Chromosome Descrambling Order Analysis in ciliates

A Thesis Submitted to the
College of Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Nazifa Azam Khan

Permission to Use

Abstract

Ciliates are a type of unicellular eukaryotic organism that has two types of nuclei within each cell; one is called the macronucleus (MAC) and the other is known as the micronucleus (MIC). During mating, ciliates exchange their MIC, destroy their own MAC, and create a new MAC from the genetic material of their new MIC. The process of developing a new MAC from the exchanged new MIC is known as gene assembly in ciliates, and it consists of a massive amount of DNA excision from the micronucleus, and the rearrangement of the rest of the DNA sequences. During the gene assembly process, the DNA segments that get eliminated are known as internal eliminated segments (IESs), and the remaining DNA segments that are rearranged in an order that is correct for creating proteins, are called macronuclear destined segments (MDSs).

A topic of interest is to predict the correct order to descramble a gene or chromosomal segment. A prediction can be made based on the principle of parsimony, whereby the smallest sequence of operations is likely close to the actual number of operations that occurred. Interestingly, the order of MDSs in the newly assembled 22,354 *Oxytricha trifallax* MIC chromosome fragments provides evidence that multiple parallel recombinations occur, where the structure of the chromosomes allows for interleaving between two sections of the developing macronuclear chromosome in a manner that can be captured with a common string operation called the shuffle operation (the shuffle operation on two strings results in a new string by weaving together the first two, while preserving the order within each string). Thus, we studied four similar systems involving applications of shuffle to see how the minimum number of operations needed to assemble differs between the types. Two algorithms for each of the first two systems have been implemented that are both shown to be optimal. And, for the third and fourth systems, four and two heuristic algorithms, respectively, have been implemented. The results from these algorithms revealed that, in most cases, the third system gives the minimum number of applications of shuffle to descramble, but whether the best implemented algorithm for the third system is optimal or not remains an open question. The best implemented algorithm for the third system showed that 96.63% of the scrambled micronuclear chromosome fragments of *Oxytricha trifallax* can be descrambled by only 1 or 2 applications of shuffle. This small number of steps lends theoretical evidence that some structural component is enforcing an alignment of segments in a shuffle-like fashion, and then parallel recombination is taking place to enable MDS rearrangement and IES elimination.

Another problem of interest is to classify segments of the MIC into MDSs and IESs; this is the second topic of the thesis, and is a matter of determining the right "class label", i.e. MDS or IES, on each nucleotide. Thus, training data of labelled input sequences was used with hidden Markov models (HMMs), which is a well-known supervised machine learning classification algorithm. HMMs of first-, second-, third-, fourth-, and fifth-order have been implemented. The accuracy of the classification was verified through 10-fold cross validation. Results from this work show that an HMM is more likely to fail to accurately classify micronuclear chromosomes without having some additional knowledge.

Acknowledgements

It's been a while since I became a graduate student of the prestigious University of Saskatchewan in Saskatoon. I do not think I will ever be able to thank enough my Professor and the team; however I will try to take this opportunity to express my formal and sincere gratitude to all that have helped me come this far.

Achieving a graduate degree from a reputed international university is like a dream come true. Hence, I would like to start by thanking the almighty God, "Allah", for giving me such an opportunity and the ability to conduct my study. I feel greatly honoured and privileged to be able to complete my research and course work for the Master's degree in Computer Science from the University of Saskatchewan, Canada. This journey would not be complete without having the enormous support from my great supervisor, Dr. Ian McQuillan. He is an excellent mentor and a very kind human being, who has given his priceless time to help me accomplish my dream. Being an international student, having been never away from home, the support and kind words from him meant a lot to me during the past two years. I would like to take this opportunity to express my heartfelt thanks and gratitude to him for the guidance and assistance that he has so kindly extended to me throughout the entire period of my study.

I should also like to take this opportunity to express my grateful thanks to my committee members, Dr. Tony Kusalik and Dr. Mark Eramian, and my external examiner, Dr. FangXiang Wu, for their invaluable time, advice, and contributions. I highly appreciate all of your comments and feedback on my thesis.

I would also like to thank my program coordinator, Gwen Lancaster, and all of my lab mates, Dr. Brett Trost, Lingling Jin, Sowgat Ibne Mahmud, Farhad Maleki, Kimberly MacKay, Katie Ovens, Daniel Hogan, and T.M. Rezwanul Islam, for their generous support and cooperation. Their kindness, help, and inspiration have made my stay comfortable in Canada and my work easier and enjoyable. Special thanks to Gwen Lancaster for everything she does for the graduate students.

On a more personal level, I would like to acknowledge the contribution of my mother Anisa Tarafder and father Nazmul Azam Khan. Being an only child, it was painful for them to allow me to come here and do my studies. I am indebted to my in-laws, my grandparents, and all other members of my extended family for their encouragement, support, and blessings. Above all, I must acknowledge, admire, and recognize the support and commitment of my beloved husband Munasir Mahtab Priom throughout this entire period. I am also thankful to all of my friends for their continuous support, and for always being there for me.

Last but not least, I acknowledge with profound thanks the funding support provided by the Department of Computer Science.

Contents

List of Tables

List of Figures

List of Abbreviations

BLAST	Basic Local Alignment Search Tool
Blastn	Nucleotide BLAST
bp	base pair
CIDS	Contiguous Increasing and Decreasing System
CIS	Contiguous Increasing System
DNA	Deoxyribonucleic Acid
E-value	Expect value
HMM	Hidden Markov Model
IES	Internal Eliminated Segment
MAC	Macronucleus
MDS	Macronuclear Destined Segment
MIC	Micronucleus
NCBI	National Center for Biotechnology Information
NIDS	Non-Contiguous Increasing and Decreasing System
NIS	Non-Contiguous Increasing System
nt	nucleotide

Chapter 1

Introduction

Ciliated protozoa are an ancient group of unicellular eukaryotic organisms [38] and an important component to aquatic ecosystems, acting as predators of bacteria and protozoa [35]. Ciliates have survived as a diverse group for approximately two billion years [38, 20, 22] and have many strange structures and functions of interest to both biologists and computer scientists.

Each ciliate cell has two functionally different nuclei; one is called the micronucleus (MIC) and the other is called the macronucleus (MAC). The number of nuclei per cell varies in different species of ciliates. For example, *Oxytricha trifallax* has two micronuclei and two macronuclei, whereas *Sterkiella nova* (formerly *Oxytricha nova*) has four micronuclei and two macronuclei [37, 4]. When two cells mate, they exchange haploid micronuclei, destroy their own macronuclei, and then develop a new macronucleus from the genetic material in the new micronucleus. In the MIC of stichotrichs (a group of ciliates), less than 5% of the DNA actually encodes genes; more than 95% has no known function beyond forming long spacers between genes [38]. Genes are widely spaced along micronuclear chromosomes, separated by the non-genomic (spacer) DNA segments [13]. Certain segments get removed when converting to MAC chromosomes, and certain segments remain in MAC chromosomes. The segments that are removed are called internal eliminated segments (IESs), and the segments that remain are called macronuclear destined segments (MDSs). Figure 1.1 shows that a functional macronucleus can be constructed by deleting IESs and merging MDSs from the micronucleus. An indication as to how ciliates recognize and remove IESs is found in the structure of an MDS at its junction with an IES [38]. IESs and MDSs are flanked by short direct repeats, called pointers or junction sequences [8] at their ends (previous papers have categorized fragments as pointers if they are repeats, including up to one mismatch, adjacent to consecutive MDSs) [10, 25, 38]. These pointers are $2-20$ bp long with one copy of the pointer at the $3'$ end of one MDS and the other copy at the $5'$ end of the next MDS (next MDS according to the correct ordering in the macronucleus) (Figure 1.2) [25]. The process of converting the micronucleus into a macronucleus is known as the gene assembly process in ciliates [38].

Different genera of ciliates perform their gene assembly process in different ways. For example, in the case of *Tetrahymena* and *Euplotes*, the MDSs of the MIC and the MAC are in the same order but the MDSs of the micronucleus are interrupted by IESs [13, 22]. But in the case of stichotrichs, the MDSs are not only interrupted by IESs, but the MDSs can also occur in a different order in the MIC [25]. The genes where this occurs are called scrambled, and they are unscrambled otherwise. It is believed that approximately 30%

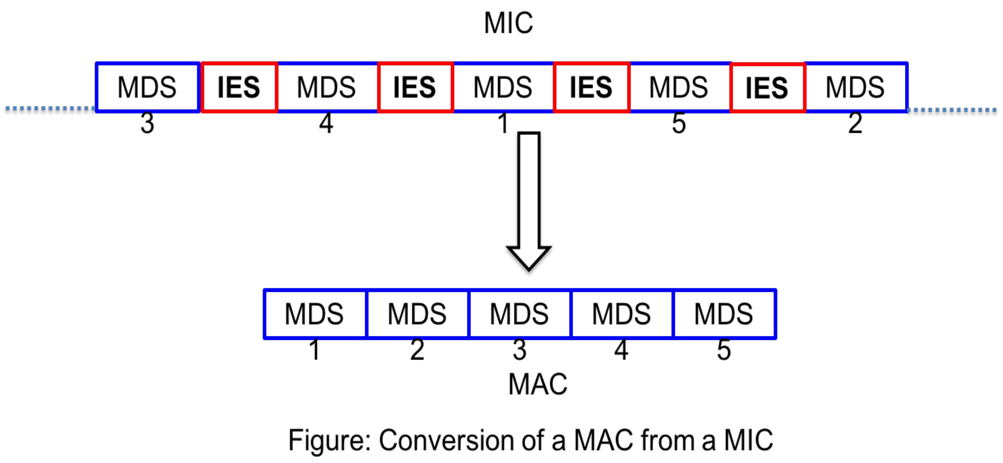


Figure 1.1: Simplified conversion of a MAC chromosome fragment from a MIC chromosome fragment.

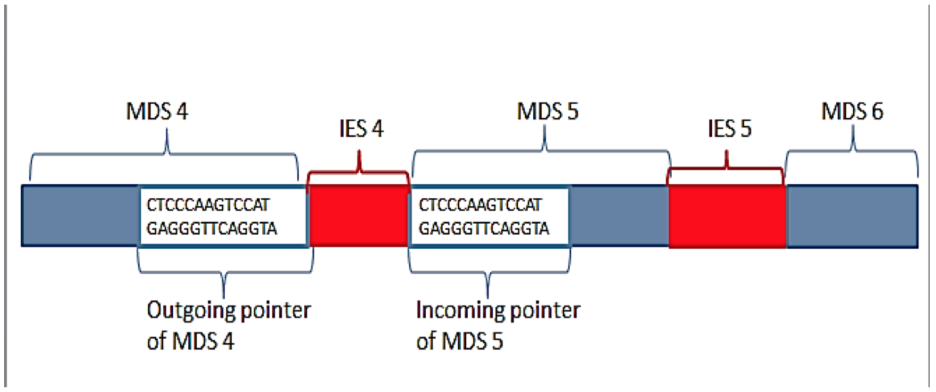


Figure 1.2: Identical outgoing and incoming pointers of two macronuclear destined segments [38].

of the micronuclear genes are scrambled [23]. Hence, gene assembly requires a massive quantity of DNA excisions and rearrangements from the micronucleus [33, 23]. The micronucleus assembly contains 98.9% of all nucleotides in the macronucleus assembly [10]. Previous estimates indicated that the MIC genome consists of somewhere between 2.4% and 18% (generally cited as 5%) of MDSs [34, 10], but more recently Chen et al. inferred that it could be as high as 10% [10]. Scrambled MIC chromosomes are typically longer and contain more MDSs (average 4.9 per kbp) than nonscrambled MIC chromosomes (average 3.7 per kbp), and scrambled MDSs are typically much shorter than nonscrambled MDSs [10]. The smallest MDS and IES can be 0 nucleotides (i.e. the two pointers are adjacent), and most 2 bp pointers are the dinucleotide TA [10].

The process of gene assembly is a splendid example of computing taking place in nature; namely, natural computing [38]. An extensive amount of parallel computation occurs during the gene assembly process. In fact, descrambling MDSs is a computationally hard problem, as even the problem of aligning a micronuclear gene to a macronuclear gene to partition it into segments is an NP-complete problem [22], meaning that very likely no optimal solution can always be found in polynomial time. Knowledge about how nature is solving

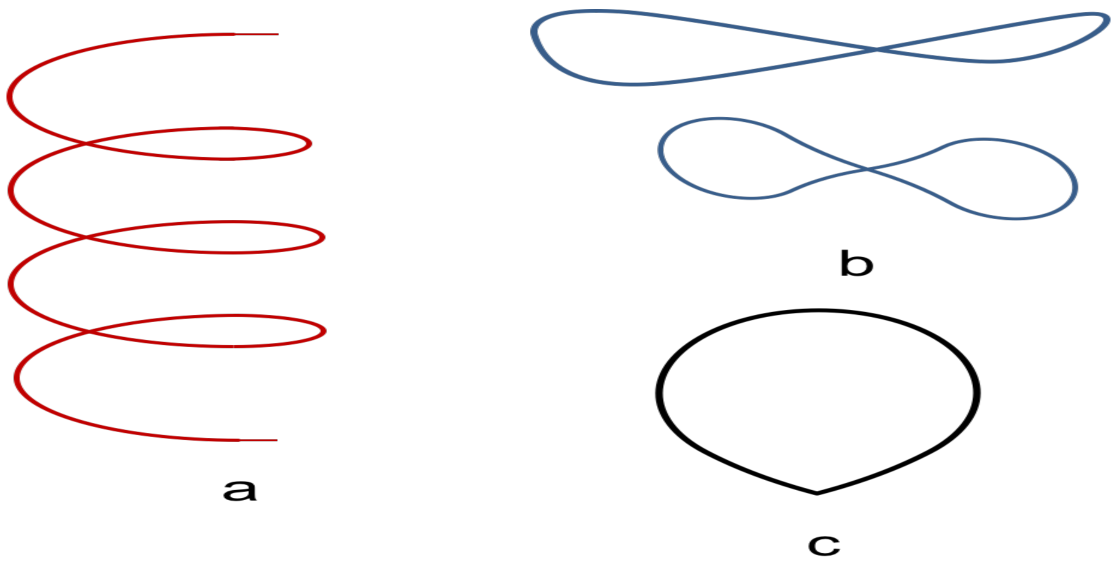


Figure 1.3: Coiled structure (a) lampbrush pattern (b) loop or ring shape (c).

these computationally complex problems may assist computer scientists to construct new algorithms and techniques, and conversely, computational results could be used to infer biological conclusions. Additionally, ciliates have been living on earth for roughly half as long as life has existed on earth, and they are therefore of interest to evolutionary biologists as well.

Researchers have been trying to resolve the mystery of the gene assembly process in stichotrichs ciliates for many years. In 1980, Meyer et al. studied *Stylonychia mytilus* by means of electron microscopy and observed that at the very beginning of the gene assembly process, IESs are eliminated in the form of chromatin rings (loops) [27]. Consequently, their research and results suggested that micronuclear chromatin becomes organised in coiled, lampbrush patterns, or loop-like structures (Figure 1.3) that might be a necessary prerequisite for later IES elimination and MDS rearrangement [28, 29]. Chromatin consists of DNA that is tightly coiled around proteins called histones that condenses to form chromosomes during cell division. In 2008 Matthias et al. concluded that multiple descrambling pathways may produce functional macronuclear molecules [30]. They also found that there are occurrences of multiple parallel inversion and transposition events through each pathways during gene descrambling which results in a permutation of the MDSs [30]. *Inversion* — sometimes referred to as the *reversal* operation — takes a particular segment of MDSs in a MIC gene and puts it back in the opposite direction (therefore, the sequence becomes the reverse complement), whereas the *transposition* operation excises a segment of DNA (an MDS here) and puts it back in a different position (Figure 1.4). In the figure, the numbers represent the order of MDSs in a MIC gene, and negative integers denote that the MDS is inverted (a convention we will adopt in this thesis). A permutation of the MDSs can be the result of multiple inversion and transposition events. Therefore, descrambling of micronuclear genes occurs through multiple pathways, and each pathway consists of a number of parallel inversion and/or transposition events.

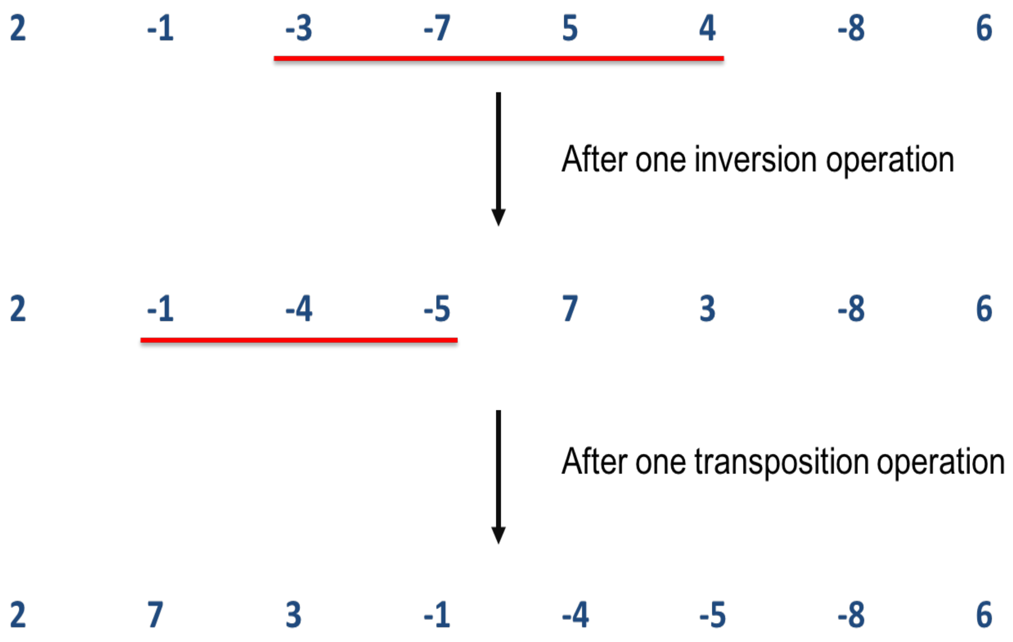

2 -1 -3 -7 5 4 -8 6
After one inversion operation
2 -1 -4 -5 7 3 -8 6
After one transposition operation
2 7 3 -1 -4 -5 -8 6

Figure 1.4: Inversion and Transposition operations.

Interestingly, the order of MDSs in the newly assembled 22,354 MIC chromosome fragments provides evidence that multiple parallel transpositions occur, where the structure allows for interleaving between two sections of the developing macronuclear chromosome in a manner that can be captured with a common string operation called the *shuffle* operation. The images in Figure 1.5 show examples of three micronuclear genes with such behaviour. The shuffle operation on two strings results in the third string by weaving together the first two, while preserving the order within each string. For example, if $a = 2\ 4\ 5\ 6\ 7$ and $b = 1\ 3\ 8\ 9$ are two strings of numbers, then the shuffle of a and b is any permutation of $r = 1\ 2\ 3 \cdots 9$ where the order of the members of a and b is followed in r as well (for example $r = 2\ 4\ 1\ 3\ 5\ 6\ 7\ 8\ 9$). Thus, the result of a shuffle operation between two segments of MDSs is actually a permutation of all MDSs, while preserving the order of the MDSs of the two segments. Interestingly, the shuffle operation can be used to describe certain parallel transposition events. The descrambling scenario of Figure 1.5a can be represented computationally by a shuffle operation between 1 3 5 7 10 12 and 2 4 6 8 9 11 13 14 15 16 17. This is because the MDSs of the macronuclear genes are always in order, and $1\ 2\ 3 \cdots 17$ is one of the results of a shuffle operation between 1 3 5 7 10 12 and 2 4 6 8 9 11 13 14 15 16 17. Similarly, for the second example (Figures 1.5b), a shuffle operation between 1 3 5 7 and 2 4 6 can give the macronuclear MDS order $1\ 2 \cdots 7$, whereas the biological interpretation of this descrambling scenario is multiple parallel transpositions of MDSs $2, 4,$ and 6 (Figures 1.6). The third example is even more complex, but the result can be obtained by splitting the whole sequence into two segments and shuffling them together.

