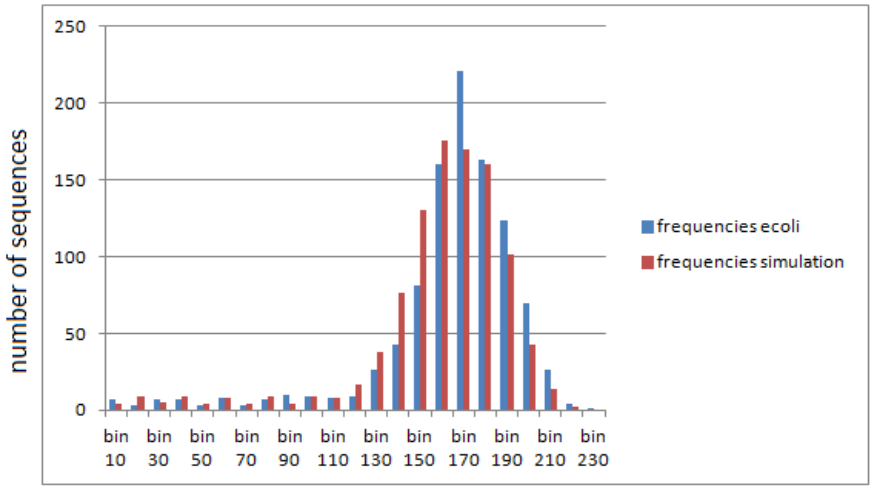

Figure 45: Sequence length (gene and random simulation): 2,463; Dissimilarity: 1%; Markov order: 5;

3.8.3 Real Sequences and Real Mutations

To complete the validation of the proposed work, several examples of Segment 6 from the H1N1 influenza virus were chosen, where a certain degree of SNP occurrences was expected. For this purpose, six such sequences were investigated, whose length is 1410 nucleotides each. The samples were taken

from different geographical locations: 1 from Georgia [55], 1 from Illinois [56], 1 from Stockholm [57], 1 from Strasbourg [58], 1 from Texas [59], and 1 from Thailand [60]. Upon aligning each sequence with one another, the following similarities were found:

Table 13: Similarities of H1N1 Influenza Virus, Segment 6;

	Georgia GQ323549	Illinois GQ894817	Stockholm GQ365682	Strasbourg GQ329108	Texas GQ323513	Thailand GQ866953
Georgia GQ323549	100%	99.4%	99.9%	99.7%	99.8%	99.7%
Illinois GQ894817		100%	99.6%	99.7%	99.5%	99.7%
Stockholm GQ365682			100%	99.9%	99.9%	99.9%
Strasbourg GQ329108				100%	99.8%	100%
Texas GQ323513					100%	99.8%
Thailand GQ866953						100%

Next, Markov Chain models have been used to calculate the distances between the sequences. This gave us the opportunity to estimate whether our model could accurately be used to find the maximum distance values that should be considered for a sequence not to be excluded. Since the Segment 6 of the influenza virus is of length 1410 nucleotides, we compared the distances with simulated sequences of length 1410, and assumed a dissimilarity of 1%. When we compared the actual distances to the worst case scenario distance values, we observed the expected big differences. We then looked at the 95% and 99%

thresholds and found those to be a lot closer to the actual values of the real gene distances (Figure 46).