The shuffle operation is nondeterministic, and therefore multiple strings can be in the shuffle of two segments, however it is implied that some structural component allows the developing MAC to fold in such

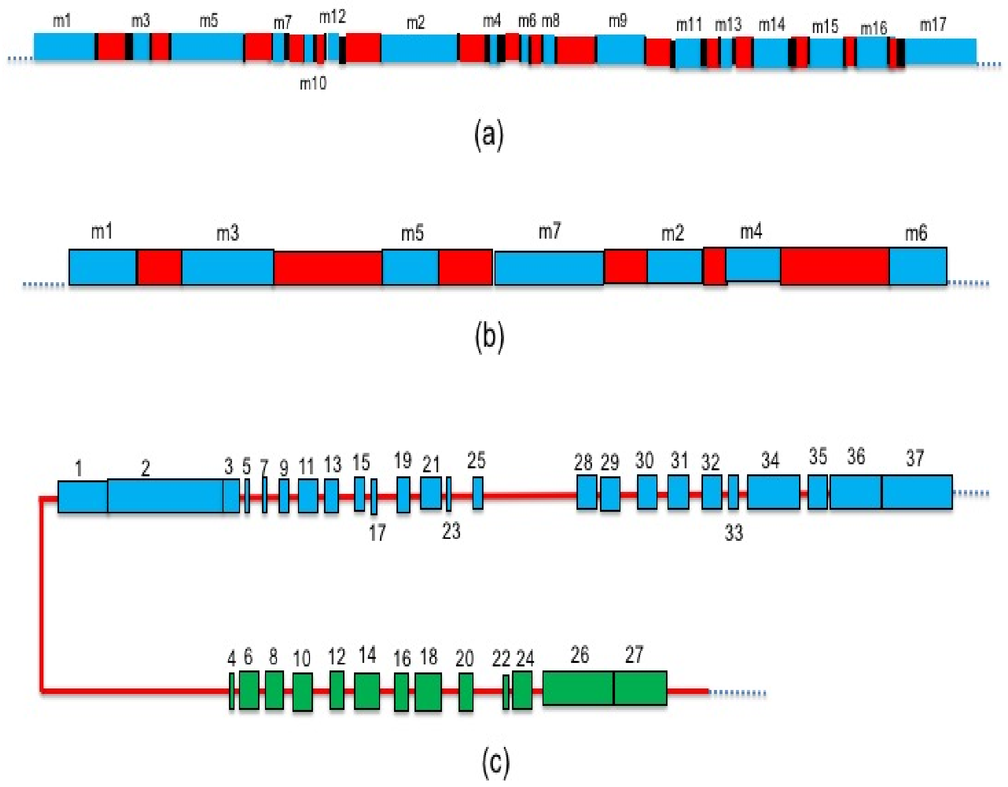


Figure 1.5: MDS organization in micronuclear genes. (a) MDS organization found in the scrambled alpha-telomere-binding protein genes of *Oxytricha trifallax* [30]. (b) MDS organization found in the scrambled alpha-telomere-binding protein genes of *Sterkiella histriomuscorum* [9]. (c) A schematic alignment of the micronuclear genes encoding the large catalytic subunit of DNA-polymerase α in *Oxytricha nova* [23] (inverted MDSs are indicated by the green MDSs).

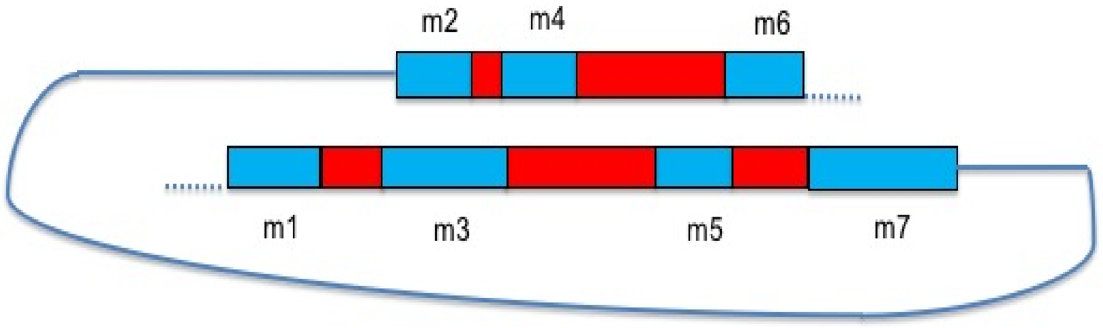


Figure 1.6: Descrambling scenario (a possible descrambling alignment) of the scrambled alpha-telomere-binding protein genes of *Sterkiella histriomuscorum* [9]. Here, a shuffle operation between 1 3 5 7 and 2 4 6 can give the macronuclear MDS order $1\ 2 \cdots 7$, where a biological interpretation of this descrambling scenario can be multiple parallel recombination events for IES elimination and MDS rearrangement.

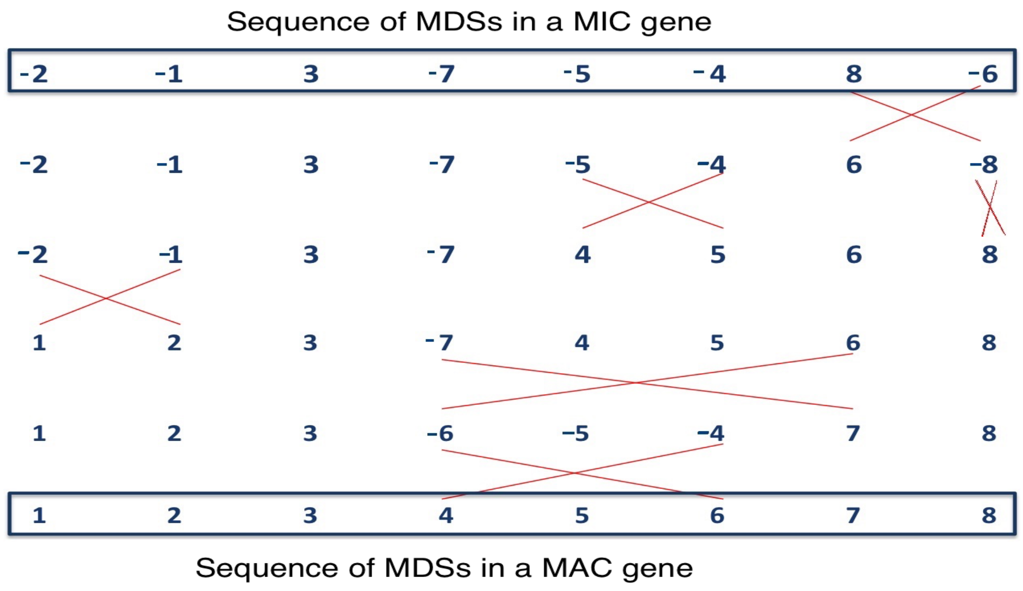


Figure 1.7: Descrambling of a MIC gene with 5 steps.

a way that they can rearrange according to shuffle. Furthermore, the sheer number of genes that can be rearranged with very few applications — as we will see in Chapter 3 — yields evidence that this type of behaviour is occurring.

Since descrambling MDSs of micronuclear chromosomes is a significant phase of gene assembly, one aspect of this research will be predicting the order in which operations are applied to descramble micronuclear chromosomes. For example, both Figures 1.7 and 1.8 show two different scenarios for descrambling the same MIC gene, where Figure 1.7 took five steps (all inversions) and Figure 1.8 took four steps. In the second scenario (Figure 1.8) each set of steps took only one inversion operation, but in the first scenario the MIC gene was descrambled with 6 inversions, where one step has 2 parallel inversion operations. In both of the figures, the first sequence represents the sequence of MDSs in a MIC gene, the final sequence represents the sequence of MDSs in a MAC gene, and the intermediate sequences show a potential descrambling pathway. Hence, a topic of interest is to predict the correct order or pathway to descramble a gene or chromosomal segment, keeping in mind that the exact order that MDSs rearrange is not always the same [30].

We will base this prediction on the principle of parsimony, whereby the smallest sequence of operations is likely close to the actual number of operations that occurred [8, 30, 21, 43]. Pevzner et al. [21] and Glenn Tesler [43] focus on this principle of parsimony for finding the smallest number of genome rearrangements operations to solve the genome rearrangement problem. Genome rearrangement is now a well-studied research problem in bioinformatics, and indeed this problem is quite similar to gene assembly. A genome rearrangement changes the order of genes in a genome, and a series of these rearrangements can alter the genomic architecture of a species [21]. Thus, it is possible to compare the sequence of genes in two different species, and attempt

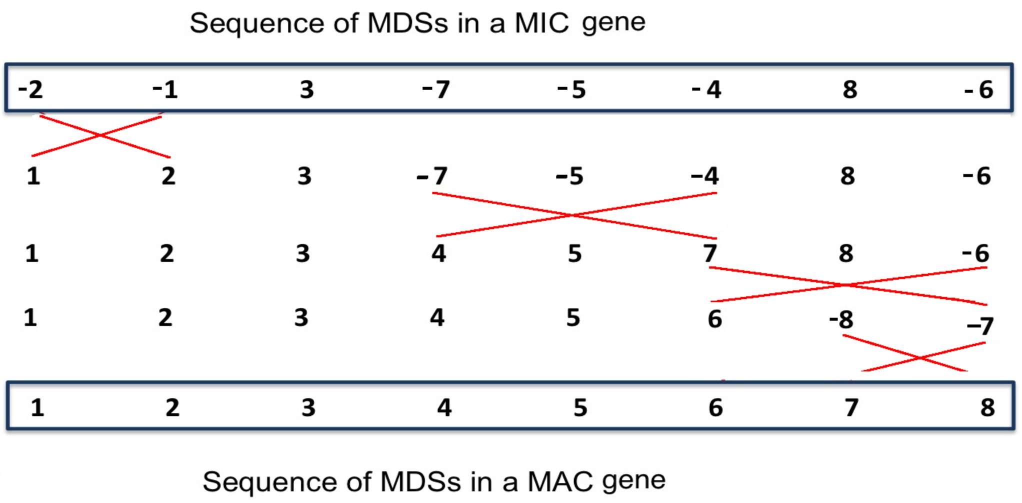


Figure 1.8: Descrambling of the same MIC gene with 4 steps.

to determine the minimum number of changes needed to transform one into the other. This is one measure of distance between species. The genome rearrangement problem tries to determine the genomic distance between species. Moreover, the studies for analyzing the differences between the gene orders of a set of species genomes — genomic distance between species — has been increasingly recognized as a powerful tool in phylogenetic tree reconstruction, as they have helped to gain a better understanding of the evolution of several groups of genomes [19].

This study aims to determine the order of parallel rearrangements by examining the differences between the MDS orders of the MIC and MAC of ciliates similarly using the principle of parsimony; this involves attempting to calculate the minimum number of steps required to descramble each chromosome segment that gets transformed into a macronuclear chromosome. While there is no guarantee that this scenario represents an actual evolutionary pathway, it is more likely than other possibilities using the principle of parsimony, and it also gives a lower bound on the number of operations that could have occurred. In addition, the results of this study gives an idea about the possible parallel operations with respect to inversions and transpositions that might occur during the descrambling. Furthermore, the manner in which a new macronucleus is developed from a new micronucleus is not completely understood. Therefore, finding the most parsimonious descrambling scenario contributes to the knowledge of the gene assembly process in ciliates.

Separately, one problem of interest is to classify segments of the MIC into MDSs and IESs, and the manner in which this occurs is not completely solved yet. Although pointers are involved in gene unscrambling facilitating both removal of IESs and sorting of MDSs, they are short and contain insufficient information to direct accurate elimination on their own. These repeats may satisfy a structural requirement for DNA elimination, but satisfy less of a role in recognition [30]. Recent studies have revealed roles of noncoding RNAs in gene assembly, suggesting that $26-27$ nt small RNAs derived from the micronuclear-limited sequences mark IES regions for elimination in *Tetrahymena* [9, 32]. However, some IESs in *Stylonychia* and *Oxytricha* are smaller than the observed small RNAs [7] and it would be challenging for these small RNAs to mark such IESs precisely. As most pointer sequences are too short to guide accurate IES removal, other factors must assist descrambling [30]. Therefore, in 2007, Mariusz et al. experimentally showed the existence of maternal macronucleus RNA templates that could guide DNA assembly [32]. They found that an RNA or DNA template derived from the maternal macronucleus may be available at a specific stage during development to provide a template for correct and precise DNA rearrangement [30, 32, 25]. Moreover, the proposed use of maternal templates could simultaneously repair imprecise excision and guide unscrambling [30]. Hence, the role of maternal macronuclear templates in detecting IESs is not precise at this stage. Therefore, an interesting question is whether or not macronuclear DNA is needed for classifying micronuclear DNA into segments.

Thus, the other topic in our work is to classify micronuclear DNA sequences into MDSs and IESs without using macronuclear DNA, given appropriate knowledge regarding the sequences. To be more specific, our approach is to use machine learning to partition the nucleotides of a MIC chromosome into MDSs or IESs without employing the corresponding MAC chromosome. Hidden Markov models (HMMs) have been chosen as the underlying machine learning algorithm to solve this problem because they are one of the prominent classifying algorithms for the very similar gene finding problem, and they have also been successful at other biological sequence analysis problems, such as pairwise and multiple sequence alignments, gene annotation, similarity search, and many others [47]. Currently, to identify macronuclear destined segments and internal eliminated segments computationally, both the micronucleus and macronucleus sequences are required as inputs to the existing software that can locate the positions of the MDSs and the IESs. This might indeed be necessary biologically, but an investigation is still worthwhile to provide a type of computational verification that the macronuclear sequence is needed. However, classifying through hidden Markov models for the determination of the locations of MDSs and IESs, allows for the prediction of the segments (the MDSs and IESs) from the micronucleus sequences only. Note that this only addresses the classification of sequences into MDSs and IESs, and not the determination of the orders of the MDSs in scrambled sequences.

In this thesis, a brief overview of the gene assembly process in ciliates, hidden Markov models, and some related definitions are discussed in Chapter 2. Chapter 3 discusses the data collection, and presents some statistics calculated from the micronuclear genome of the ciliate species *Oxytricha trifallax*. This chapter also discusses the preprocessing of the input sequences for the descrambling order analysis, and also the classifying of MDSs and IESs for the machine learning classification problem. Then, Chapter 4 provides a detailed discussion of the descrambling order determination problem, implemented algorithms for determining the descrambling order, and an analysis of the results obtained. Next, a brief description of the implemented HMMs for classifying MDSs and IESs, their accuracy, performance, and effectiveness are presented in the Chapter 5. Finally, Chapter 6 gives an overview of this research work, and discusses possible future work.

Chapter 2

Background

Researchers have been trying to resolve the mystery of the gene assembly process in stichotrichous ciliates for many years. This chapter provides a brief overview of the gene assembly process in ciliates, and discusses fundamental aspects of genome rearrangements, hidden Markov models, and some related definitions.

2.1 Conjugation in Ciliates

During conjugation in ciliates, no offspring are produced; instead, two ciliates exchange genetic material and each then leaves the conjugation process as a genetically new organism [20]. In this process, two cells stick together and form a connecting cytoplasmic channel [38] (Figure 2.1). Then, the diploid micronuclei in each cell undergoes meiosis; this means it contains two copies of each chromosome and creates four haploid daughter micronuclei. In the case of stichotrichs, it has two diploid micronuclei and two macronuclei [13]. Then each of the two mating cells sends a haploid micronucleus to its partner through the cytoplasmic channel [38]. The migrated haploid micronucleus fuses with a stationary haploid micronucleus and forms a new diploid micronucleus [25]. After that, the two cells detach from each other and heal their cell membranes.

2.2 Gene Assembly in Ciliates

The noncoding DNA segments of the MIC genes largely disable the coding ability of the genes. The function of the MAC genes (as with any gene) is to result in RNA or protein synthesis, hence, the MDSs are joined end-to-end to form properly coding genes during the gene assembly process of ciliates [38].

2.2.1 Pointers

According to Ehrenfeucht et al. [14] and Prescott et al. [36], pointers are crucial in guiding the gene assembly process. A pointer is a sequence of nucleotides in the MDSs which is adjacent with IESs. Every two adjacent MDSs have identical pointers. However, there are identical sequences in an MDS and IES that are not pointers, and therefore, repeat sequences alone are not sufficient to guide alignment. But, it is possible that some structural components align the two real pointers for the excision of an IES. Pointers reside at the two ends of every MDS, excluding the first and last one. The pointers at the starting and ending points of

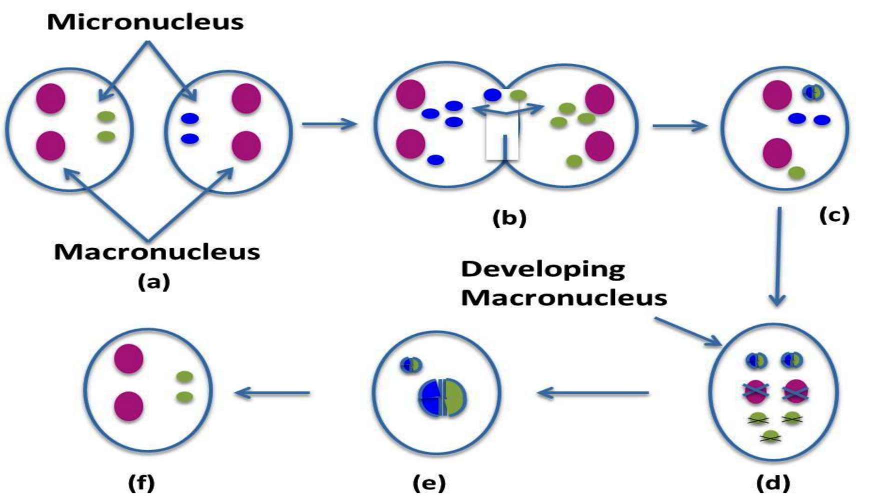


Figure 2.1: Conjugation in ciliates [38]. (a) Two ciliates before mating. (b) Two ciliates connected by a cytoplasmic channel. (c) After separation from a partner with a new diploid micronucleus (half green, half blue). (d) The new diploid micronucleus divided into its haploid daughter and the unused micronuclei degenerate. (e) One of the new daughters MIC has developed into a MAC. (f) At the end of the conjugation the MIC and MAC have divided.

an MDS are called the incoming and outgoing pointers respectively. One MDS will join another only if the outgoing pointer of the first MDS matches with the incoming pointer of the second MDS [38] (Figure 2.2). Thus, the first MDS has only an outgoing pointer, and the last MDS has only an incoming pointer. Although these sequences probably have a function in descrambling the MDSs, the repeat sequences are present not only as part of pointers, but also copies of these sequences can appear throughout the MIC [8]. These pairs of short direct repeats (average repeat length 4 bp between nonscrambled segments, 9 bp between scrambled segments [30]) are present at consecutive MDS-IES junctions, and one copy of each pointer is retained in the MAC.

2.2.2 Intramolecular Model

A primary theoretical model for gene assembly was introduced by Prescott et al. [36] and Ehrenfeucht et al. [14] in 2001. It consists of three unary molecular operations based on pointers (Figure 2.3). For excision of IESs and joining of the MDSs, the DNA is believed to fold in such a manner that the identical incoming and outgoing pointers align in parallel. Then a switching enzyme called a recombinase cuts the two aligned repeats and rejoins them in a switched configuration (a process known as recombination), which regenerates two hybrid copies of the pointer [38]. The one copy of the pointer leads to a linear molecule by joining the MDSs, and the other hybrid copy becomes a part of a circular IES, which is soon destroyed. According to the intramolecular model, the gene assembly is accomplished by the following three operations.

- **Loop Excision:** The molecules fold into a loop in such a way, that the two copies of pointers are aligned [38, 25] (Figure 2.3a). This operation occurs when a micronuclear molecule contains a pair of

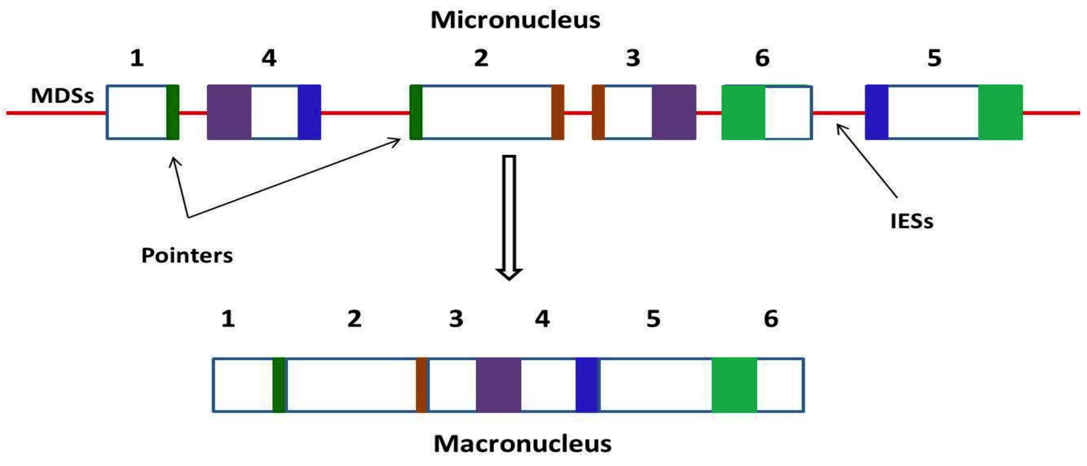


Figure 2.2: Pointers in the MIC gene and MAC gene.

pointers flanking an IES.

- **Hairpin Recombination:** This applies to a molecule that contains one or more pairs of pointers in which one pointer is the inversion of the other [38]. In this case, the molecule folds into a hairpin shape in such a way that the two copies of the pointer become aligned in the same direction (Figure 2.3b), hence the two pointers can recombine [38, 25].

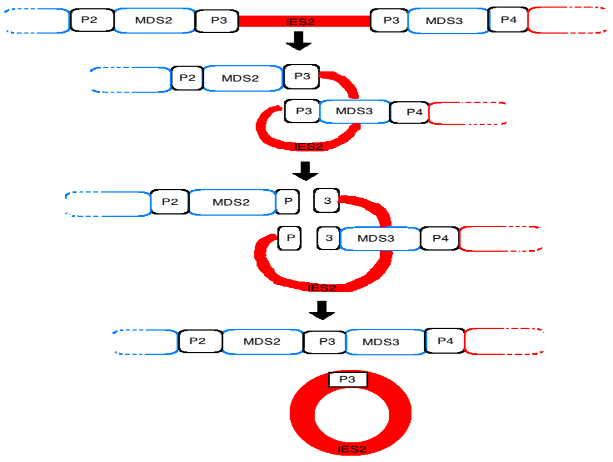
- **Double loop:** When a molecule has two pairs of pointers and a segment of the gene confined by one pair of pointers overlaps with the segment confined by the other pair of pointers, then the double loop operation can take place [38]. Here, the molecule folds into two loops in such a manner that one loop is comprised of one pair of pointers and the other loop has the other pair of pointers [38, 25, 20] (Figure 2.3c).

2.3 Existing Bioinformatics Tools for determining MDSs and IESs

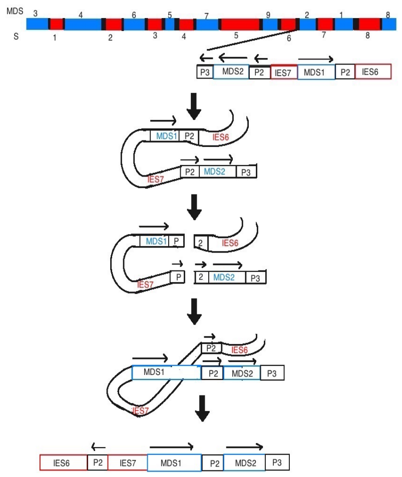
There are several bioinformatics tools for different purposes that can be used to assist researchers to study and analyze distinct ciliate-specific computational biology problems. Currently, classifying a micronuclear gene sequence into macronuclear destined segments and internal eliminated segments can be done in two ways: one is by determining multiple local alignments (with an alignment tool such as nucleotide BLAST) between the two sequences, and then using the regions of local similarity to infer the MDSs by custom scripts [10], and the other one is through an online program, named Gene Unscrambler [8].

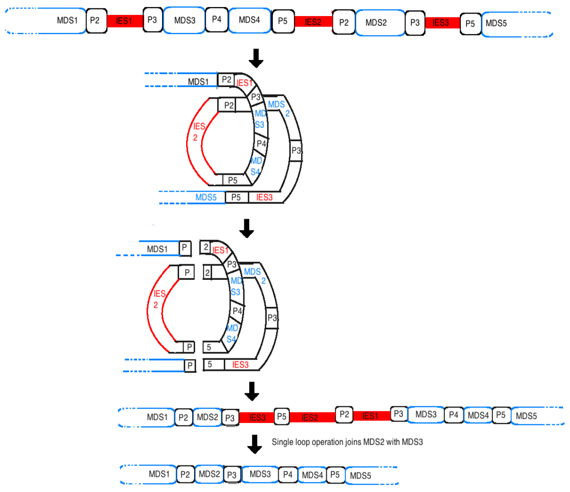
2.3.1 Sequence Alignment

There are many sequence alignment programs that exist that could be used to find local matches between a micronuclear and macronuclear sequence. However, we will focus here on the use of the tool BLAST to do



(a) Loop Excision.





(b) Hairpin Recombination. **(c)** Double Loop Recombination.

Figure 2.3: Intramolecular Model

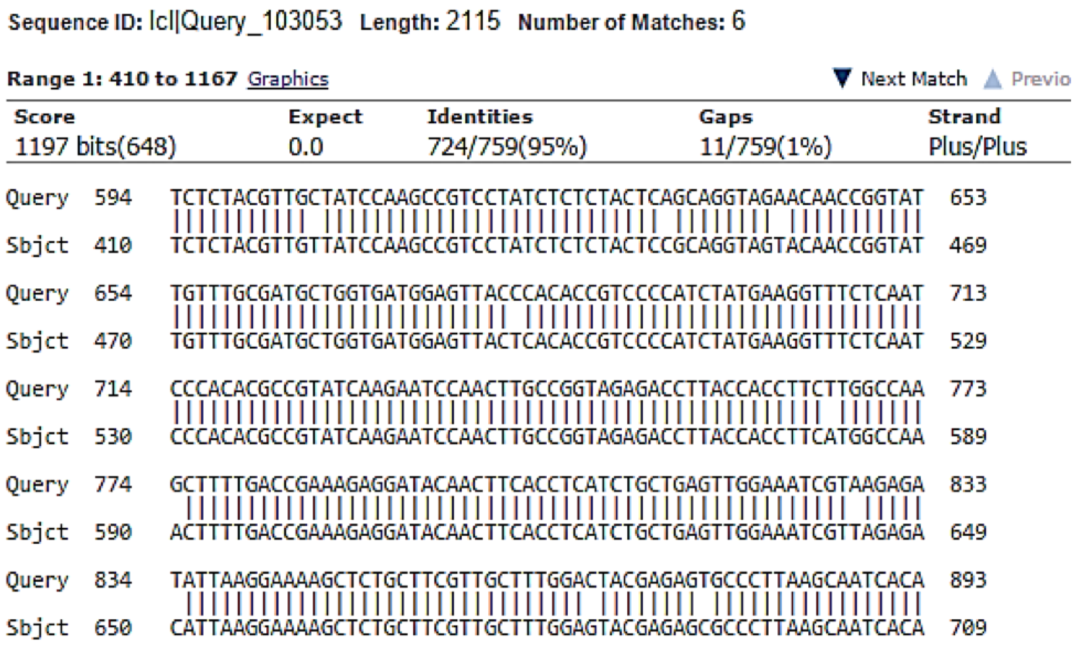
this task to match the use of this task that has been done in [10].

The Basic Local Alignment Search Tool (BLAST) finds regions of local similarity between sequences. BLAST is a set of programs provided by the National Center for Biotechnology Information (NCBI) for aligning nucleotide or protein sequences to other sequences, or those present in a selected target database, and it calculates the statistical significance of matches. NCBI is the world's largest repository of databases relevant to biotechnology and biomedicine, and an important resource for bioinformatics tools and services. Nucleotide BLAST, or blastn, compares a nucleotide query sequence against a nucleotide sequence database, and returns alignments between the given query and target sequences with the match score, matching coordinates, percentage identity (to show the degree of matching), and the expected value (E-value), which describes the number of hits one can "expect" to see by chance when searching a database of a particular size. The lower the E-value, or the closer it is to zero, the more "significant" the match is. BLAST can be used to infer functional and evolutionary relationships between sequences as well as to identify members of gene families. Thus, blastn with a MIC and MAC sequence determines the near-identical sections between these two sequences.

In particular, blastn can take a MIC and MAC gene pair as input; then it aligns these two sequences using a pairwise sequence alignment algorithm and returns potentially multiple alignments between the given MIC and MAC sequences along with the match score, percentage identity, and E-value of each. Thus, an alignment returned by blastn indicates precisely each portion of the MAC gene that matches with a portion of the MIC gene by indicating the coordinates of the matching segments in the MIC and MAC gene (Figure 2.4). Through summarizing these alignments, the MDSs can be predicted, and the unmatched parts of the MIC can be classified as IESs.

2.3.2 Gene Unscrambler

In 2004, software was released to analyze MIC and MAC genes [8]. It automatically aligned the macronuclear and micronuclear forms of a gene, outputting the location of the macronuclear destined segments, internal eliminated segments, and pointer sequences. This tool was accessible through the mds_ies_db database [6], but recently they updated the tool and now it does not allow to give input sequences. Instead, it functions purely as a pre-populated database whereby one can search the database through the micronuclear and macronuclear contigs and gene names (contigs are predicted fragments of chromosomes over the four nucleotides "A, C, G, and T"). After requesting a search, it returns information such as the searched item's length, its type (MIC or MAC), its DNA sequence (a FASTA file), locations of the genes residing in it, the number of MDSs it has, and the number of MAC matches (for MIC contigs) or MIC matches (for MAC contigs). This program also provides two types of information with visual representations; one is a diagram of the MIC-MAC contig mapping (matching segments of the MIC or MAC contig with existing MAC and MIC contigs in the database respectively) with the positions of the genes in it, and the other is either the coordinates of the pointers and MDSs for a specific MAC contig, or coordinates of IESs and MDSs for a specific MIC contig through a table.



```
Sequence ID: lcl|Query_103053  Length: 2115  Number of Matches: 6

Range 1: 410 to 1167  Graphics                                  ▼ Next Match  ▲ Previo
Score              Expect      Identities          Gaps              Strand
1197 bits(648)     0.0         724/759(95%)        11/759(1%)        Plus/Plus

Query  594    TCTCTACGTTGCTATCCAAGCCGTCCTATCTCTCTACTCAGCAGGTAGAACAACCGGTAT  653
              ||||||||||| |||||||||||||||||||||||||| |||||||| ||||||||||||
Sbjct  410    TCTCTACGTTGTTATCCAAGCCGTCCTATCTCTCTACTCCGCAGGTAGTACAACCGGTAT  469

Query  654    TGTTTGCGATGCTGGTGATGGAGTTACCCACACCGTCCCCATCTATGAAGGTTTCTCAAT  713
              ||||||||||||||||||||||||||| ||||||||||||||||||||||||||||||||
Sbjct  470    TGTTTGCGATGCTGGTGATGGAGTTACTCACACCGTCCCCATCTATGAAGGTTTCTCAAT  529

Query  714    CCCACACGCCGTATCAAGAATCCAACTTGCCGGTAGAGACCTTACCACCTTCTTGGCCAA  773
              |||||||||||||||||||||||||||||||||||||||||||||||||||| |||||||
Sbjct  530    CCCACACGCCGTATCAAGAATCCAACTTGCCGGTAGAGACCTTACCACCTTCATGGCCAA  589

Query  774    GCTTTTGACCGAAAGAGGATACAACTTCACCTCATCTGCTGAGTTGGAAATCGTAAGAGA  833
               ||||||||||||||||||||||||||||||||||||||||||||||||||||| |||||
Sbjct  590    ACTTTTGACCGAAAGAGGATACAACTTCACCTCATCTGCTGAGTTGGAAATCGTTAGAGA  649

Query  834    TATTAAGGAAAAGCTCTGCTTCGTTGCTTTGGACTACGAGAGTGCCCTTAAGCAATCACA  893
               ||||||||||||||||||||||||||||||| ||||||||| |||||||||||||||||
Sbjct  650    CATTAAGGAAAAGCTCTGCTTCGTTGCTTTGGAGTACGAGAGCGCCCTTAAGCAATCACA  709
```

Figure 2.4: Blastn report between a pair of MIC and MAC gene sequences.

2.4 Hidden Markov Models

Often, biological sequence analysis involves classifying the nucleotides/residues of a sequence in various ways [12]. For such problems, Hidden Markov Models (HMMs) have been used extensively, for example, in finding periodicities in DNA, for exploring structural similarities of families of genes, for producing multiple sequence alignments, for finding palindromic repeats, and for protein secondary structure prediction [17]. Hidden Markov models are a formal foundation for making probabilistic models of linear sequence 'labelling' problems [12]. They are widely used to define class-conditional densities inside a generative classifier [31].

A hidden Markov model is a statistical model for representing probability distributions over sequences of observations. With an HMM, a sequence of emissions is observed but the sequence of states the model went through to generate the emissions is not known. Hidden Markov models are a particular kind of Bayesian Network [15]. Hence, it calculates the transition and emission matrices by applying Bayesian probabilistic theorems on the observed sequences of emissions and states. The transition matrix represents the probability distribution of changing states, whereas the emission matrix denotes the probability of each occurring from each state. It also calculates the initial matrix which shows the probability distribution for each state to be the initial state. Then, given the transition and emission matrices, often the Viterbi algorithm is used with HMMs to compute the most likely sequence of states the model would go through to generate a given sequence of emissions.

The order of a Markov model of fixed order is the length of the history or context upon which the probabilities of the possible values of the next state depend. A first-order hidden Markov model assumes that

)

the probability of a certain observation at time t only depends on the observation of time $t-1$. Similarly, a second-order HMM assumption would have the probability of an observation at time t depend on the observation of time $t-1$ and time $t-2$. It has been seen that in computational biology studies, higher order HMMs contribute more in terms of accuracy [5].

Of particular interest, HMMs have been widely used for gene predicting, which involves classifying a sequence segment into coding and non-coding regions [5]. Gene prediction programs typically use hidden Markov models to implement mathematical models of biological signals such as splice sites or the translation start and end points [5, 41]. Within genes, the main coding and non-coding regions of a gene are known as exons and introns, respectively. (Predicting exons and introns of a gene is analogous to determining MDSs and IESs in ciliates). Several recent gene-finding methods use Markov models of coding and noncoding regions in order to classify sequence segments as either exons or introns [5]. In the cases of predicting genes, introns, exons, and classifying splice sites, some predefined knowledge is known, which helps the HMM to predict and classify most accurately [41, 17]. Different studies use this knowledge differently and end up with different HMMs of different order at different levels (for example, many gene classification programs use one HMM at first level for determining start codons and another HMM at the end for determining stop codons for predicting genes — the start codon marks the site at which protein coding sequence begins, and the stop codon marks the site at which the protein coding sequence ends) [5, 41, 17, 1]. The parameters for such models are typically estimated using the maximum likelihood method; that is, by using the observed conditional frequencies of an appropriate training set of known genes to estimate the corresponding conditional probabilities [5]. Identifying CpG Islands (regions where we see many more CG dinucleotides than elsewhere in a DNA sequence) through HMMs is one of the significant and prominent studies of computational biology [46]. A recent study using a second-order HMM to produce simulated DNA data using frequency distributions of CpG island showed highly accurate predictions of the pre-specified CpG islands, and confirms that an HMM can be an accurate predictive tool [2]. However, it was also found that the outcome of a HMM is highly sensitive to the setting of the initial probability estimates. In 2014, another study recognized that the training inconsistency observed in [2] is not a consequence of their general method, and is instead a consequence of initial parameter estimates, as initial parameter estimates significantly impact the results produced by HMMs for CpG island prediction [18]. Recently, there have been many software packages developed that use HMMs to classify and predict the intergenic and intragenic regions of a DNA sequence [5, 1]. GeneMark, GLIMMERHMM, GENSCAN, BGF, FGENESH etc. are some examples of bioinformatics tools that implement HMMs to solve different computational biology problems.

2.5 Genome Rearrangements

Two genomes (from any organism, not just ciliates) may have many genes in common, but the genes may be arranged in different sequences, or be moved between chromosomes, or be reversed [21, 42]. Such differences in gene orders are the results of rearrangement events that occur throughout molecular evolution, and their study is known as genome rearrangements in the field of bioinformatics.

2.5.1 Genome Rearrangement Events

While comparing entire genomes from different species, researchers often want to compute a form of distance between them using a set of basic genome rearrangement operations. There are in general many such scenarios, but no computational way to determine the actual one that occurred during evolution without additional data. Biologists are mostly interested in the most parsimonious evolutionary scenario, which means the scenario involving the minimum number of evolutionary changes. This is because the genomic distance with the minimum number of genome rearrangement events is usually quite close to the actual number of rearrangement steps that occurred, provided that the number of steps is much smaller than the number of genes [43]. Despite having many other genome rearrangement events, researchers often focus only on the reversals [21], as it is the most common evolutionary event among affecting gene order [3]. For example, chromosome X in mice can be transformed to chromosome X in human using seven reversals [42] (Figure 2.5). (This reversal operation is the same as an inversion; this community tends to use the word reversal.) This operation cuts out a DNA segment and puts it back in reverse orientation; that is $XYZ \to XY^rZ$ (where Y^r is the reverse complement of Y) [21, 3] (Figure 2.6). Another common genome rearrangement event is *transposition*, which cuts out a DNA segment and inserts it into another location (i.e. $UVWXY \to UXVWY$ — transposition of a DNA segment, that is denoted by X) [42] (Figure 2.7). In genome rearrangements, the capital letter labels, $A, B, \ldots, L$, of Figures 2.6 and 3.2 are the sequences of genes, whereas for ciliates, we use them to represent the sequences of MDSs.

2.5.2 Permutations

The order of the genes in a genome can be described by a permutation. In these problems, it is customary to denote the genes by numbers $1, 2, \ldots, n$, and the order of genes in two organisms is represented by permutations $\pi = \pi_1\pi_2\pi_3 \cdots \pi_n$ and $\sigma = \sigma_1\sigma_2\sigma_3 \cdots \sigma_n$, where the same gene in the two permutations is given the same number. In such a situation, it is always possible to renumber the permutations so that one of them is the identity permutation. For example, the permutation of a genome in Figure 2.8 has been represented by 1 2 3 8 7 6 5 4 9 10. The reading direction of the genes is represented by signs, for example, the inversely oriented genes are denoted by negative integers (Figure 2.8b). Figure 2.8a shows the orders of the genes without considering their reading directions (signs), while Figure 2.8b shows the genes with their orientations, or reading directions. When the reading directions of the genes are considered, they are called the directed permutations (Figure 2.8b), and undirected permutations otherwise (Figure 2.8a).

The *reversal* operation, $r(i, j)$, has the effect of reversing the order of genes between the ith number and the jth number. Suppose, $\pi = \pi_1 \cdots \pi_{i-1}\overrightarrow{\pi_i\pi_{i+1} \cdots \pi_{j-1}\pi_j}\pi_{j+1} \cdots \pi_n$ (here, the arrow is just showing the section that will be reversed), then a reversal $r(i, j)$ will turn π into $\pi' = \pi_1 \cdots \pi_{i-1}\overleftarrow{\pi_j\pi_{j-1} \cdots \pi_{i+1}\pi_i}\pi_{j+1} \cdots \pi_n$ [21].

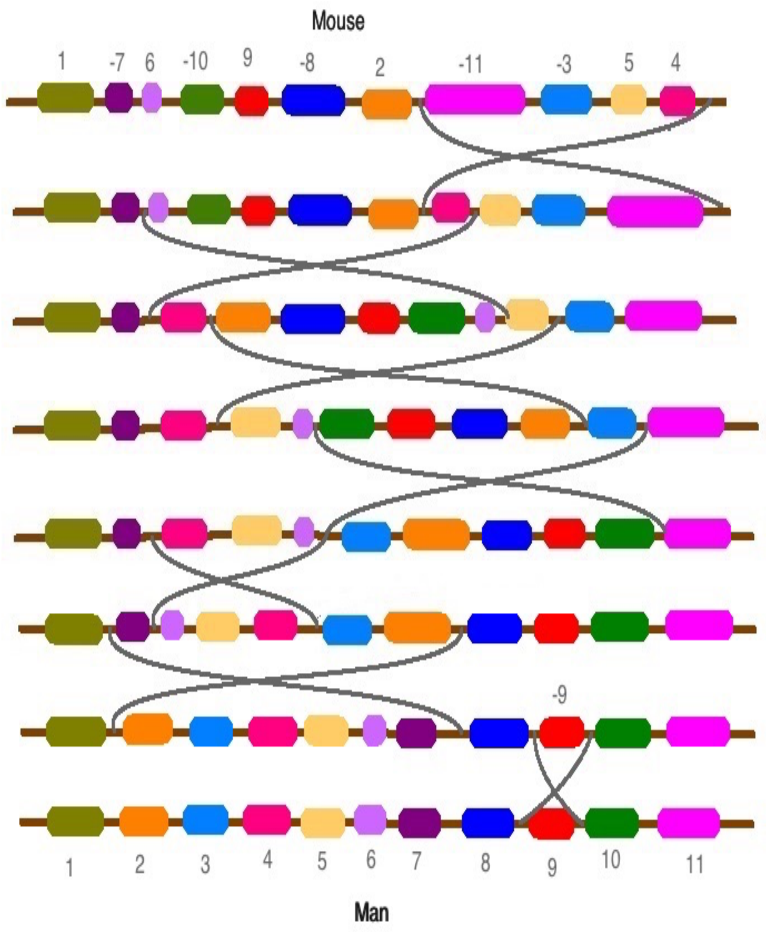


Figure 2.5: Conversion of chromosome X of mice into chromosome X of human with 7 reversals [42].