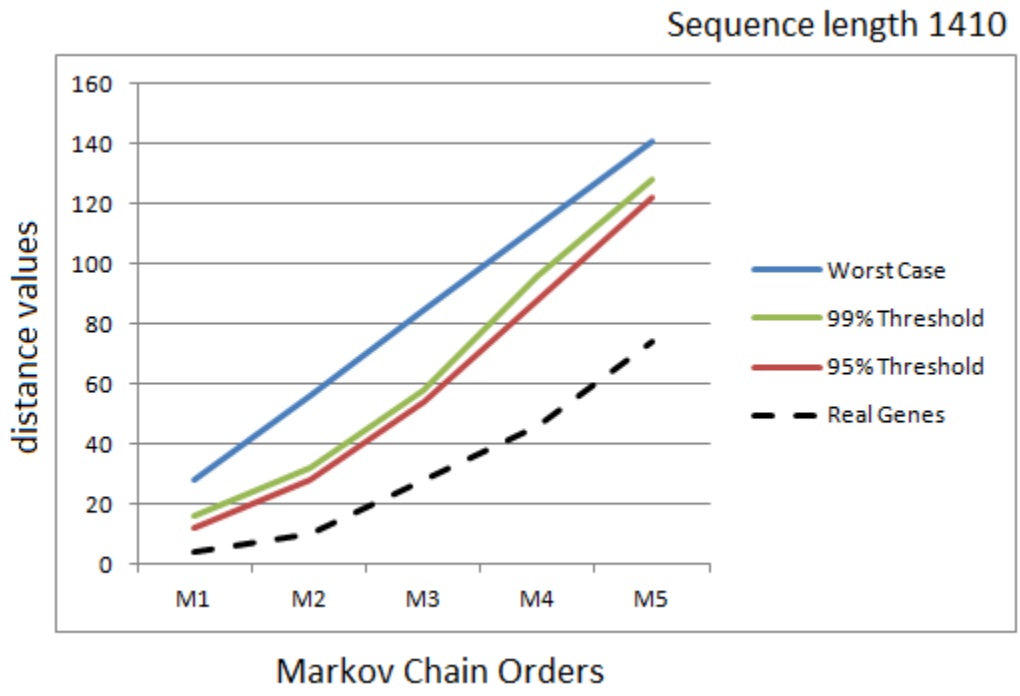

Figure 46: Real gene distance values (H1N1 influenza virus segment 6) compared to simulated sequences of the same length for Markov Chain orders 1-6;

3.9 Discussion and Conclusion

We have successfully shown that our model can adequately be applied to real data. The distance scores we retrieve hold the information whether a sequence needs to be excluded from the alignment process or not. By having a given sequence length, we can set a distinct threshold to keep at least 95% or 99% of all sequences that possess a distance value below said threshold. We can choose the proper Markov Chain model in order to estimate the maximum allowed distance. The simulations predicted that in lower Markov Chain models

the differences between the thresholds we set and the worst cases are much larger than at higher Markov Chain models. Increasing sequence lengths were expected to have the same effect. Our application on real data has proven those predictions right (Figure 47 + 48).

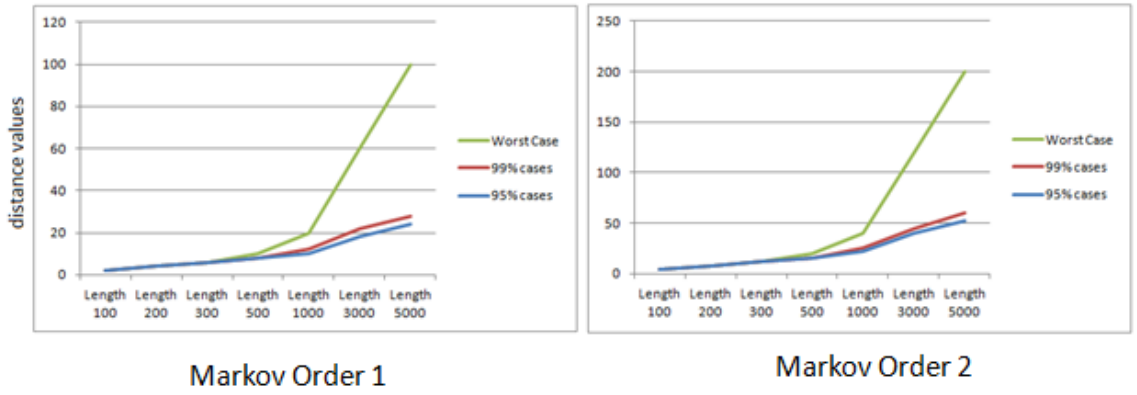

Figure 47: With increasing sequence length, the difference between worst case dissimilarity and the threshold to include 95% or 99% of the sequences increases; 1% dissimilarity;