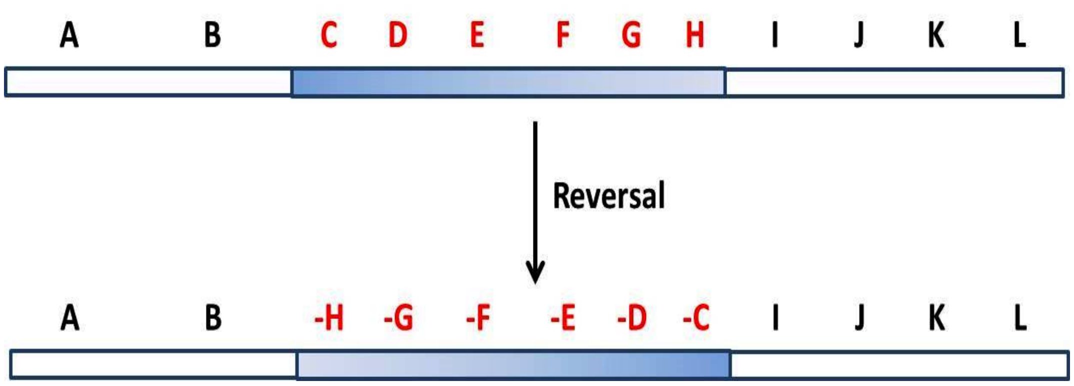


Figure 2.6: Reversal operation.

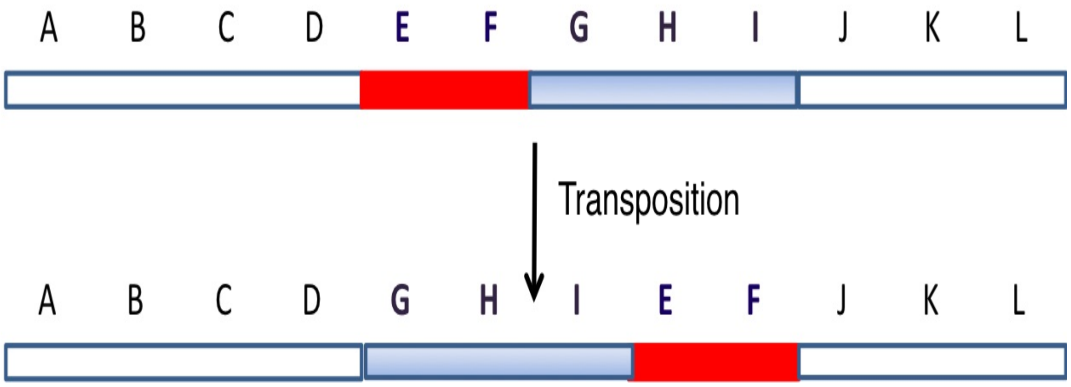
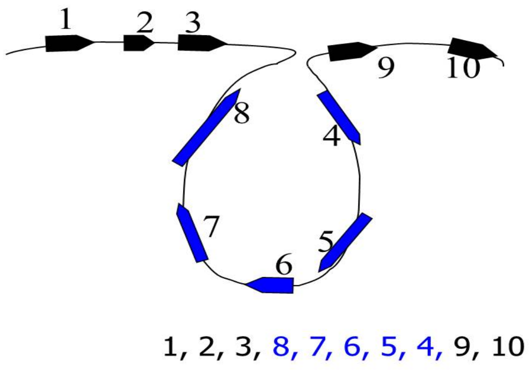
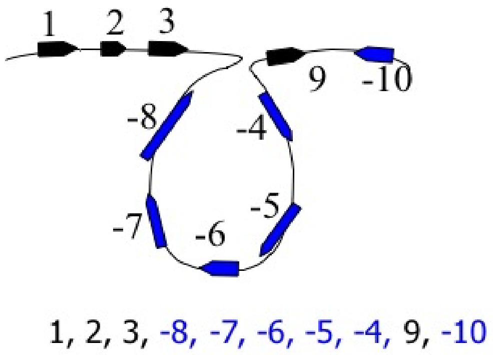


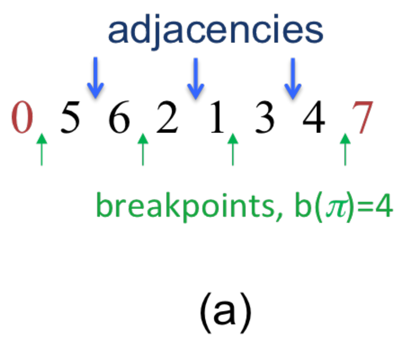
Figure 2.7: Transposition operation.





(a) (b)

Figure 2.8: Permutations in a genome, represented using unsigned (a) and signed (b) permutations.



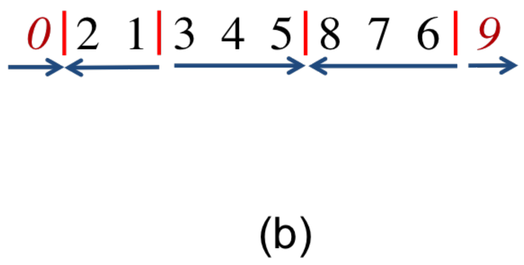


Figure 2.9: (a) Adjacency and Breakpoints (b) Strips.

2.5.3 Adjacency, Breakpoints, and Strips

For a permutation $\pi = \pi_1\pi_2\pi_3.....\pi_n$, this is extended to an *extended permutation* by $\pi_0 = 0$ at the first position of π and $\pi_{(n+1)} = n+1$ at the last position of π. The extended permutation of a permutation π is denoted by $ext(\pi)$. A pair of neighbouring elements π_i and $\pi_{(i+1)}$, for $0 \leq i \leq n$, is called an *adjacency* if π_i and $\pi_{(i+1)}$ are consecutive numbers (i.e. consecutive elements π_i and π_{i+1} are adjacent if $|\pi_i - \pi_{i+1}| = 1$), and a *breakpoint* otherwise [21]. The number of breakpoints in π is denoted by $b(\pi)$. In Figure 2.9a, the extended permutation $\pi = 0\ 5\ 6\ 2\ 1\ 3\ 4\ 7$, π has three adjacencies 5 6, 2 1, 3 4 and four breakpoints 0 5, 6 2, 1 3, 4 7. For extended permutations, only the extended identity permutation sequence has no breakpoints. For non-extended permutations, there are two (the identity plus the reverse of the identity) [3].

A *strip* in an extended permutation π is an interval between two consecutive breakpoints; that is, it is any maximal segment without breakpoints [21] (Figure 2.9b). A strip is called increasing if its elements are increasing, otherwise the strip is decreasing. It is convenient to consider a single-element strip as decreasing with the exception of the strips π_0 and $\pi_{(n+1)}$, which will be defined as always increasing. For example, the permutation 0 2 1 3 4 5 8 7 6 9 has two decreasing strips (i.e. 2 1, and 8 7 6) and three increasing strips (i.e. 0, 3 4 5, and 9). The arrows of Figure 2.9b denote the increasing (with a right arrow) and decreasing strips (with a left arrow).

2.5.4 Genome Rearrangements and Descrambling Order in Ciliates

Given a set of genomes, the key problem of genome rearrangement is to find out a shortest set of events transforming one genome into one another. It is assumed that this shortest set of events is very close to the actual number of events that occurred throughout evolution, provided the number of steps is much smaller than the number of genes [21, 43]. Hence, this inspired many researchers to study the optimal transformation scenario and there are a number of existing genome rearrangement software packages (GRAPPA, webMGR, Itsik Mantin's applet, Tesler's GRIMM [43] etc.) to predict a shortest set of events that transforms one genome into another.

On the other hand, the descrambling order analysis problem is to determine the minimum number of steps required to transform a MIC chromosome into a MAC chromosome and similarly, it is assumed that this minimum set of operations is very close to the actual number of events that occurs during the gene assembly process [30, 8]. Thus, genome rearrangements study gene order, whereas descrambling order analysis studies the parallel steps to descramble genes. As a result, genome rearrangement algorithms and parameters play an important role in determining descrambling order algorithms.

2.6 Concatenation and Shuffle Operations

Concatenation and shuffle are two important word operations in computer science. The concatenation operation joins two sequences of numbers (or characters; we will only use it on numbers) as follows: given two sequences $\pi = \pi_1\pi_2 \cdots \pi_n$ and $\theta = \theta_1\theta_2 \cdots \theta_m$, the concatenation of π and θ, denoted by $\pi\theta$ is $\pi_1 \cdots \pi_n\theta_1 \cdots \theta_m$. For example, if $a = 2\ 4\ 5\ 6$, and $b = 1\ 3\ 8\ 9$ are two strings of numbers then after concatenating a and b the resultant sequence will become 2 4 5 6 1 3 8 9.

Let u and v be two sequences of integers. Then the shuffle of u and v is the set

$$u \sqcup\!\sqcup\, v = \{x_1y_1x_2y_2 \cdots x_ry_r \mid u = x_1x_2 \cdots x_r, v = y_1y_2 \cdots y_r\}.$$

In this definition, each x_i and y_i is a (potentially empty) segment of integers rather than a single integer. In our case, we will only be examining the shuffle of integer sequences where each integer only occurs at most once in both lists. Hence, each word w in the shuffle of u and v is actually a partial permutation (meaning it can be missing some integers from being a permutation) of all elements of u and v, while preserving the order of the elements of u and v in w. Note that $|w| = |u| + |v|$ (where $|u|$ is the length of u) is a necessary condition for the existence of a shuffle. For example, $u = 1\ 3\ 4\ 5\ 8$ and $v = 2\ 6\ 7\ 9$, then one string w in the shuffle of u and v is 1 2 3 4 6 7 9 5 8 and another is 1 2 3 4 5 6 7 8 9, but not 1 4 2 7 3 8 5 9 6. This means w can be any permutation of $1\ 2\ 3 \cdots 9$ that preserves the order of the elements of u and v.

2.7 Subsequences, Subwords, and Segments

Given two words u, v, then u is a *subsequence* of v if $v = v_0u_1v_1u_2v_2 \cdots u_nv_n$, where $u = u_1u_2 \cdots u_n$. So, u does not need to be contiguous in v. Given two words u, v, u is a *subword* of v if $v = xuy$, for some x, y. In this case, u needs to be contiguous in v. We use the word *segments* as a generalized term, to indicate either subwords, or subsequences. Thus, a segment u of v would be either a subword of v, or a subsequence of v.

2.8 Graph Theory

Graph theory is a well-studied topic of computer science and mathematics involving graphs. *Graphs* are the mathematical structures that can be used to model the pairwise relationships between objects. Graphs are incredibly useful structures in computer science — they arise in all sorts of applications, including scheduling, optimization, communications, the design and analysis of algorithms, data mining, image segmentation, clustering, image capturing, and networking [40]. A *directed graph* is a graph, where all the edges are directed from one vertex to another. That is, a directed graph $G = (V, E)$, consists of a set of vertices V, a set of edges E, where E is a subset of $V \times V$, and an edge (u, v) represents a directed edge from u to v [44]. A *directed acyclic graph* is a directed graph with no cycles [44], which means there is no way to start at some vertex v and follow a sequence of edges that eventually loops back to v again. A *walk* is any route through a graph from vertex to vertex along edges, thus a walk is an alternating sequence of vertices and connecting edges [44]. A *path* is a walk that does not include any vertex twice, except that its first vertex might be the same as its last [44]. Therefore, a *longest path* from source to destination in a graph is a simple path of *maximum length* from source to destination [44].

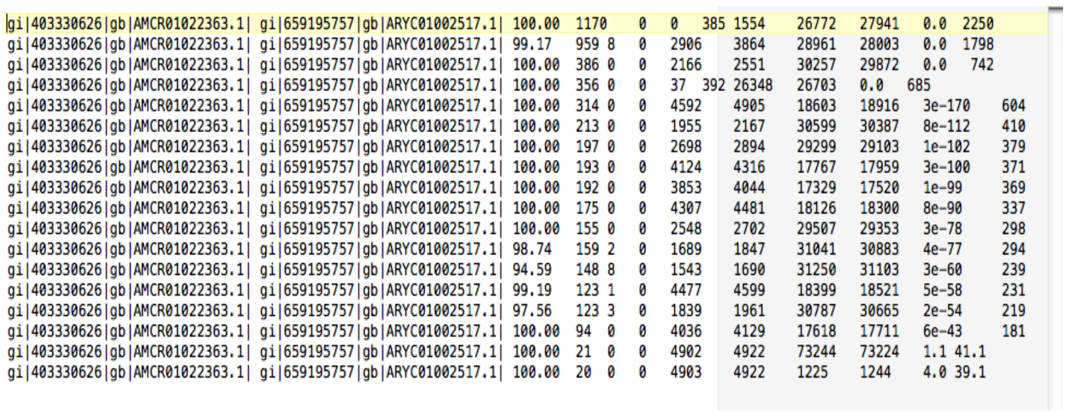
Chapter 3

The Ciliate Genome, and Preliminary Data Processing

This chapter discusses data collection, and then presents some statistics calculated from the ciliate genome. This information is of value from the perspective of understanding the ciliate genome. In addition, it is a necessary preprocessing step for the remaining portions of this thesis.

The data used for this thesis was raw genome data from the ciliate species *Oxytricha trifallax.* The data was retrieved from NCBI on May 20, 2015 in the form of 22,363 macronuclear contigs and 25,720 micronuclear contigs. A contig is a contiguous sequence of DNA created by repeatedly assembling overlapping sequenced fragments of a chromosome. These contigs needed further processing, as the input needed to analyze the chromosome descrambling order would be the order of MDSs on the micronuclear chromosomes. The procedure for determining the order of MDSs on the micronuclear chromosomes was chosen to be the same as the methods used in the Chen et al. paper [10] for the same purpose, so that comparisons could be made. The 22,363 macronuclear contigs were aligned against a database of 25,720 micronuclear contigs by using Nucleotide BLAST (blastn, discussed in Section 2.3.1). Parameters of [-ungapped -word_size 20 -outfmt 10] were used. The parameter 'ungapped' means the sequence alignment would be an ungapped alignment, 'word_size' 20 indicates to blastn that the length of the initial exact match would be 20, and 'outfmt' 10 denotes that the generated outputs will be in comma-separated format. Ungapped alignments do not allow any insertion or deletion of nucleotides in the process of generating the alignment. A gap is generated when a nucleotide has either been deleted or inserted in the sequence. Indeed, these parameter values are identical to those used in [10].

From the blastn report generated, the MAC/MIC contigs having the alignments with the lowest E-value (Expect value) were chosen and defined to be a *MAC/MIC sequence pair* (this is between the MIC contig and matching MAC contig). Here, more than one MAC contig could be paired with the same MIC contig but at different positions. For example, MAC contig-1 could match with MIC contig-12 from positions 200 to 4000, and MAC contig-4 could match with the same MIC contig-12 at a position of 6000 to 12,000. But multiple MIC contigs were not allowed to pair with the same MAC contig. This is because almost all MAC chromosomes have one gene, but MIC chromosomes tend to be much longer and have many genes [10]. Then, through summarizing these alignments (Figure 3.1), a subword of a micronuclear contig was divided into



```
gi|403330626|gb|AMCR01022363.1| gi|659195757|gb|ARYC01002517.1| 100.00  1170    0    0   385 1554    26772   27941   0.0  2250
gi|403330626|gb|AMCR01022363.1| gi|659195757|gb|ARYC01002517.1| 99.17   959 8   0    2906     3864    28961   28003   0.0  1798
gi|403330626|gb|AMCR01022363.1| gi|659195757|gb|ARYC01002517.1| 100.00  386 0   0    2166     2551    30257   29872   0.0   742
gi|403330626|gb|AMCR01022363.1| gi|659195757|gb|ARYC01002517.1| 100.00  356 0   0    37   392 26348   26703   0.0   685
gi|403330626|gb|AMCR01022363.1| gi|659195757|gb|ARYC01002517.1| 100.00  314 0   0    4592     4905    18603   18916   3e-170    604
gi|403330626|gb|AMCR01022363.1| gi|659195757|gb|ARYC01002517.1| 100.00  213 0   0    1955     2167    30599   30387   8e-112    410
gi|403330626|gb|AMCR01022363.1| gi|659195757|gb|ARYC01002517.1| 100.00  197 0   0    2698     2894    29299   29103   1e-102    379
gi|403330626|gb|AMCR01022363.1| gi|659195757|gb|ARYC01002517.1| 100.00  193 0   0    4124     4316    17767   17959   3e-100    371
gi|403330626|gb|AMCR01022363.1| gi|659195757|gb|ARYC01002517.1| 100.00  192 0   0    3853     4044    17329   17520   1e-99     369
gi|403330626|gb|AMCR01022363.1| gi|659195757|gb|ARYC01002517.1| 100.00  175 0   0    4307     4481    18126   18300   8e-90     337
gi|403330626|gb|AMCR01022363.1| gi|659195757|gb|ARYC01002517.1| 100.00  155 0   0    2548     2702    29507   29353   3e-78     298
gi|403330626|gb|AMCR01022363.1| gi|659195757|gb|ARYC01002517.1| 98.74   159 2   0    1689     1847    31041   30883   4e-77     294
gi|403330626|gb|AMCR01022363.1| gi|659195757|gb|ARYC01002517.1| 94.59   148 8   0    1543     1690    31250   31103   3e-60     239
gi|403330626|gb|AMCR01022363.1| gi|659195757|gb|ARYC01002517.1| 99.19   123 1   0    4477     4599    18399   18521   5e-58     231
gi|403330626|gb|AMCR01022363.1| gi|659195757|gb|ARYC01002517.1| 97.56   123 3   0    1839     1961    30787   30665   2e-54     219
gi|403330626|gb|AMCR01022363.1| gi|659195757|gb|ARYC01002517.1| 100.00  94  0   0    4036     4129    17618   17711   6e-43     181
gi|403330626|gb|AMCR01022363.1| gi|659195757|gb|ARYC01002517.1| 100.00  21  0   0    4902     4922    73244   73224   1.1 41.1
gi|403330626|gb|AMCR01022363.1| gi|659195757|gb|ARYC01002517.1| 100.00  20  0   0    4903     4922    1225    1244    4.0 39.1
```

Figure 3.1: Blastn report an alignment of a MAC contig with a MIC contig.

MDSs and IESs. Here, the term 'subword of a micronuclear contig' is used instead of "micronuclear contig", because there were many cases where a macronuclear contig was matched with a section of a micronuclear contig instead of the whole micronuclear contig, and another macronuclear contig matched with a different section of the same micronuclear contig.

Of the 22,363 MAC contigs, only 9 of these sequences did not match with a MIC contig. But, the rest of the 22,354 MAC contigs contain 99.9944% of the nucleotides of the MAC contigs. The 22,363 contigs have a sum of 67,158,112 bps, and then after mapping, 58,732,010 bps were mapped to the MIC genome, which is 87.453% of the MAC genome. Moreover, the 22,354 macronuclear contigs that matched only did so with 4459 distinct micronuclear contigs at different positions. This means multiple MAC contigs frequently mapped to the same MIC contigs. An example of this scenario has been shown in Figure 3.2. This example in Figure 3.2 is additionally noteworthy since the MDSs intertwine between the two different MAC contigs, which is known to occur [10]; i.e. the first MDS of contig 5 occurs before the last MDS of contig 1. The large number of MIC contigs that did not match a MAC contig might be caused by the relatively large amount of spacer DNA between genes that is not present in the MAC. Moreover, we only considered the MIC contig that had the lowest E-value with the each MAC. Other MIC contigs that matched with lower scoring values were ignored as they had poorer sequence similarity. To verify these hypothesis, we removed the frequently mapped 4459 MIC contigs, and then ran blastn another time again with the 22,363 MAC contigs against the remaining MIC contigs. The results showed that, only 26.73% of the MAC genome was mapped with the remaining MIC contigs (the 22,363 contigs have a sum of 67,158,112 bps, and then after mapping, 17,954,650 bps were mapped with the remaining MIC contigs, which is 26.73% of the MAC genome). We did not pursue this issue further as there was sufficient data to study all the problems in this thesis.

Then, once the matching segments were determined, for each MAC contig, if n subwords matched a MIC contig, then a permutation of the numbers $1\ 2 \cdots n$ was determined giving the order of the matching segments on the matching MIC segment. For example, Figure 3.3 shows how a MAC contig matches with a MIC contig and that is used to determine the MDS order on the MIC contig.

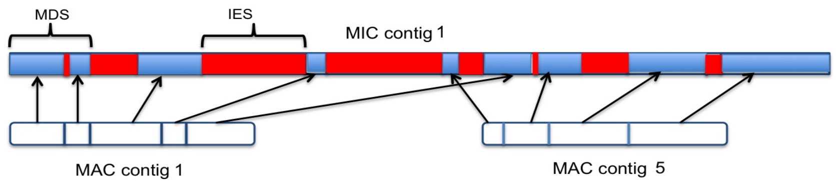


Figure 3.2: Alignment of a MIC contig with two distinct MAC contigs.

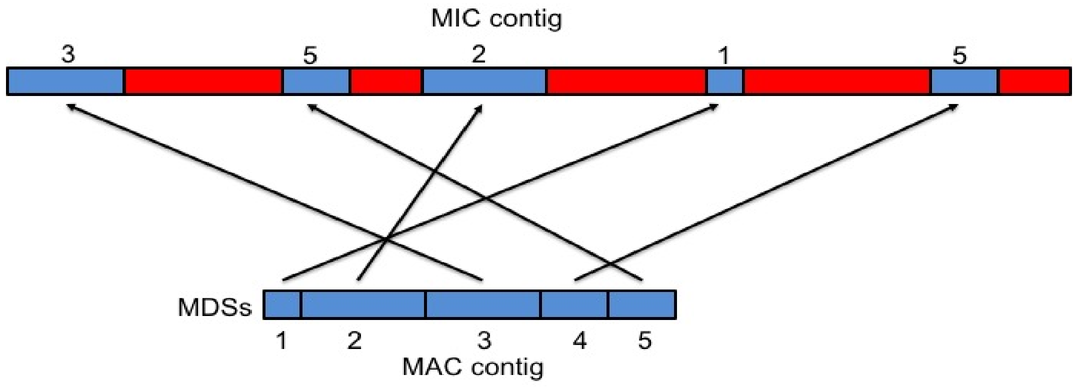


Figure 3.3: Mapping of the matching segments of a MAC contig with a MIC contig to determine the order of MDSs on the MIC contig.

3.1 Analysis of MDS Orders

For each of the 22,354 matching MAC contigs, an order of the MDSs is determined. The sequences of MDSs on the macronuclear contigs are always defined to be sequential, or in sorted order, i.e, $1\ 2\ 3\ 4 \cdots n$, and the sequences of MDSs of the corresponding subwords of the micronuclear contigs are one of the possible permutations of $1\ 2\ 3\ 4 \cdots n$. For the rest of this thesis, the *MIC MDS sequence* of a MAC contig will be the order of MDSs in the contig as determined by this procedure.

Among the 22,354 matching sequences, the MIC MDS sequence of 8803 MAC contigs were unscrambled of the form $1\ 2 \cdots n$ for some n. For these, no rearrangement is needed during assembly. Further, 9512 sequences were also unscrambled with the negated identity, and were in the following order: $-n\ -(n-1) \cdots -1$. Here, the '$-$' sign represents that the MDS is oriented in the opposite direction. The reason why the negated identity is unscrambled is one of the strings is written $5'$ to $3'$ along one of the two strands, and the other strings is written $5'$ to $3'$ along the other strand. Together, these 18,315 (8803 + 9512) are called the *unscrambled sequences*. Thus, there are a total of 4039 MAC contigs where the MIC MDS sequence is scrambled; i.e. not either the identity or the negated identity. These are called the *scrambled sequences.*

These 22,354 sequences have a total of 226,720 MDSs with an average of 10.142 per contig. Among them, the largest sequence has 240 MDSs, and there are 1138 sequences with 1 MDS only. It was observed that the unscrambled sequences have fewer MDSs than scrambled sequences. The 18,315 unscrambled sequences have a total of 157,152 MDSs, whereas, the 4039 scrambled sequences have a total 69,568 MDSs. Thus, the average number of MDSs for unscrambled sequences is 8.58, and for scrambled sequences it is 17.22. But on average, the length of MDSs of unscrambled sequences is larger than the length of MDSs of scrambled sequences. The average length of MDSs in unscrambled sequences is 298.41 nucleotides, whereas the average length of MDSs in scrambled sequences is 223.75.

An analysis on these 4039 MIC MDS sequences that are scrambled showed that there were 207 sequences which had all odd MDSs followed by all even MDSs (not necessarily sequentially), or vice versa. Of these, 106 sequences have all evens first and then all odds, and the remaining 101 sequences have odds followed by evens. We named the patterns as *even-odd* and *odd-even* patterns respectively. There are 15 sequences that have all consecutive even numbers first, and then all consecutive odd numbers (i.e. $2\ 4\ 6 \cdots 1\ 3\ 5 \cdots$), and 2 sequences that have vice versa (i.e $1\ 3\ 5 \cdots 2\ 4\ 6...$). Though, there were 2 sequences of these 17 sequences that have one inverted MDS in this pattern. They are $1\ -2$ and $2\ -1$. Thus, to study these patterns more, the 4039 scrambled MIC MDS sequences were further analysed.

Initially, every scrambled sequence was divided into two categories: one with the sequences with MDSs only in one direction (i.e. either all of the MDSs are in the non-inverted order, or in the inverted order), and the other with the sequences that have MDSs in both directions. Then, we got 2443 scrambled sequences containing only MDSs that exist in one orientation (which are called *unidirectional sequences*), and 1596 scrambled sequences containing two separate subsequences that exist in both orientations (called *bidirectional sequences*). Of the 1596 bidirectional sequences having two subsequences in both orientations, the individual subsequences may or may not be in the correct order. Thus, we further analysed these 1596 scrambled sequences. Of these, there are 952 sequences where both subsequences are unscrambled, 539 sequences where any one of the subsequences are unscrambled, and 105 sequences where neither are unscrambled.

Next, the 1596 MIC MDS scrambled sequences were divided into its two sub-subsequences: one holding the order of MDSs that are in the non-inverted order, and the other holding the order of MDSs that are inverted. Of the 1596 bidirectional sequences, separating out the two sequences in each orientation gives 3192 subsequences. Each subsequence was considered as a new independently unidirectional sequence holding the order of the MDSs, in a set called the *extracted unidirectional sequences.* After this, the total number of scrambled sequences became 5635 from 4039 — as there were 2443 unidirectional sequences, and 3192 extracted unidirectional sequences. Then, the 5635 scrambled sequences were processed by taking all maximal subwords of consecutive numbers, keeping only the smallest of those numbers, then removing unused numbers (this has the effect of treating subwords that are already in order as only one number). Henceforth in the thesis, only these sequences will be used. This was done to investigate more about the scrambled patterns of the data.

Therefore, after separating out all inverted parts as separate sequences, and removing consecutive numbers, there are 5635 scrambled input sequences. Of the 2443 unidirectional sequences after removing consecutive numbers from the unidirectional sequences, these are called *renumbered unidirectional sequences*, and of the 3192 extracted unidirectional sequences, after removing consecutive numbers, these are called *renumbered extracted unidirectional sequences*. Interestingly, the even-odd and odd-even patterns were common in these scrambled input sequences. Of the 2443 renumbered unidirectional sequences, 1044 sequences had all odd order MDSs first, then all even order MDSs, and 1045 sequences had all even ordered MDSs first, then all odd ordered MDSs. Moreover, among the 1044 odd-even sequences, there were 986 sequences that had all the consecutive odd numbers MDSs (i.e. $1\ 3\ 5\cdots$) and then, all the consecutive even numbers MDSs (i.e. $2\ 4\ 6\cdots$). Similarly, among 1045 even-odd sequences, there were 985 sequences that had all the consecutive even numbers MDSs (i.e. $2\ 4\ 6\cdots$), and then all the consecutive odd numbers MDSs (i.e. $1\ 3\ 5\cdots$). It is important to treat sequences that have both inverted and non-inverted MDSs separately, as the next chapter shows this will indeed effect the descrambling order. Next, from the 3192 renumbered extracted unidirectional sequences, 2443 sequences were unscrambled, 293 sequences had all odd order MDSs first, then all even order MDSs, and 321 sequences had all even ordered MDSs first, then all odd ordered MDSs. Moreover, among 293 odd-even sequences, there were 280 sequences that had all the consecutive odd numbers MDSs (i.e. $1\ 3\ 5\cdots$) and then, all the consecutive even numbers MDSs (i.e. $2\ 4\ 6\cdots$). Similarly, among 321 even-odd sequences, there were 302 sequences that had all the consecutive even numbers MDSs (i.e. $2\ 4\ 6\cdots$) and then, all the consecutive odd numbers MDSs (i.e. $1\ 3\ 5\cdots$). Figure 3.4 shows a data flow diagram to depict the categorizations of the MIC MDS sequences that have been discussed above.

In summary, initially there were 22,354 MIC MDS sequences, among them 18,315 MIC MDS sequences (81.93%) were unscrambled sequences, and 4039 MIC MDS sequences (18.06%) were scrambled sequences. These 4039 scrambled sequences were separated out so that all inverted parts were separate sequences, and after removing the consecutive numbers, there are 5635 scrambled input sequences. Of the 5635 scrambled input sequences, 2443 were unscrambled (43.35%) (as they are the unscrambled subsequences of the bidirectional sequences holding either the inverted, or non-inverted MDSs), 150 follow even/odd patterns, but not consecutive (2.66%), 2553 follow consecutive even/odd patterns (45.30%). Therefore, 639 sequences are neither unscrambled, nor are of the form consecutive odd/even numbers (11.34%) (489 are also not even/odd patterns (8.68%), and these are called "more complex scrambled" in Table 3.1). Table 3.1 shows the detail summary of the statistics above.

For the descrambling order analysis project, the input data are the renumbered unidirectional sequences and renumbered extracted unidirectional sequences of MIC MDS sequences of all the MAC contigs.

3.2 Preprocessing and Analysis for MDS and IES Identification

Next, with the specific coordinates of MDSs and IESs in subwords of 4459 micronuclear contigs, 22,354 sequences of micronuclear subwords had the nucleotides labelled as MDS, or IES. Among the sequences, 1197 sequences had no IESs with 1138 of them having only one MDS, and the remaining 59 sequences containing

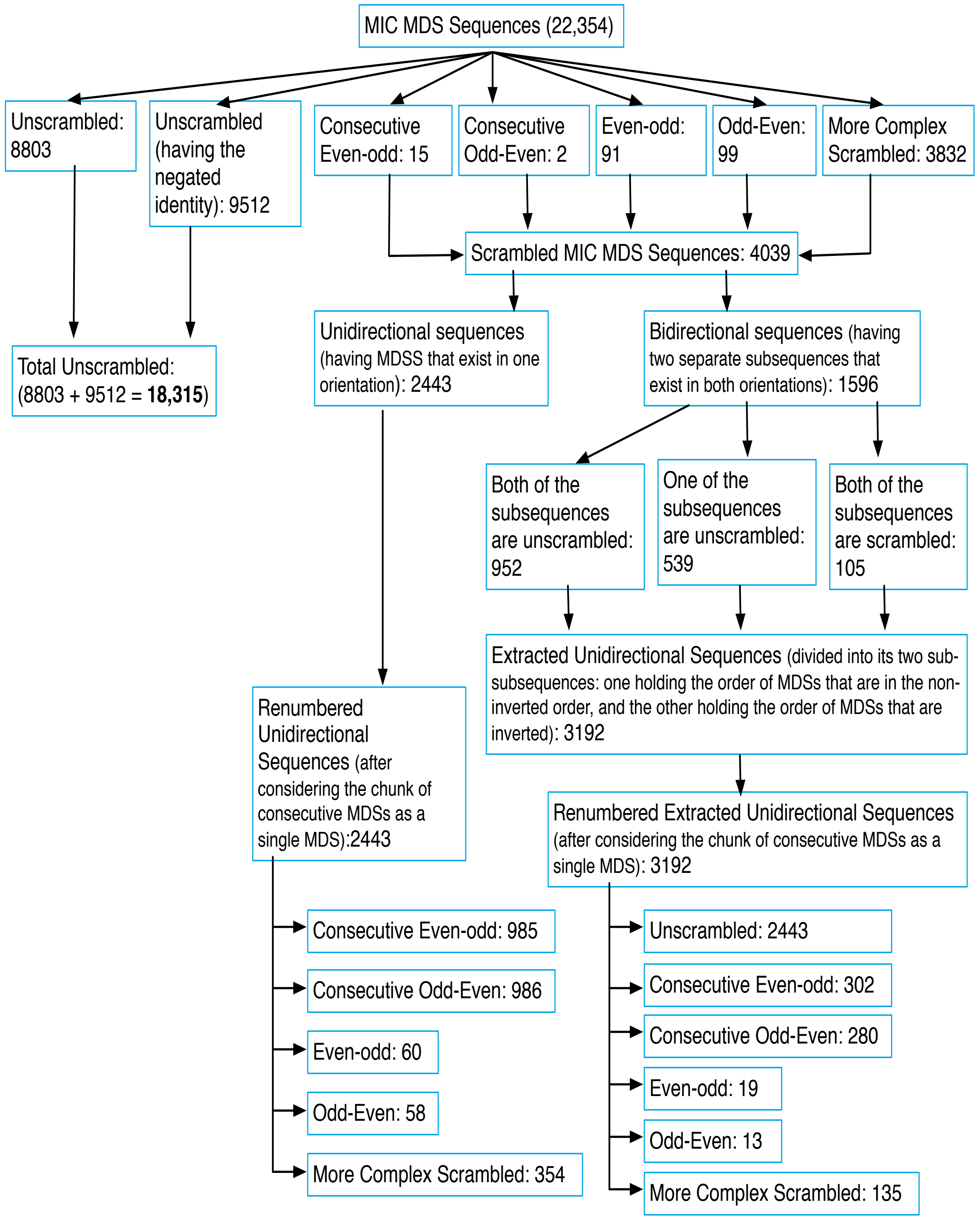

MIC MDS Sequences (22,354)
Unscrambled: 8803
Unscrambled (having the negated identity): 9512
Consecutive Even-odd: 15
Consecutive Odd-Even: 2
Even-odd: 91
Odd-Even: 99
More Complex Scrambled: 3832
Scrambled MIC MDS Sequences: 4039
Total Unscrambled: (8803 + 9512 = 18,315)
Unidirectional sequences (having MDSS that exist in one orientation): 2443
Bidirectional sequences (having two separate subsequences that exist in both orientations): 1596
Both of the subsequences are unscrambled: 952
One of the subsequences are unscrambled: 539
Both of the subsequences are scrambled: 105
Extracted Unidirectional Sequences (divided into its two sub-subsequences: one holding the order of MDSs that are in the non-inverted order, and the other holding the order of MDSs that are inverted): 3192
Renumbered Unidirectional Sequences (after considering the chunk of consecutive MDSs as a single MDS):2443
Renumbered Extracted Unidirectional Sequences (after considering the chunk of consecutive MDSs as a single MDS): 3192
Consecutive Even-odd: 985
Consecutive Odd-Even: 986
Even-odd: 60
Odd-Even: 58
More Complex Scrambled: 354
Unscrambled: 2443
Consecutive Even-odd: 302
Consecutive Odd-Even: 280
Even-odd: 19
Odd-Even: 13
More Complex Scrambled: 135

Figure 3.4: Categorizations of the MIC MDS sequences of the *Oxytricha trifallax* ciliates.

Table 3.1: MDS order patterns among the MIC MDS sequences of the *Oxytricha trifallax* ciliates.

MIC MDS Sequence patterns			Frequency of such sequences in data
Unscrambled sequences			18,315
Scrambled Input Sequences (5635)	Renumbered Unidirectional Sequences (2443)	Consecutive odd-even sequences	986
		Consecutive even-odd sequences	985
		Odd-Even pattern sequences	58
		Even-Odd pattern sequences	60
		More complex scrambled sequences	354
	Renumbered Extracted Unidirectional Sequences (3192)	Unscrambled sequences	2443
		Consecutive odd-even sequences	280
		Consecutive even-odd sequences	302
		Odd-Even pattern sequences	13
		Even-Odd pattern sequences	19
		More complex scrambled sequences	135

MDSs and pointers only. So, these sequences were excluded from the input sequences of the classifying MDSs and IESs by HMMs problem, as they contain 0 bp of IESs. Thus, the 21,157 labelled sequences of micronuclear contig subwords are the input sequences for the classifying MDSs and IESs by HMMs problem.

Also, these 22,354 micronuclear subwords have more basepairs as part of IESs than MDSs. The 22,354 micronuclear sequences have 27.25% MDS and 72.75% IESs, in terms of base pairs. Both MDSs and IESs are A -T rich — IESs have 37.25% A, 37.18% T, and similarly, MDSs have 34.56% A, and 34.53% T (Figure 3.5). The number of each distinct k-mers (i.e. subwords of length k) was also calculated for IESs and MDSs, for k in $2, 3, 4, 5$. Input data distributions for these k-mer nucleotides are shown in Figure 3.6. In Figure 3.6, the x-axis represents the k-mers, and y-axis represents the frequency of each k-mer, for k in $2, 3, 4, 5$. Thus, increasing values of k in each k-mer adds the previous nucleotides. For example, the knowledge of the previous

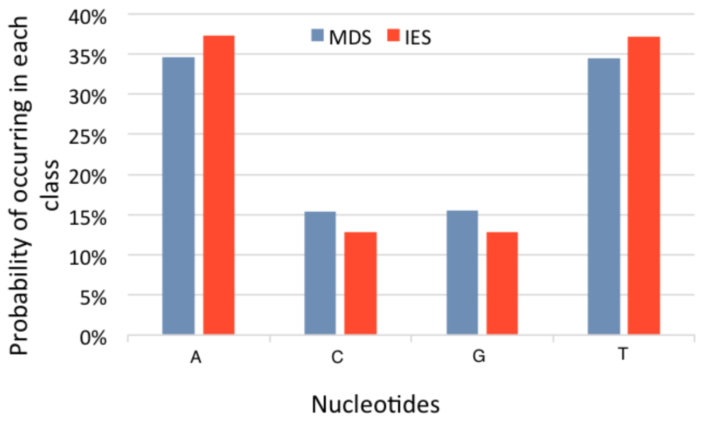
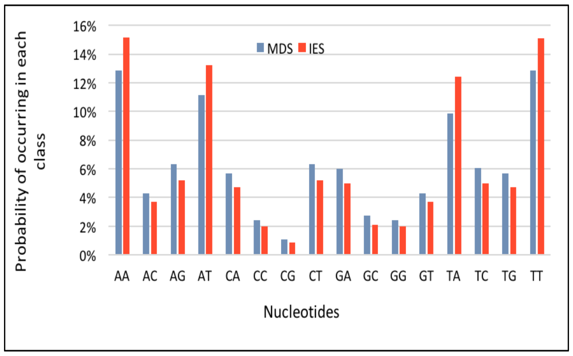


Figure 3.5: MDSs and IESs have more A-T than C-G in them.

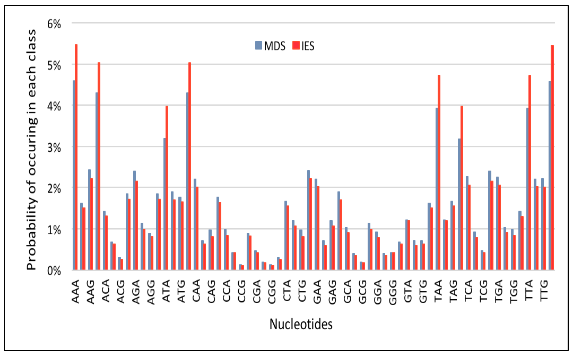
nucleotide showed that, CG is the least occurring dinucleotide for both IESs (0.86%) and MDSs (1.07%), but the highest occurring dinucleotide in IESs is AA (15.13%), and in MDSs is TT (12.87%) (Figure 3.6a). However, CCGT is the least occurring 4-mer with 0.0001% probability, and CGGC and CGGG are the (tied for) second least common 4-mers with 0.0002% probability in both MDSs and IESs. Moreover, the data distribution showed that there is a higher probability of getting an A or T after an A or T, and the probability of getting A or T after C or G is higher than getting C or G after C or G. Finally, the input data with 5-mers demonstrated the following: ACCGG, CCGGT, CGCCC, CGCCG, GTCCG, TCCGG, TCGCG, and ACGCG bases were never seen in both MDSs and IESs, ACCCG and ACGGG were never seen in the IESs, and CACGG and CCCCG were never seen in the MDSs, though, CCCCG occurred in IESs with a 0.0001% probability.