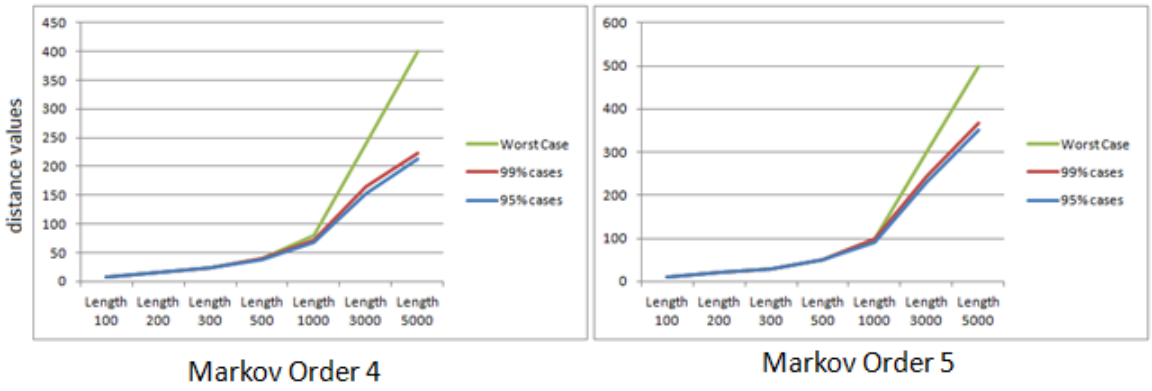

Figure 48: The difference between the worst case dissimilarity and the threshold to include 95% or 99% of the sequences decreases with higher Markov Chain orders; dissimilarity 1%;

The higher the dissimilarity of the sequences is, the greater is the deviation of the worst case scenario from the actual distance values and from the 95% or 99%

thresholds. Previous examples (1% dissimilarity) have been evaluated with a 2% (Figure 49) and a 5% (Figure 50) dissimilarity as well for confirmation.

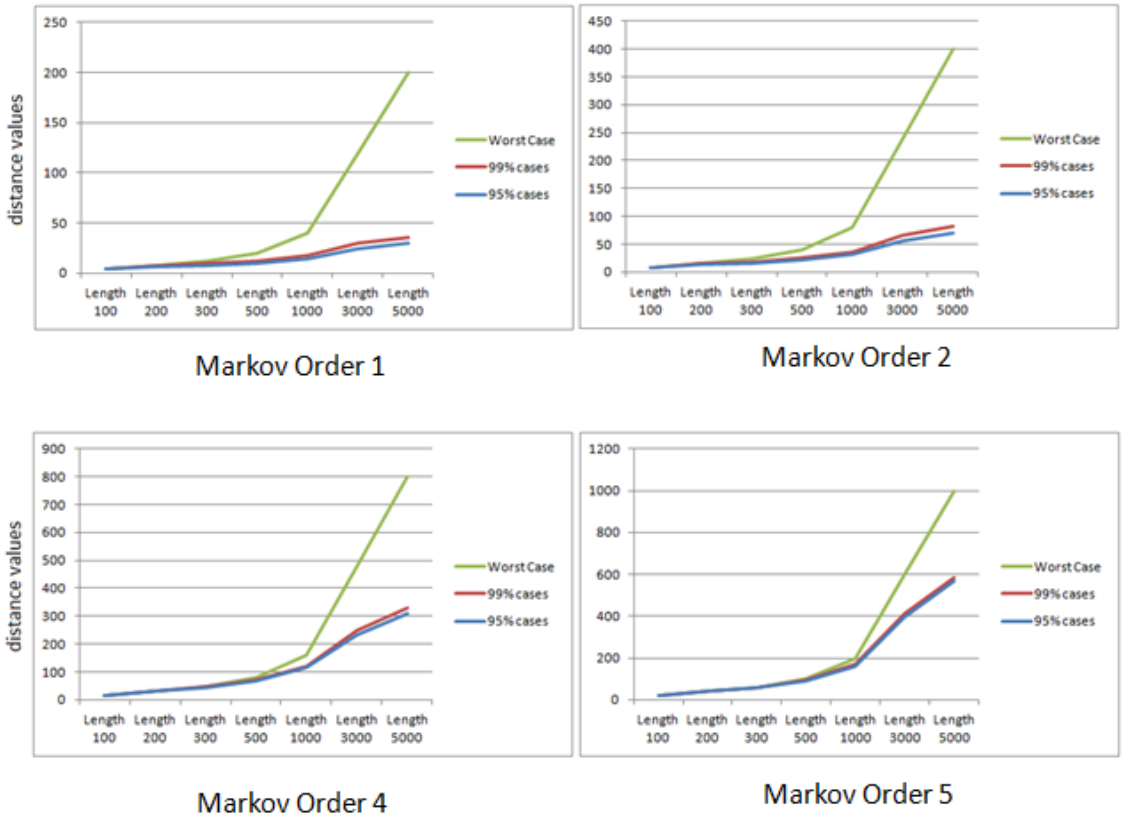

Figure 49: Increasing Markov Chain orders reduce the differences between the worst case cumulative distance values and the thresholds for 95% and 99%; increasing dissimilarities (here 2%) have the opposite effect;

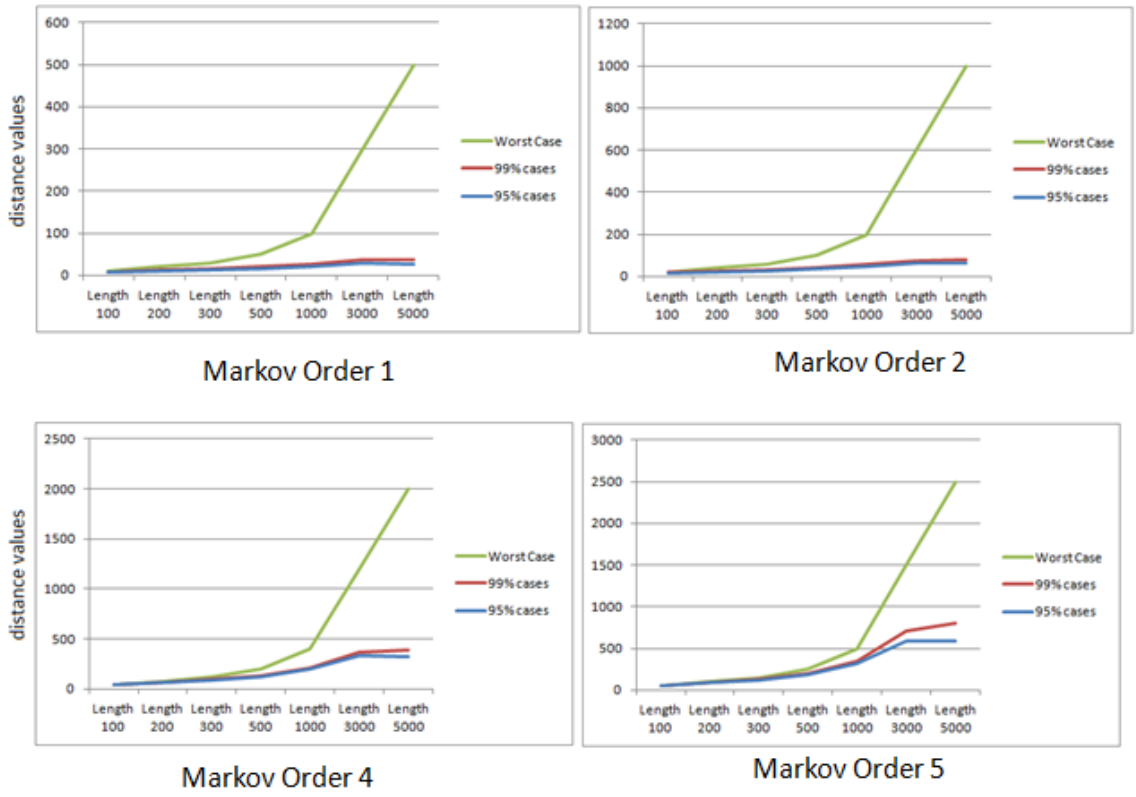

Figure 50: Higher rates of dissimilarities (here 5%) increase the gap between worst case cumulative differences and the thresholds for 95% and 99%;

The presented solution will allow to greatly improve the times needed to perform pair-wise sequence alignment, because we reduce the total number of comparisons that need to be performed. With the information one has about the data set, the maximum allowed threshold will be calculated and sequences will be excluded if they score above this threshold. In combination with the binary data compression introduced in the beginning of this dissertation, the resulting data set will only require a fraction of the space it initially needed. Additionally, the now much smaller data set is also cleaned of low quality reads and other redundant information such as sequence headers and quality scores.