3.3 Limitations of Data

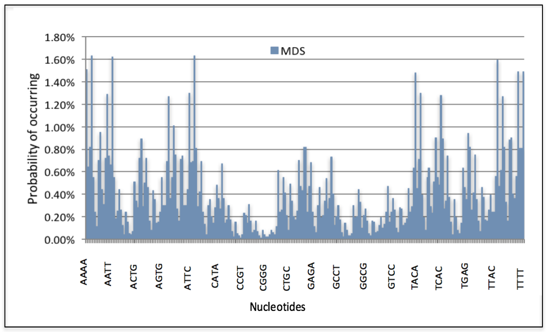
The data processing started with each MAC contig mapping to all 25,720 MIC contigs, finding the best hit, and then associated the best hit's MIC contig with the MAC contig. This procedure was continued for all of the 22,363 MAC contigs. There were some cases where segments from a single MAC contig were mapped with two separate MIC contigs (Figure 3.7 explains this scenario). In these cases, for the order analysis work, we made the decision to ignore the MDSs associated with the second MIC contig, as there is no clear "order" to MDSs in this case (as MDSs are ordered sequentially with MIC contigs). Of the, 22,354 MAC contigs, there were 411 cases where more than 99.9% of nucleotides mapped, and 17,313 cases where more than 90% of nucleotides mapped. Thus, among the remaining 5041 cases, there are 1284 sequences, where more than 50% of the nucleotides are unmapped, and among them, there are 213 sequences, where more than 80% nucleotides are unmapped. In total, 12.54% of the nucleotides of the 22,363 MAC contigs were unmapped. There were 30 examples where over 99% of the nucleotides were unmapped. The macronuclear contigs where



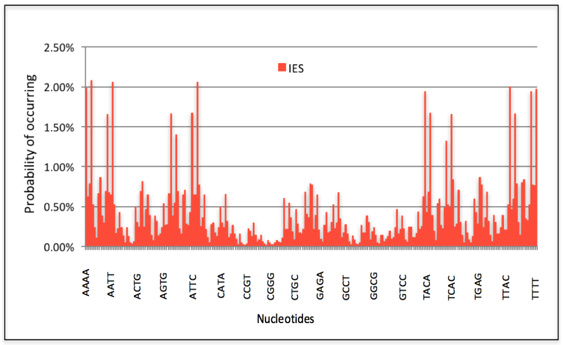
(a) 2-mer nucleotides



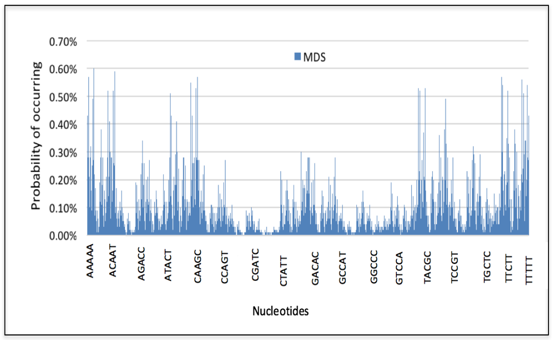
(b) 3-mer nucleotides



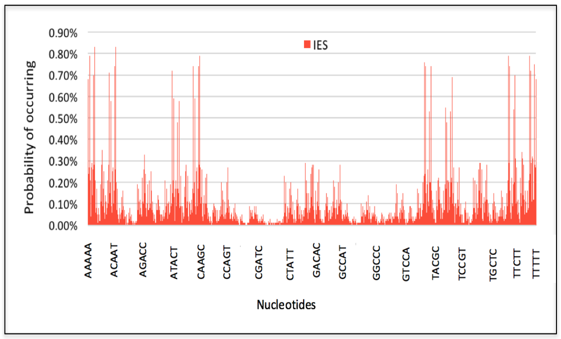
(c) 4-mer nucleotides - class MDS



(d) 4-mer nucleotides - class IES



(e) 5-mer nucleotides - class MDS



(f) 5-mer nucleotides - class IES

Figure 3.6: Distribution of k-mers in class MDS and IES, for $k = 2, 3, 4, 5$.

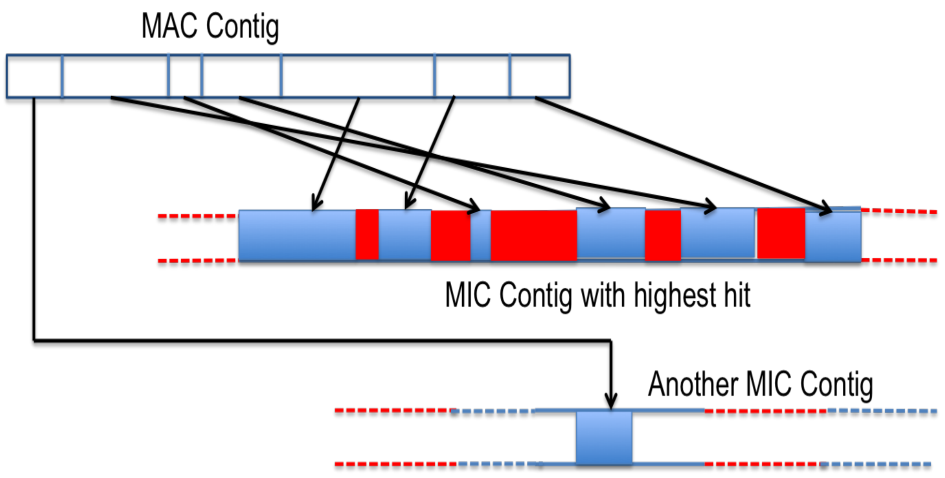


Figure 3.7: Mapping of one MAC contig with more than one MIC contigs.

more than 50% nucleotides were unmapped, gave rise to the fact that there might be a possibility of missing micronuclear genome data. However, this fact should not significantly affect the study in this thesis since there is a large amount of data from which conclusions can be drawn. Moreover, the 22,363 macronuclear contigs code for 24,578 proteins [24].

The reasons behind using contigs rather than genes for descrambling order analysis is two fold: firstly, working with genes instead of whole contigs was leading to some missing MDSs that reside within the macronuclear contigs but not in the genes (this information has been cross-checked with mds_ies_db database [6]). Secondly, in the macronucleus, the MDSs rearrange in a correct order so that they can undergo transcription. Thus, the descrambling of MDSs actually occurs on MAC chromosomes rather than genes (although most MAC chromosomes only contain one gene).

Additionally, for the problem of classifying MDSs and IESs by HMMs, the input for training were the subwords of micronuclear contigs that were divided into MDSs and IESs. The input data always started with a MDS, and ended in a MDS, because the segments between two MDSs are certainly a true IES. Moreover, the remaining unclassified sections of a micronuclear contig which is not a MDS might not end as an IES also, as it might be spacer, promotors, or other sections of a micronucler DNA sequence. Finally, it is possible to eliminate these unknown parts from the input data without much impact because the 22,354 micronuclear contigs subwords were sufficiently long in terms of base pairs with a maximum length 350,663 bp, minimum length 131 bp, and average length of 10,768 bp.

Chapter 4

Descrambling Order Analysis

As discussed in Chapter 1, the patterns of the order of MDSs in micronuclear genes (Figure 1.5) shows evidence of some biological operations that can be computationally described with shuffle operations. Furthermore, the MDS order analysis in Chapter 3 showed that, these "shuffle" patterns are seen notably in the order of MDSs of the micronuclear chromosome fragments of the *Oxytricha trifallax* ciliate genome, an idea that is explored further and quantified in various ways in this chapter. Thus, this study provides computational evidence of the fact that a structural component is partially enforcing this shuffle-like behaviour by properly aligning segments in an interleaving fashion.

Therefore, the following sections provide a detailed discussion of the descrambling order determination problem, implemented algorithms for predicting descrambling order, and an analysis of the results obtained.

4.1 Problem Overview

The order of MDSs of a MIC chromosome corresponding to a MAC chromosome is represented by a permutation $\pi = \pi_1\pi_2\pi_3 \cdots \pi_n$, where $|\pi_1|, \ldots, |\pi_n|$ is a permutation of $1, 2, \ldots, n$. The order of the MDSs in each MAC chromosome is represented by the identity permutation on n elements i.e. $1\ 2 \cdots n$. Informally, the descrambling order analysis problem is to determine the minimum number of "steps" required to transform an input permutation π into the identity permutation, where each step contains certain operations that change different parts of strings in parallel. This research attempts to predict the descrambling order at a higher level of abstraction suggested by the patterns occurring in ciliate MIC data, instead of lower level changes at the molecular level. Thus, the operations do not represent any single molecular level ciliate biological operation (for example, the number of inversions, or loop deletion operations are not considered here); instead, it is a generalized term for representing parallel operations. Every parallel application of this operation can be interpreted as a single *move*. Thus, given a permutation $\pi = \pi_1 \cdots \pi_n$, the descrambling order determination problem is to find the shortest number of moves needed to transform π into the identity permutation. Note that this approach is in contrast with the genome rearrangement problems discussed in Chapter 2, which does focus on individual operations.

This study focuses on determining the required minimum number of moves to descramble the section of a MIC chromosome corresponding to one MAC chromosome. This is because there are usually multiple genes per MIC but usually only one per MAC [10]. An attempt is made to determine the minimum number of moves needed for each such part of a MIC. The reason for focusing on the smallest number of moves is, it is assumed that the real number of operations required to descramble a micronuclear chromosome is fairly close to the smallest number of required moves. Moreover, because we do not know the exact mechanism by which descrambling takes place, we will study four similar systems of moves involving shuffle to see how the minimum number of operations differs between the types. If one system of moves gives significantly lower number of moves required, then there are advantages to this system in terms of parsimony.

As mentioned in the data analysis section, there are a number of sequences (after renumbering) in consecutive odd-even and consecutive even-odd patterns, which means all the consecutive even numbers (order of MDSs) occur first, followed by the consecutive odd numbers, or vice versa. When the odd and even numbers are in consecutive order, then it only requires a single application of shuffle to transform the permutation into the identity permutation. For example, the permutation of alpha-telomere-binding protein genes of *Sterkiella histriomuscorum* is 1 3 5 7 2 4 6 (Figure 1.5). This gene can be descrambled in one step by taking the shuffle of two subwords 2 4 6 with 1 3 5 7. In that case, there is a possibility that the DNA might fold into a U-shape, and then recombination operations take place in parallel (or without significantly changing the structure between individual recombination) to descramble the micronuclear chromosome, and therefore a structural component is partially enforcing an alignment of appropriate MDSs so that the operation is applied correctly. An abstracted example of such a scenario is shown in Figure 4.1. Note that the computational description of this operation is similar with the shuffle operation except that it applies the operation on segments of the same input string rather than on two separate strings. Recall also from Section 2.6 that shuffle gives a set of possibilities, rather than a unique one. In contrast, concatenation is deterministic on words (the concatenation of two sequences is a single sequence). This is not the case for shuffle, as the shuffle of two words contains potentially many words, and therefore shuffle is nondeterministic.

For example, the permutation 1 3 5 7 9 11 2 4 6 8 10 12 of Figure 4.1, is the concatenation of 1 3 5 7 9 11 and 2 4 6 8 10 12. And, a shuffle operation between the same two segments 1 3 5 7 9 11 and 2 4 6 8 10 12 can give the identity permutation 1 2 3 4 5 6 7 8 9 10 11 12. Therefore, the identity permutation $w = 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12$ can be obtained via one application of the shuffle operation from two subwords $u = 1\ 3\ 5\ 7\ 9\ 11$ and $v = 2\ 4\ 6\ 8\ 10\ 12$ of the input permutation (where the input permutation is equal to uv). However, for the descrambling order analysis study, a shuffle operation can be thought of as multiple recombination operations as w can be also derived by multiple recombination operations of the elements of v in u, or u in v — preserving the order of the elements of u and v. Determining the minimum number of applications of shuffle to descramble a permutation will involve determining the subwords u and v of the input permutation first, since u and v are the segments on which the shuffle is applied. Thus, determining the minimum number of segments is equivalent to determining the minimum number of applications of shuffle. Indeed, the number of operations will be always one less than the number of operands. This point will be explained further in the next section. Therefore, determining the minimum number of segments will be the

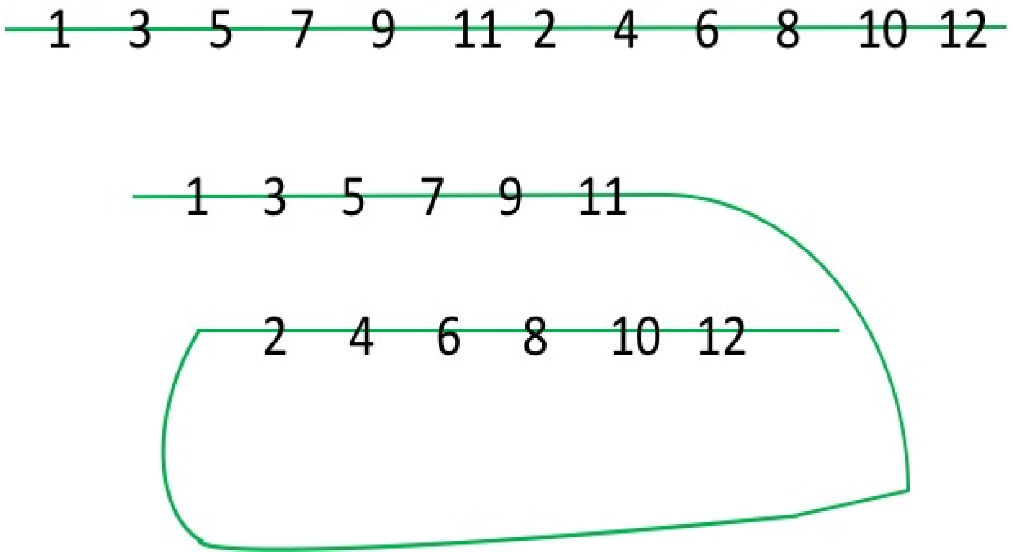


Figure 4.1: A sequence with odd ordered MDSs first, followed by even ordered.

focus.

Let π be a permutation. A move can be applied on different types of segments of the permutation π. The shuffle operation will be applied on segments of π, and each move will bring π one step closer to the identity permutation by merging parts of π. We will define four different systems, whereby the moves within each are slightly different.

4.2 Formal Definitions

The systems and algorithms require the following definitions.

Given a natural number n, then $\mathbb{Z}_+(n) = \{1, 2, \ldots, n\}$ and $\mathbb{Z}_-(n) = \{-n, \ldots, -1\}$, and $\mathbb{Z}_{+-}(n) = \mathbb{Z}_+(n) \cup \mathbb{Z}_-(n)$ (note 0 is not in this set). For $i \in \mathbb{Z}_{+-}(n)$, let $sgn(i)$ be $+1$ if $i > 0$ and -1 otherwise. It is also common to examine sequences of numbers represented in the form of strings (or words) with numbers for the alphabet. Given an alphabet A, then A^+ is the set of all non-empty words over A. Therefore, $\mathbb{Z}_{+-}(n)^+$ is the set of all non-empty strings over the alphabet $\mathbb{Z}_{+-}(n)$. A string $\pi = \pi_1 \cdots \pi_n$, $\pi_1, \ldots, \pi_n \in \mathbb{Z}_{+-}(n)$ is positive if $\pi \in \mathbb{Z}_+(n)^+$, and negative if $\pi \in \mathbb{Z}_-(n)^+$. Also, π is increasing if $\pi_1 < \pi_2 < \cdots < \pi_n$, and is decreasing if $\pi_1 > \pi_2 > \cdots > \pi_n$. A subword of π is any word $\pi_i\pi_{i+1}\cdots\pi_j$, $1 \leq i \leq j \leq n$. A subword $x = \pi_i \cdots \pi_j$ of π is called a maximal increasing subword if x is increasing, and $\pi_{i-1} \cdots \pi_j$ and $\pi_i \cdots \pi_{j+1}$ (if they exist) are not increasing. Similarly for other properties, such as maximal positive and increasing. The inversion of π, π^I, is the string obtained by reversing π and switching the sign of each number (this is analogous to the reverse complement). The reversal of π, π^R, is the string obtained by reversing π without switching the sign of each number. If $\pi \in \mathbb{Z}_+(n)^+ \cup \mathbb{Z}_-(n)^+$, then let $\overline{\pi}$ be equal to π if π is positive, and the inversion of π if π is negative (by reversing the numbers and making them all positive). Then $\overline{\pi}$ is always positive for every π. In addition, if $\pi \in \mathbb{Z}_+(n)^+ \cup \mathbb{Z}_-(n)^+$ where π is either increasing or decreasing, then let $\hat{\pi}$ be equal to π if π is positive and increasing, the reversal of π if π is positive and decreasing (by reversing the numbers without changing their signs), the inversion of π if π is negative and increasing, and the reversal of π^I if π is negative and decreasing (by reversing the numbers of π^I without changing their signs). The result is always a positive increasing sequence.

An input permutation is a string $\pi = \pi_1\pi_2 \cdots \pi_n$, $\pi_1, \ldots, \pi_n \in \mathbb{Z}_{+-}(n)$, where $|\pi_1|, \ldots, |\pi_n|$ is a permutation of $1, 2, \ldots, n$. And, π is a partial permutation, if there is no repeating number in $|\pi_1|, \ldots, |\pi_n|$ (but not all numbers need to be used). Intuitively, π is used to represent the sequence of MDSs in a MIC sequence, where a number is positive if it is in the forward orientation, and negative if it is in the inverted orientation. The goal is to transform it into the identity permutation $1, 2, \ldots, n$. It will be seen that, this is equivalent to splitting π into different segments. This will be done in four different ways. Each will be described first, followed by a more thorough investigation of each.

- **Contiguous Increasing System (CIS):** Given an input permutation $\pi = \pi_1 \cdots \pi_n$, a CIS partition of π is a sequence of words $u_1, \ldots, u_m$, with each $u_i \in \mathbb{Z}_+(n)^+ \cup \mathbb{Z}_-(n)^+$, and u_i is increasing for each i, $1 \leq i \leq m$, such that $\pi = u_1u_2 \cdots u_m$.

  Notice that whenever there is a CIS partition of words $u_1, \ldots, u_m$ where each word is positive, then the identity permutation is in $u_1 \sqcup\!\sqcup u_2 \sqcup\!\sqcup \cdots \sqcup\!\sqcup u_m$. More generally, whenever there is a CIS partition of words $u_1, \ldots, u_m$, then the identity permutation is in $\bar{u_1} \sqcup\!\sqcup \cdots \sqcup\!\sqcup \bar{u_m}$. This system allows the shuffle on increasing subwords (each either positive or negative, Figure 4.2a). In this case, the smallest (or close to smallest) number of increasing subwords of the input permutation such that the input permutation is the concatenation of the subwords is desired. In such a case, the identity is in the shuffle of the subwords after taking the inversion of any negative subwords. This point will be discussed further after after the systems are defined. For example, the permutation $5\ 6\ 2\ -8\ -7\ -4\ -3\ -1$ can be split into three increasing subwords, $u = 5\ 6$, $v = 2$, $w = -8\ -7\ -4\ -3\ -1$, the input is indeed uvw, and the identity is in $\bar{u} \sqcup\!\sqcup \bar{v} \sqcup\!\sqcup \bar{w}$.

- **Contiguous Increasing and Decreasing System (CIDS):** Given an input permutation $\pi = \pi_1 \cdots \pi_n$, a CIDS partition of π is a sequence of words $u_1, \ldots, u_m$, with each $u_i \in \mathbb{Z}_+(n)^+ \cup \mathbb{Z}_-(n)^+$, and u_i is increasing or decreasing for each i, $1 \leq i \leq m$, such that $\pi = u_1u_2 \cdots u_m$.

  In such a case where there is a CIDS partition, notice that the identity is in $\hat{u_1} \sqcup\!\sqcup \hat{u_2} \sqcup\!\sqcup \cdots \sqcup\!\sqcup \hat{u_m}$. Here, both increasing and decreasing subwords are allowed. A move can be applied on increasing and/or decreasing subwords (Figure 4.2b). Thus, the permutation $5\ 6\ 2\ 1\ -8\ -7\ -4\ -3$ can be split into three subwords $u = 5\ 6$, $v = 2\ 1$, $w = -8\ -7\ -4\ -3$, where the decreasing subword v will be turned into an increasing subword $\hat{v} = 1\ 2$, (where the hat symbol is reversing the word). Then the identity permutation is in shuffle of $\hat{u} = 5\ 6$, $\hat{v} = 1\ 2$, and $\hat{w} = 3\ 4\ 7\ 8$. Note that, the input permutation can be obtained by concatenating u, v, and w (uvw).