4. Conclusion and Future Work

The presented work addresses two of the major challenges in HTS data analysis. Using the proposed approaches one can reduce the file sizes of the sequences produced by NGS instruments by 95% - 99%. This lessens the amount of hard-disk space required to store the data sets and allows a much faster transmission over the internet thereof. Additionally, the time needed to read and write to the proposed data formats is several magnitudes smaller than what alternative compression formats require. Where once storage clusters and machines with high end processing powers were needed, many problems can now be dealt with right there where the data was created, without the need to move it.

Furthermore, the computation times needed for some of the most time consuming analytical procedures in genomics are decreased significantly. Via the proposed method, one can preemptively reduce the size of a data set before time consuming alignment procedures are applied. Utilizing Markov Chain models to identify dissimilarities amongst sequences, distant sequences can be excluded from the time consuming sequence alignment process. This made it possible to cluster over 9 million gene sequences into 6 million clusters with one representative sequence each within several hours - a process previously expected to have requirements of vast proportions.

During the course of development to facilitate the presented approaches, several data structures and algorithms have been implemented. They do not only serve as tools to address the discussed challenges, but are also applied in attempts to

deal with other big-data problems faced in the NGS data analysis and genomics, such as pathogen detection in presence of complex environments, analysis of metagenomic samples, and copy number variation analysis.

References

[1] J. J. McCarthy, H. L. McLeod, G. S. Ginsburg, Genomic Medicine: A Decade of Successes, Challenges, and Opportunities. Sci. Transl. Med. 5, 189sr4 (2013).

[2] K. M. East, A. M. Hott, N. P. Callanan, N. E. Lamb, Journal of Genetic Counseling: Biotech 101: An Educational Outreach Program in Genetics and Biotechnology. Volume 21, Issue 5, pp 704-712, 2012

[3] A.G.A. Khaled, K. A. Hamam, M.H. Motawea, G.A.R. El-Sherbeny, Journal of Genetic Engineering and Biotechnology: Genetic studies on tissue culture response and some agronomical traits in Egyptian bread wheat. Volume 11, Issue 2, pp 79-86, 2013

[4] F. Sanger, M. Dowding, World Scientific Pub Co Inc: Selected Papers of Frederick Sanger (With Commentaries). 1996

[5] F. Sanger, S. Nicklen, A.R. Coulson, National Academy of Sciences of the United States of America: DNA sequencing with chain-terminating inhibitors. Volume 74, No. 12, pp 5463-5467, 1977

[6] B. Toner, " In Sequence Survey: Illumina Holds Two-Thirds of Sequencing Market, Splits Desktop Share with Ion PGM", GenomeWeb, October 2012. Web. April 2014

[7] http://www.illumina.com/, last accessed April 2014

[8] K. Robinson, "What Might Knock Illumina Off Its Perch?", OmicsOmics, April 2014. Web. April 2014

[9] http://www.pacificbiosciences.com/, last accessed April 2014

[10] http://www.454.com/, last accessed April 2014

[11] https://www.lifetechnologies.com/us/en/home/brands/ion-torrent.html, April 2014

[12] http://www.appliedbiosystems.com/absite/us/en/home/applications-technologies/solid-next-generation-sequencing/next-generation-systems/solid-sequencing-chemistry.html, April 2014

[13] R. Cronn, A. Liston, M. Parks, D. S. Gernandt, R. Shen, T. Mockler, Nucleic Acids Research: Multiplex sequencing of plant chloroplast genomes using Solexa sequencing-by-synthesis technology. Volume 36, Issue 19, pp e122, 2008