- **Non-Contiguous Increasing System (NIS):** Similarly, for a given input permutation $\pi = \pi_1 \cdots \pi_n$, a NIS partition of π is a sequence of subsequences $s_1, \ldots, s_m$, with each $s_i \in \mathbb{Z}_+(n)^+ \cup \mathbb{Z}_-(n)^+$, and s_i is increasing for each i, $1 \leq i \leq m$, such that $\pi \in s_1 \sqcup\!\sqcup s_2 \sqcup\!\sqcup \cdots \sqcup\!\sqcup s_m$.

  In such a case, the input permutation is in the shuffle of the segments. Further, the identity permutation is in $\bar{s_1} \sqcup\!\sqcup \cdots \sqcup\!\sqcup \bar{s_m}$. This system allows the shuffle on increasing subsequences. In this case, both the input permutation, and the identity permutation can be compared as a string of numbers derived from shuffle operation (Figure 4.2c). For example, the permutation 5 6 2 1 3 4 7 8 can be split into three increasing subsequences, $u = 5\ 6\ 7\ 8$, $v = 2$, and $w = 1\ 3\ 4$. Thus, the input permutation is in $u \sqcup\!\sqcup v \sqcup\!\sqcup w$, and the identity is in $\bar{u} \sqcup\!\sqcup \bar{v} \sqcup\!\sqcup \bar{w}$.

- **Non-Contiguous Increasing and Decreasing System (NIDS):** Lastly, a NIDS partition of input permutation $\pi = \pi_1 \cdots \pi_n$, is a sequence of subsequences $s_1, \ldots, s_m$, with each $s_i \in \mathbb{Z}_+(n)^+ \cup \mathbb{Z}_-(n)^+$, and s_i is either increasing or decreasing for each i, $1 \leq i \leq m$, such that $\pi \in s_1 \sqcup\!\sqcup s_2 \sqcup\!\sqcup \cdots \sqcup\!\sqcup s_m$.

  In this case, π is in $s_1 \sqcup\!\sqcup \cdots \sqcup\!\sqcup s_m$, and also the identity is in $\hat{s_1} \sqcup\!\sqcup \cdots \sqcup\!\sqcup \hat{s_m}$. The differences between this system, and the continuous increasing and decreasing system are, the segments of moves for this system are not subwords but instead subsequences (Figure 4.2d). Similarly, the decreasing subsequences will be transformed into increasing subsequences by allowing necessary reversals and/or inversion operations. In addition, the input permutation and the identity permutation both can be derived by the shuffle operations between the determined subsequences. For example, the permutation 5 6 -3 2 1 -4 7 8 can be split into three subsequences, $u = 5\ 6\ 7\ 8$, $v = -3\ -4$, and $w = 2\ 1$. The decreasing subsequence w can be turned into an increasing subsequence $\hat{w} = 1\ 2$, by allowing a reversal operation. Similarly, the decreasing subsequence $v = -3\ -4$ can be turned into an increasing subsequence $\hat{v} = 3\ 4$ by allowing a reversal operation on $v^I = 4\ 3$. Thus, the identity is in $\hat{u} \sqcup\!\sqcup \hat{v} \sqcup\!\sqcup \hat{w}$, and the input permutation is in $u \sqcup\!\sqcup v \sqcup\!\sqcup w$.

Note that for all the systems, we are only interested in calculating the number of segments, which corresponds to the number of shuffle applications plus one. The number of reversals and inversions (for CIDS and NIDS) are not calculated, and has no impact on results.

Henceforth, different types of systems might take a different minimum number of moves for descrambling MIC chromosomes (ie. they can have a different minimum number of segments). Moreover, the examples show how determining the segments leads to the possible descrambling scenarios. Note that the biological interpretation of a move operation is one or more parallel recombination operations. Biologically, a CIS partition of π can be thought of as a parallel recombination of different subwords, where the inversion of a subword can occur before a parallel recombination. However, the other systems are intended as a theoretical investigation as to whether the number of operations can be reduced by adding more complexity to the operations.

Given any two positive increasing words u, v which use disjoint numbers, then there is a positive increasing

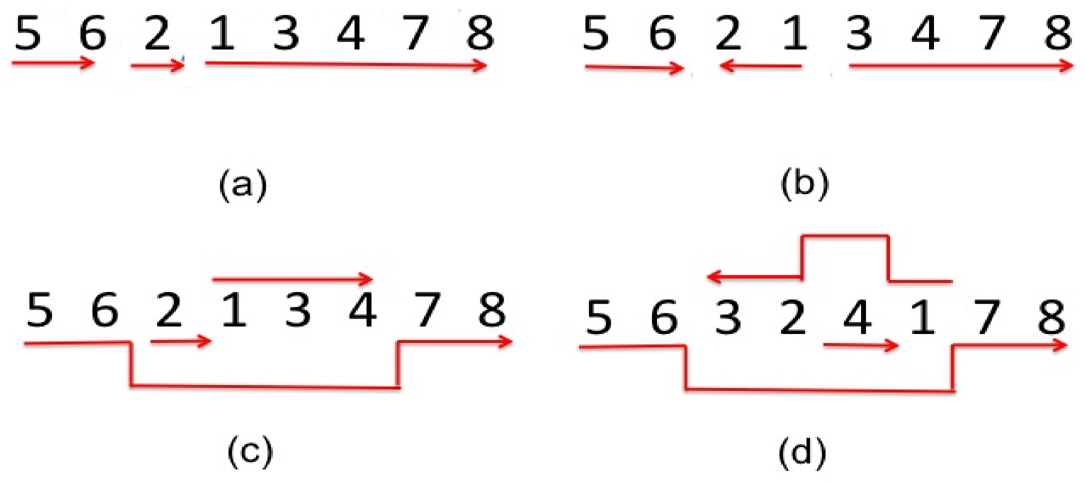


Figure 4.2: Red coloured arrows show the different types of segments on which the four systems allow moves (a) Increasing subwords (5 6), (2), and (1 3 4 7 8). (b) Increasing and decreasing subwords: increasing subwords (5 6), (3 4 7 8), and decreasing subwords (2 1). (c) Increasing subsequences (1 3 4), (2), and (5 6 7 8). (d) Increasing and decreasing subsequences (3 2 1), (4), and (5 6 7 8).

word in $u \sqcup\!\sqcup v$. Similarly, if both are negative and increasing than there is a positive increasing word in $\overline{u} \sqcup\!\sqcup \overline{v}$. If u is positive and v is negative, then there is a positive increasing sequence in $\overline{u}$ shuffled with $\overline{v}$. And in general, given $\pi = \pi_1 \cdots \pi_n$ and a CIS partition $u_1, \ldots, u_m$, then the identity must be in $\overline{u_1} \sqcup\!\sqcup \cdots \sqcup\!\sqcup \overline{u_m}$. That is, if there is a partition of size m, then the identity can be obtained with $m-1$ shuffle operations.

Conversely, if $\pi = u_1 \cdots u_m$, and the identity is in $\overline{u_1} \sqcup\!\sqcup \cdots \sqcup\!\sqcup \overline{u_m}$, then each u_i, $1 \leq i \leq m$ is either positive increasing or negative increasing. Hence, counting the number of segments in a CIS partition is always exactly one more than counting the number of applications of shuffle. This is similar for the other three systems as well. Hence, we will just count the segments for all of these systems instead of counting operations.

Thus, this study aims to determine the required minimum number of segments for each of these systems. For each, different algorithms will be tested that either determine the minimal number of segments, or that approximate it using some heuristic algorithm. From there, the average number of segments will be calculated for each algorithm on the real MIC/MAC pairs from Chapter 3. And, a low number of moves will lend theoretical evidence that this type of parallel recombination is taking place, where some structural component is enforcing the shuffle-like behaviour. The following sections describe the implemented algorithms and their results for each type of system.

4.3 Contiguous Increasing System

Consider an input permutation $\pi = \pi_1\pi_2\cdots\pi_n$. In this section, an algorithm to determine the optimal CIS partitions will be given. In π, a pair of adjacent elements π_i and π_{i+1}, $1 \leq i \leq n-1$, is called a *neighbour* if $\pi_i < \pi_{i+1}$ and either π, π_{i+1} are both positive or both negative; otherwise the pair is called a *cut-off point.* Note, π_i and π_{i+1} are two adjacent elements in π for all $1 \leq i \leq n-1$, and π_i and π_{i+1} are either neighbours or cut-off points.

If input permutation π has an increasing positive or negative subword $\pi_i\cdots\pi_j$, ie. $\pi_i < \cdots < \pi_j$ all the same sign, then each adjacent pairs in $\pi_i, \pi_{i+1}, \ldots, \pi_j$ are neighbours. Thus, in an increasing positive or negative permutation $\pi = \pi_1 < \pi_2 < \pi_3 < \cdots < \pi_n$, then π has zero cut-off point. In the similar way, a positive or negative permutation that has its elements in descending order will have n increasing subwords, where n is the length of the permutation, because $\pi = \pi_1 > \pi_2 > \pi_3 > \cdots > \pi_n$, and there are $n-1$ cut-off points. Thus, the number of increasing subwords depends on the cut-off points. As an increasing permutation will have no cut-off points, and an n length descending order permutation will have $n-1$ cut-off points, in both cases, the number of increasing subwords will be the number of cut-off points in π plus one.

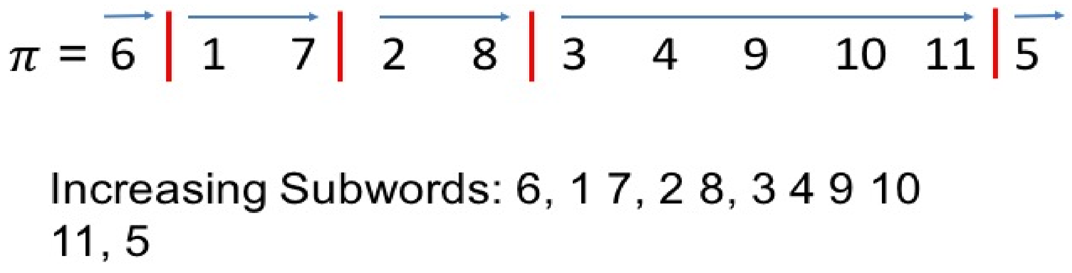
In general, let $\pi = \pi_1\pi_2\pi_3\cdots\pi_n$ have $c(\pi)$ cut-off points after position $c_1, \ldots, c_m$ of π. Then, it is impossible to have an increasing segment that includes a number from both before and after a cut-off point. Therefore, any CIS partition has at most $m+1$ elements in it. Furthermore, there exists a CIS partition with $m+1$ elements in it, because there is one increasing positive or negative subword that has the elements in between π_1 and the element at position c_1, and an increasing positive or negative subword between positions $c_i + 1$ and c_{i+1} for each i, $1 \leq i < m$, and one last increasing positive or negative subword that has the elements starting from $c_m + 1$ to π_n. As $m = c(\pi)$, the smallest number of increasing subwords in π will always be $c(\pi) - 1 + 2 = c(\pi) + 1$. Thus, the minimum number of CIS segments for an input permutation π is always equal to the number of cut-off points plus one.

Therefore, Algorithm 1 determines the optimal minimum size of a CIS partition of an input permutation π. To do this, it is sufficient to count the number of cut-off points $c(\pi)$ for each input permutation π. Then, it determines the minimum number of segments (increasing subwords) $s(\pi)$ in π, where it must be that $s(\pi) = c(\pi) + 1$. Figure 4.3 shows an example of increasing positive or negative subwords, and cut-off points in a permutation π.

For example, the sequence 2 4 6 1 3 5 7 has a consecutive even-odd pattern, one cut-off point between 6 and 1, and two increasing subwords: 2 4 6 and 1 3 5 7. Similarly, a consecutive odd-even pattern 1 3 5 7 9 2 4 6 8 10 has one cut-off point between 9 and 2, and two increasing subwords: 1 3 5 7 9, and 2 4 6 8 10. Thus, the sequences having consecutive odd-even patterns, or consecutive even-odd patterns will always have two increasing subwords — one with the consecutive odd numbers, and other with the consecutive even numbers. This is because the sequences having these patterns will always have a single cut-off point in between the ending of consecutive odd/even numbers, and the starting of consecutive even/odd numbers, respectively.

Algorithm 1 Minimum number of segments in CIS.

```
1: procedure IncreasingSubwords($\pi = \pi_1 \cdots \pi_n$)
2:     $segments \leftarrow 0$;
3:     $cut\_off\_points \leftarrow 0$;
4:     for $i \leftarrow 1$ to $n-1$ do
5:         if $\pi_i > \pi_{i+1}$ or $sgn(\pi_i) \neq sgn(\pi_{i+1})$ then
6:             $cut\_off\_points \leftarrow cut\_off\_points + 1$;
7:         end if
8:     end for
9:     $segments \leftarrow cut\_off\_points + 1$;
10:    return $segments$;
11: end procedure
```



π = 6 | 1 7 | 2 8 | 3 4 9 10 11 | 5
Increasing Subwords: 6, 1 7, 2 8, 3 4 9 10 11, 5

Figure 4.3: Example of a MIC chromosome fragment that has 5 increasing subwords to descramble.

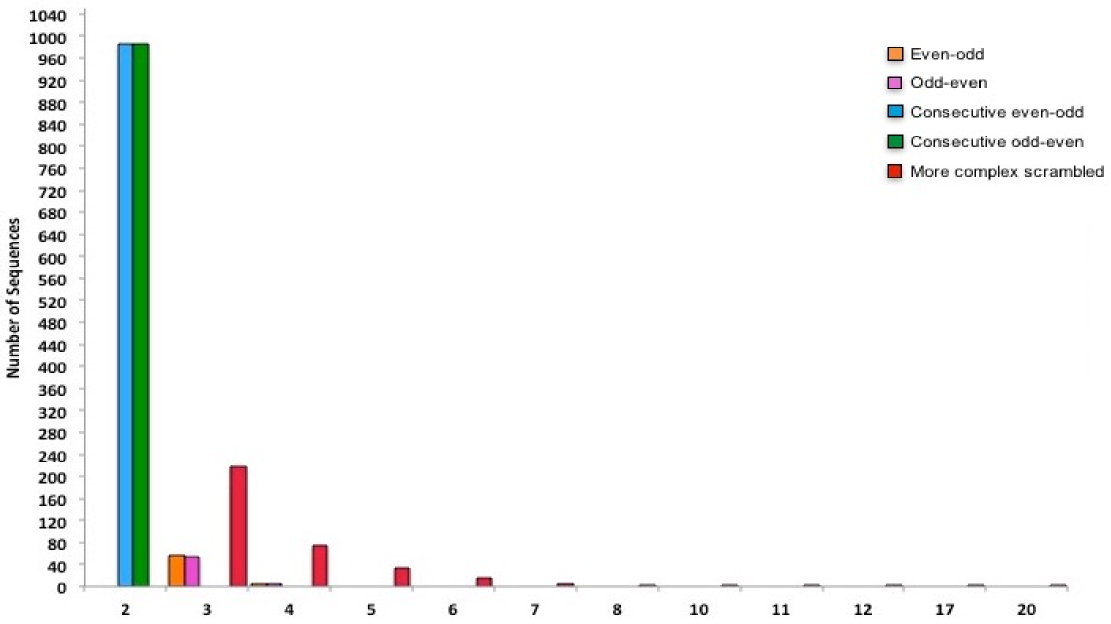


Figure 4.4: Relationship between the number of segments (increasing subwords) and the number of sequences in the dataset achieving that for renumbered unidirectional sequences.

Note that these sequences can be descrambled computationally by only one application of the shuffle operation between the consecutive odd numbers, and the consecutive even numbers. Also notice that for all sequences studied in Chapter 3 which have consecutive odd/even patterns after the renumbering procedure, these also give a single cut-off point and two segments. The graphs in Figures 4.4 and 4.5 show the visualisations of the number of segments versus the number of sequences achieving that minimal number of segments for renumbered unidirectional and renumbered extracted unidirectional sequences, respectively. The graphs show that the minimum number of increasing subwords for both of the datasets is 2, and the sequences with consecutive odd/even patterns are the only sequences with 2 increasing subwords. Indeed, the sequences that have only 1 increasing subword are already unscrambled, and only the sequences having the consecutive odds and evens will have 2 increasing subwords. In the graphs, the rest of the sequences have at least 3 increasing subwords. And, this is true for all of the sequences not having consecutive odd/even patterns, because the unscrambled segments of MDSs have already been considered as a single MDS while processing the scrambled input sequences. Thus, under this system the sequences with other patterns will need at minimum 2 shuffle operations to descramble, because they have at least 3 increasing subwords. For example, the following sequence 1 4 6 8 10 12 2 5 7 11 13 3 9, goes under the more complex scrambled patterns, and has 3 increasing subwords. Thus, the identity is in 1 4 6 8 10 12 $\sqcup\!\sqcup$ 2 5 7 11 13 $\sqcup\!\sqcup$ 3 9.

Algorithm 1 runs in linear time in the size of the input permutation, and is therefore very simple to calculate optimally. Table 4.1 and 4.2 show the average number of increasing subwords determined by Algorithm 1, along with the maximum number of increasing subwords, the number of sequences in each sequence pattern, the average length of increasing subwords, and the maximum length of increasing subwords

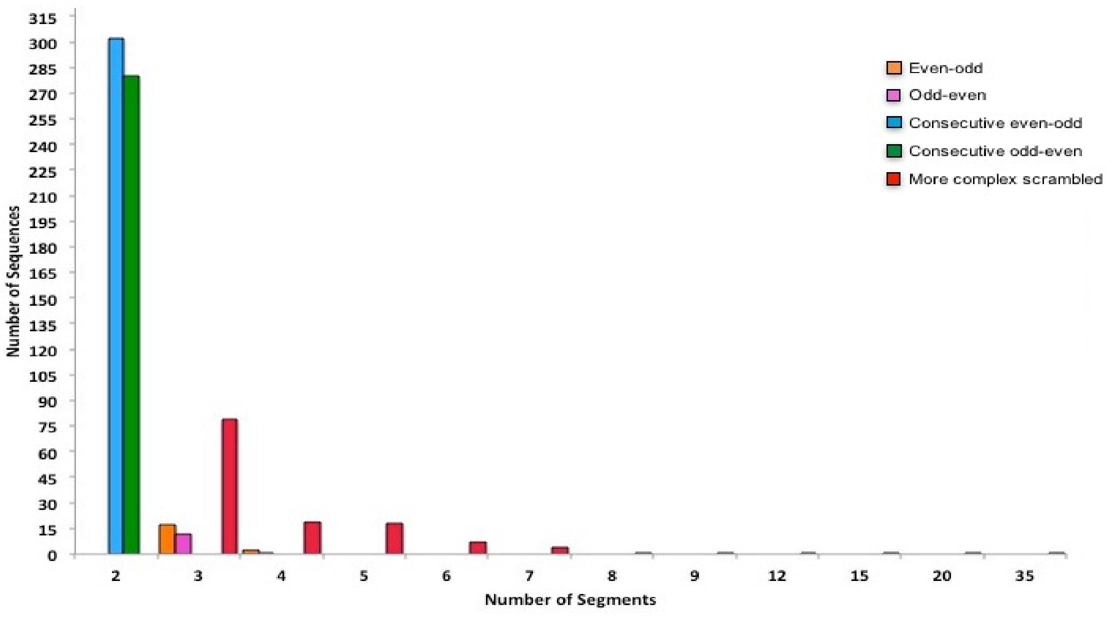


Figure 4.5: Relationship between the number of segments (increasing subwords) and the number of sequences in the dataset achieving that for renumbered extracted unidirectional sequences.

for renumbered unidirectional and renumbered extracted unidirectional scrambled sequences as described in Section 3.1. Note that for the bidirectional sequences, it is calculating the number of segments for the non-inverted parts and the inverted parts separately.

4.4 Contiguous Increasing and Decreasing System

For an input permutation $\pi = \pi_1\pi_2 \cdots \pi_n$, an element π_i, $1 \leq i \leq n$, is called a *stop point* if either of the following two conditions hold:

1. $sgn(\pi_{i-1}) \neq sgn(\pi_i)$,

2. $(sgn(\pi_{i-2}) = sgn(\pi_{i-1}) = sgn(\pi_i))$ and either $(\pi_{i-2} > \pi_{i-1}$ and $\pi_{i-1} < \pi_i)$ or $(\pi_{i-2} < \pi_{i-1}$ and $\pi_{i-1} > \pi_i)$,

and we say π_i is *ordered* otherwise. This means, if an element π_i, $1 \leq i \leq n$, is ordered, then either $i = 1$, or $i = 2$ and π_{i-1} and π_i are the same sign, or $i > 2$ and π_{i-2}, π_{i-1}, π_i are the same sign and either $\pi_{i-2} < \pi_{i-1} < \pi_i$ or $\pi_{i-2} > \pi_{i-1} > \pi_i$.