[14] L. Liu, Y. Li, S. Li, N. Hu, Y. He, R. Pong, D. Lin, L. Lu, M. Law, BioMed Research International: Comparison of Next-Generation Sequencing Systems. Volume 2012, 2012

[15] H. Y. Lam, M. J. Clark, R. Chen, R. Chen, G. Natsoulis, M. O'Huallachain, F. E. Dewey, L. Habegger, E. A. Ashley, M. B. Gerstein, A. J. Butte, H. P. Ji, M. Snyder, Nature Biotechnology: Performance comparison of whole-genome sequencing platforms. Volume 30, pp 78-82, 2012

[16] E. C. Hayden, "Next-generation genome sequencers compared", Nature. April 2012. Web. April 2014

[17] G. Cochrane, C. E. Cook, E. Birney, GigaScience: The future of DNA sequence archiving. 2012

[18] W. R. Pearson, D. J. Lipman, Proceedings of the National Academy of Sciences of the United States of America: Improved tools for biological sequence comparison. Volume 85, pp 2444-2448, 1988

[19] A. Pollack, "DNA Sequencing Caught in Deluge of Data", The New York Times. November 2011. Web. April 2014

[20] S. W. Jun, M. Liu, K. E. Fleming, FPGA: Scalable multi-access flash store for big data analytics. Pp 55-64, 2014

[21] G. K. Wallace, Communications of the ACM: The JPEG still picture compression standard. Volume 34, Issue 4, pp 30-44, 1991

[22] M. Kircher, S. Sawyer, M. Meyer, Nucleic Acids Research: Double indexing overcomes inaccuracies in multiplex sequencing on the Illumina platform. Volume 40, Issue 1, pp e3, 2011

[23] P. J. A. Cock, C. J. Fields, N. Goto, M. L. Heuer, P. M. Rice, Nucleic Acids Research: The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. Volume 38, Issue 6, pp 1767-1771, 2009

[24] S. M. Huse, J. A. Huber, H. G. Morrison, M. L. Sogin, D. M. Welch, Genome Biology: Accuracy and quality of massively parallel DNA pyrosequencing. Volume 8, Issue 7, 2007

[25] W. R. Pearson, Methods in Enzymology: Rapid and sensitive sequence comparison with FASTP and FASTA. Volume 183, pp 63-98, 1990

[26] R. Leinonen, H. Sugawara, M. Shumway, Nucleic Acids Research: The Sequence Reads Archive. Volume 39, Issue suppl 1, pp D19-D21, 2010

[27] R. G. Gallager, Information Theory, IEEE Transactions: Variations on a theme by Huffman. Volume 24, Issue 6, 1978

[28] J. Ziv, A. Lempel, Information Theory, IEEE Transactions: Compression of individual sequences via variable-rate coding. Volume 24, Issue 5, pp 530-536, 1978

[29] G. Lakhani, Image Processing, IEEE Transactions: Modified JPEG Huffman coding. Volume 12, Issue 2, pp 159-169, 2003

[30] M. S. Vinton, Acoustics, Speech, and Signal Processing: Scalable and progressive audio codec. Volume 5, pp 3277-3280, 2001

[31] P. Deutsch, Network Working Group: DEFLATE Compressed Data Format Specification version 1.3. May 1996

[32] İ. Öztürk, İ. Soğukpınar, International Journal of Computer, Information Science and Engineering: Analysis and Comparison of Image Encryption Algorithms. Volume 1, Issue 3, 2007

[33] H. Li, R. Durbin, Bioinformatics: Fast and accurate short read alignment with Burrows-Wheeler transform. Volume 25, Issue 14, pp 1754-1760, 2009

[34] P. Ferragina, SODA '04: Compression boosting in optimal linear time using Burrows-Wheeler Transform. pp 655-663, 2004

[35] B. Langmead, S. L. Salzberg, Nature: Fast gapped-read alignment with Bowtie 2. Volume 9, pp 357-359, 2012

[36] R. Li, C. Yu, Y. Li, T. W. Lam, S. M. Yiu, K. Kristiansen, J. Wang, Bioinformatics: SOAP2: an improved ultrafast tool for short read alignment. Volume 25, Issue 15, pp 1966-1967, 2009

[37] A. Andersson, S. Nilsson, Journal of Experimental Algorithms: Implementing radixsort. Volume 3, Number 7, 1998

[38] D. A. Benson, M. Boguski, D. J. Lipman, J. Ostell, Nucleic Acids Research: GenBank. Volume 22, Issue 17, pp 3441-3444, 1994

[39] Y. Gangman, S. H. Sze, M. R. Thron, Bioinformatics: Identifying clusters in functionally related genes in genomes. Volume 23, Issue 9, pp 1053-1060, 2007

[40] D. S. Hirschberg, Journal of the ACM: Algorithms for the Longest Common Subsequence Problem. Volume 24, Issue 4, 1977

[41] S. R. Eddy, Nature Biotechnology: Where did the BLOSUM62 alignment score matrix come from?, Volume 22, pp 1035-1036, 2004

[42] B. Morgenstern, K. Frech, A. Dress, T. Werner, Bioinformatics: DIALIGN: Finding local similarities by multiple sequence alignment. Volume 14, Issue 3, pp 290-294, 1998

[43] X. Huang, Bioinformatics: On global sequence alignment. Volume 10, Issue 3, pp 227-235, 1994

[44] E. L. L. Sonnhammer, R. Durbin, Gene: A dot-matrix program with dynamic threshold control suited for genomic DNA and protein sequence analysis. Volume 167, Issues 1-2, pp GC1-GC10, 1995

[45] http://microbes.ucsc.edu/cgi-bin/hgGateway?db=salmTyph, April 2014

[46] http://microbes.ucsc.edu/cgi-bin/hgGateway?hgsid=555757&clade=eukaryota-protista&org=Salmonella+typhimurium+LT2&db=0, last accessed April 2014

[47] S. B. Needleman, C. D. Wunsch, Journal of Molecular Biology: A general method applicable to the search for similarities in the amino acid sequence of two proteins. Volume 48, Issue 3, pp 443-453, 1970

[48] T. F. Smith, M. S. Waterman, Journal of Molecular Biology: Identification of common molecular subsequences. Volume 147, Issue 1, pp 195-197, 1981

[49] I. Holmes, R. Durbin, Journal of computational Biology: Dynamic Programming Alignment Accuracy. Volume 5, Issue 3, pp 493-504, 2009

[50] D. Sankoff, Proceedings of the National Academy of Sciences of the United States of America: Matching sequences under deletion/insertion constraints/ Volume 69, Issue 1, pp 4-6, 1972

[51] A. Gorbenko, Applied Mathematical Sciences: On the Longest Common Subsequence Problem. Volume 6, Issue 116, pp 5781-5787, 2012

[52] W. R. Gilks, Encyclopedia of Biostatistics: Markov Chain Monte Carlo, 2005

[53] P. H. Peskun, Biometrika: Optimum Monte-Carlo sampling using Markov chains, Volume 60, Issue 3, pp 607-612, 1973

[54] M. Katinka, P. Cossart, L. Sibilli, I. Saint-Girons, M. A. Chalvignac, G. Le Bras, G. N. Cohen, M. Yaniv, Proceedings of the Academy of Sciences of the

United States of America: Nucleotide sequence of the thrA gene of Escherichia coli. Volume 77, Issue 10, pp 5730-5733, 1980

[55] http://www.ncbi.nlm.nih.gov/nuccore/GQ323549, last accessed April 2014

[56] http://www.ncbi.nlm.nih.gov/nuccore/GQ894817, last accessed April 2014

[57] http://www.ncbi.nlm.nih.gov/nuccore/GQ365682, last accessed April 2014

[58] http://www.ncbi.nlm.nih.gov/nuccore/GQ329108, last accessed April 2014

[59] http://www.ncbi.nlm.nih.gov/nuccore/GQ323513, last accessed April 2014

[60] http://www.ncbi.nlm.nih.gov/nuccore/GQ866953, last accessed April 2014