If input permutation π has an increasing subword $\pi_i \cdots \pi_j$, ie. $\pi_i < \cdots < \pi_j$, then each element in $\pi_{i+1}, \ldots, \pi_j$ are ordered. Similarly, if input permutation π has a decreasing subword $\pi_i \cdots \pi_j$, ie. $\pi_i > \cdots > \pi_j$, then each element $\pi_{i+1}, \ldots, \pi_j$ are ordered. Thus, in an increasing permutation $\pi = \pi_1 < \pi_2 < \pi_3 < \cdots < \pi_n$, then π has zero stop points. In the same way, a permutation that has its elements in descending order will have zero stop points. Thus, the number of increasing or decreasing subwords depends on the stop points,

Table 4.1: Increasing subword statistics with Contiguous Increasing System for renumbered unidirectional sequences.

Sequence patterns	**No. of Sequences**	**Avg. number of increasing subwords**	**Max. number of increasing subwords**	**Avg. length of increasing subwords**	**Max. length of increasing subwords**
Consecutive odd-even patterns	986	2	2	2.300	36
Consecutive even-odd patterns	985	2	2	2.305	37
Odd-even patterns	58	3.086	4	2.385	10
Even-odd patterns	60	3.067	4	2.728	26
More complex scrambled patterns	354	3.799	20	2.791	43

where each subword can be either increasing or decreasing. As both the increasing permutation and a descending permutation will have no stop points, both of them will have the number of stop points in π plus one, increasing or decreasing maximal subwords. Note that we are looking for "maximal increasing or decreasing subwords", for example 2 3 4 1 has one maximal increasing subword 2 3 4, one decreasing subword 4 1, and there is an overlap at stop point 4. If we do not consider "maximal", then 3 4 could count as an increasing subword.

In general, let $\pi = \pi_1\pi_2\pi_3\cdots\pi_n$ have $st(\pi)$ stop points, at position $st_1, \ldots, st_m$ of π, $m \geq 0$ (where these are ordered). Then, it is impossible to have a segment that includes a number from both before and including a stop point. Therefore, any CIDS partition has at most $m+1$ elements in it. Furthermore, there exists a CIDS partition with $m+1$ elements in it, because there is one subword that has the elements in between π_1 and the element at position $st_1 - 1$, and a subword between positions st_i and $st_{i+1} - 1$ for each i, $1 \leq i < m$, and one last subword that has the elements starting from st_m to π_n. As $m = st(\pi)$, the smallest number of subwords in π will be always $st(\pi) - 1 + 2 = st(\pi) + 1$. Thus, the minimum number of CIDS segments for an input permutation π is always equal to the number of stop points plus one.

Note though that for CIDS, there can be an element that is part of both a maximal increasing and a decreasing subword. For example, 2 3 4 1 has a maximal increasing subword of 2 3 4 and 4 1. But these overlaps are of length at most one and do not affect the number of increasing or decreasing subwords. But

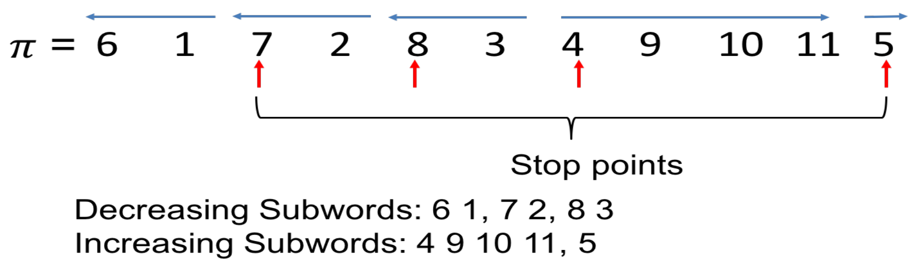
Table 4.2: Increasing subword statistics with Contiguous Increasing System for renumbered extracted unidirectional sequences.

Sequence patterns	**No. of Sequences**	**Avg. number of increasing subwords**	**Max. number of increasing subwords**	**Avg. length of increasing subwords**	**Max. length of increasing subwords**
Consecutive odd-even patterns	280	2	2	2.323	20
Consecutive even-odd patterns	302	2	2	2.488	25
Odd-even patterns	13	3.077	4	3.05	10
Even-odd patterns	19	3.105	4	4.423	21
More complex scrambled patterns	135	4.37	35	3.383	86

one can interpret the identity as being in either $2\ \hat{3}\ 4 \sqcup \hat{1}$ or in $2\hat{\ }3 \sqcup 4\hat{\ }1$.

Algorithm 2 determines the optimal minimum size of a CIDS partition of an input permutation π. To do this, it is sufficient to count the number of stop points $st(\pi)$ for each input permutation π. Then, it determines the minimum number of segments (increasing and/or decreasing subwords) $s(\pi)$ in π, where it must be that $s(\pi) = st(\pi) + 1$. Figure 4.6 shows an example of increasing and decreasing subwords, and stop points in a permutation π.

For example, the sequence 2 4 6 1 3 5 7 has a consecutive even-odd pattern, one stop point at 1, and two increasing subwords: 2 4 6, and 1 3 5 7. Similarly, an example sequence from consecutive odd-even pattern



π = 6 1 7 2 8 3 4 9 10 11 5
Stop points
Decreasing Subwords: 6 1, 7 2, 8 3
Increasing Subwords: 4 9 10 11, 5

Figure 4.6: Example of a MIC chromosome fragment that has 3 decreasing subwords, and 2 increasing subwords to descramble.

Algorithm 2 Minimum number of segments in CIDS.

```
procedure IncreasingAndDecreasingSubwords($\pi = \pi_1 \cdots \pi_n$)
    $segments \leftarrow 0$;
    $stop\_points \leftarrow 0$;
    if $n == 1$ then
        $stop\_points \leftarrow 0$;
    else if $n == 2$ then
        if $sgn(\pi_i) \neq sgn(\pi_{i-1})$ then
            $stop\_points \leftarrow 1$;
        end if
    else
        for $i \leftarrow 3$ to $n$ do
            if $sgn(\pi_i) \neq sgn(\pi_{i-1})$ then
                $stop\_points \leftarrow stop\_points + 1$;
            end if
            if $((sgn(\pi_{i-2}) == sgn(\pi_{i-1}) == sgn(\pi_i))$ && $(((\pi_{i-2} < \pi_{i-1})$ && $(\pi_{i-1} > \pi_i)) \| ((\pi_{i-2} > \pi_{i-1})$ && $(\pi_{i-1} < \pi_i)))$ then
                $stop\_points \leftarrow stop\_points + 1$;
            end if
        end for
    end if
    $segments \leftarrow stop\_points + 1$;
    return $segments$;
end procedure
```

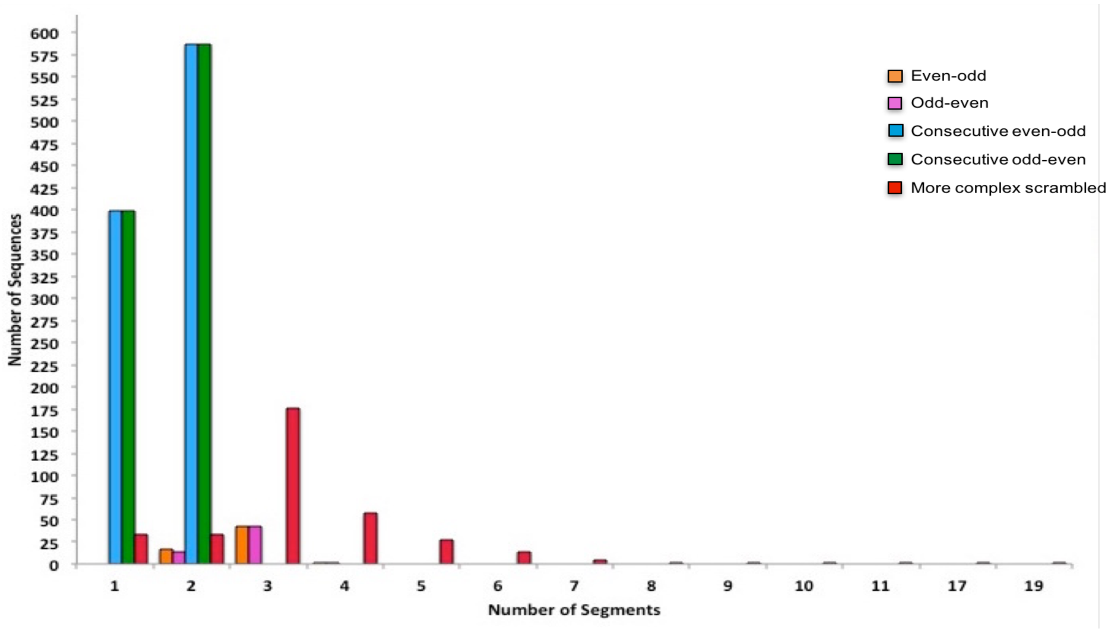


Figure 4.7: Relationship between the number of segments (increasing and decreasing subwords) and the number of sequences in the dataset achieving that for renumbered unidirectional sequences.

is 1 3 5 7 9 2 4 6 8 10, with one stop point at 2, and two increasing subwords: 1 3 5 7 9 and 2 4 6 8 10. Thus, the sequences with a length greater than 2, and having consecutive odd-even, or consecutive even-odd pattern will always have two increasing subwords — one with the consecutive odd numbers, and the other with the consecutive even numbers. This is because the sequences having these patterns with length greater than 2 will always have a single stop point at the ending of consecutive odd/even numbers, and the starting of consecutive even/odd numbers, respectively. The consecutive odd/even patterns sequences will have only 2 increasing subwords, and can be descrambled computationally by only one application of shuffle operation between the consecutive odd numbers, and consecutive even numbers. Under this system, the permutations that have its elements in descending order will have only 1 subword. And, the other sequences will have at least 2 subwords. For example, the following sequence -3 -8 -2 -7 -1 -6 -11 -5 -4 -10 -9, goes under the more complex scrambled pattern, and has 4 stop points, and 5 subwords: 3 decreasing subwords -3 -8, -2 -7, and -1 -6 -11, and 2 increasing subwords -5 -4 and -10 -9. The graphs in Figures 4.7 and 4.8 show the visualisations of the number of segments versus the number of sequences achieving that number of segments for renumbered unidirectional and renumbered extracted unidirectional sequences, respectively. We can see in the graphs that in this system the minimum number of subwords for both of the datasets is 1, because the sequences in decreasing or descending order have only 1 decreasing subword. Moreover, a large number of consecutive odd/even sequences have only 1 subword. In addition, in the dataset, a large number of sequences are exactly the permutation 2 1.

Algorithm 2 runs in linear time in the size of the input permutation, and is therefore very simple to calculate optimally. Table 4.3 and 4.4 show the average number of subwords determined by Algorithm 2,

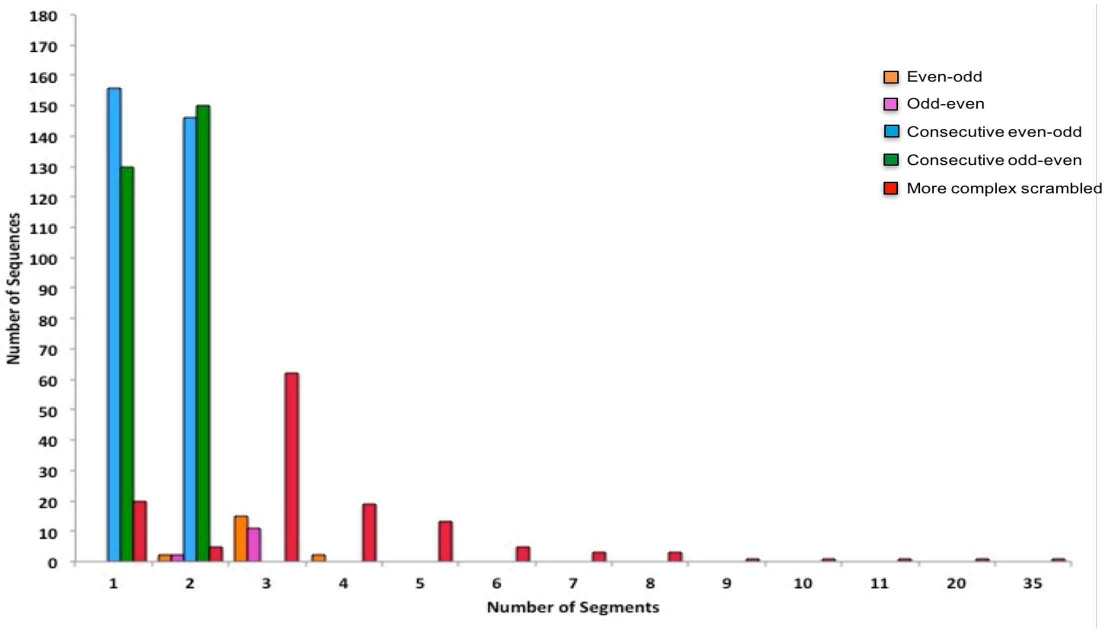


Figure 4.8: Relationship between the number of segments (increasing and decreasing subwords) and the number of sequences in the dataset achieving that for renumbered extracted unidirectional sequences.

along with the maximum number of maximal positive or negative, increasing or decreasing subwords, number of sequences in each sequence pattern, average length of subwords, and the maximum length of subwords for renumbered unidirectional and renumbered extracted unidirectional scrambled sequences.

Tables 4.1, 4.2, 4.3, and 4.4 describe the results of CIS and CIDS. The difference in average length of subwords, and the maximum length of subwords for both CIS and CIDS infers that most of the segments are not very long. The average number of segments shows that considering decreasing subwords reduces the average number of applications of shuffles. Indeed, minimizing the number of segments is equivalent to minimizing the number of applications of shuffle. The average number of segments came down to 3.37 from 3.799 for renumbered unidirectional sequences, and to 3.83 from 4.37 for renumbered extracted unidirectional scrambled sequences with Algorithm 2. Moreover, the average results for consecutive odd-even and consecutive even-odd pattern sequences decreased — 2 to approximately 1.5, because all the consecutive even-odd sequences that are 2 1 will have only 1 subword with this algorithm. Thus, the reductions in the average number of subwords considering decreasing subwords in CIDS indicates the existence of chunks of MDSs in the sequences where all the MDSs are in descending order.

4.5 Non-Contiguous Increasing System

This system allows shuffle on increasing subsequences (i.e. non-contiguous) instead of increasing subwords only. As discussed above, if $\pi = \pi_1 \cdots \pi_n$ is a given input permutation, then a NIS partition of π is a sequence of increasing subsequences $s_1, \ldots, s_m$, such that $\pi \in s_1 \sqcup s_2 \sqcup \cdots \sqcup s_m$, and the identity permutation is in

Table 4.3: Subword statistics with Contiguous Increasing and Decreasing System for renumbered unidirectional sequences.

Sequence patterns	**No. of Sequences**	**Avg. number of subwords**	**Max. number of subwords**	**Avg. length of subwords**	**Max. length of subwords**
Consecutive odd-even patterns	986	1.595	2	3.47	20
Consecutive even-odd patterns	985	1.594	2	4.078	25
Odd-even patterns	58	2.775	4	3.297	10
Even-odd patterns	60	2.766	4	4.578	21
More complex scrambled patterns	354	3.37	19	3.861	43

Table 4.4: Subword statistics with Contiguous Increasing and Decreasing System for renumbered extracted unidirectional sequences.

Sequence patterns	**No. of Sequences**	**Avg. number of subwords**	**Max. number of subwords**	**Avg. length of subwords**	**Max. length of subwords**
Consecutive odd-even patterns	280	1.535	2	2.323	20
Consecutive even-odd patterns	302	1.483	2	2.488	25
Odd-even patterns	13	2.846	3	3.05	10
Even-odd patterns	19	3	4	4.423	21
More complex scrambled patterns	135	3.83	35	3.383	86

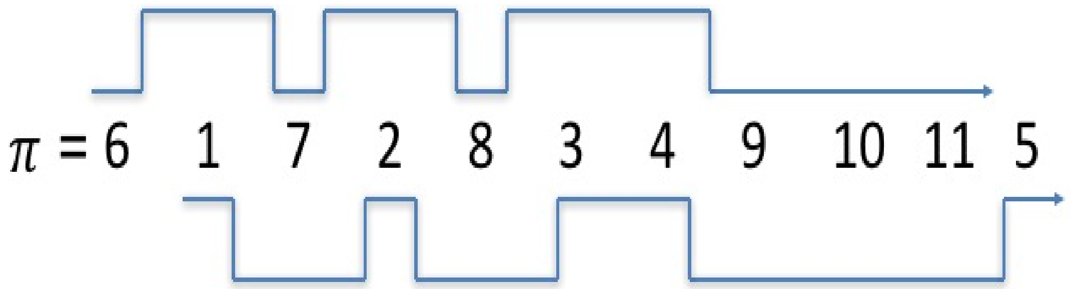


Figure 4.9: Example of a MIC chromosome fragment that has 2 increasing subsequences to descramble.

$\bar{s_1} \sqcup \cdots \sqcup \bar{s_m}$. Note again that, $\bar{s_i}$, for each i, $1 \leq i \leq m$ is s_i if s_i is positive, and the inversion of s_i if s_i is negative. Figure 4.9 shows an example of increasing subsequences in a permutation π, where both the input permutation and the identity permutation is in 6 8 9 10 11 $\sqcup$ 1 2 3 4 5. The following sections discuss the implemented algorithms for determining the segments within the non-contiguous increasing system. Four separate algorithms were created using different techniques to try to reach optimality. The reason that multiple algorithms are presented is because an optimal algorithm has not been found for this system (more precisely, we are not sure whether any of the algorithm are optimal).

4.5.1 Determining Segments by Eliminating Cut-off Points

For the contiguous increasing system, the optimal number of segments was cut-off point dependent. Hence, we hypothesized that removing at least one cut-off point with each segment would give optimal results for non-contiguous increasing systems also. Thus the algorithm *CutOffPointsRemoval* (Algorithm 3) was implemented based on removing at least one cut-off point for each increasing subsequence. The main strategy of this algorithm is to continually add an element to the end of an increasing subsequence s from π, if it is bigger than the last element of s, and adding the new element from π would eliminate a cut-off point in π. Initially, the first element of s is π_1, and the algorithm keeps searching for elements to add to s throughout π. If no element of π satisfies these two conditions, then it takes the elements from π_1 to the left of the first cut-off point of π as the increasing subsequence s, as this would eliminate the first cut-off point of π. Then, π is updated by deleting the elements of s from π. The increasing subsequence s is the first segment of π to participate in the shuffle operation. These procedures are repeated until π becomes empty. Thus, the algorithm would always have to take one final increasing subsequence that will not eliminate any cut-off point, as the last increasing subsequence would have the remaining elements of π, where all the elements would already be in ascending order in π.

Algorithm 3 executes the above steps for π, and determines a small, but not always minimal size of an NIS partition of an input permutation π. Tables 4.5 and 4.6 show the average number of increasing subsequences determined by Algorithm 3, along with the maximum number of increasing subsequences, the number of sequences in each sequence pattern, the average length of increasing subsequences, and the maximum length

Algorithm 3 Minimum number of segments in NIS: cut-off points removal.

```
procedure CutOffPointsRemoval($\pi = \pi_1 \cdots \pi_n$)
    extend $\pi$ by adding $\pi_0 \leftarrow 0$ in $\pi$;        ▷ Thus $\pi = \pi_0\pi_1\pi_2 \cdots \pi_{n-1}\pi_n$ and $\pi_0 = 0$.
    segments $\leftarrow 0$;
    while $\pi$ is not empty do
        element $\leftarrow \pi_1$;
        for $i \leftarrow 1$ to $n-1$ do
            if $(\pi_i > \pi_{i+1})$ && $(element \leq \pi_i)$ && $(\pi_{i-1} < \pi_{i+1})$ then
                add $\pi_i$ in the subsequence $s$;
                element $\leftarrow \pi_i$;
            end if
            if $(i == n-1)$ then
                if $(\pi_i > \pi_{i+1})$ && $(element < \pi_{i+1})$ then
                    add $\pi_n$ in $s$;
                else
                    if the first element after last cut-off point in $\pi >$ element then
                        add all of the elements from first element after last cut-off point to $n$ of $\pi$ in $s$;
                    end if
                end if
            end if
        end for
        if ($s$ is empty) && (length($\pi$) $\neq 1$) then
            add $\pi_1$ to all elements before first cut-off point of $\pi$ in $s$;
        end if
        if $s$ is not empty then
            segments $\leftarrow$ segments $+ 1$;
            delete the elements of $s$ from $\pi$;
        else
            if length($\pi$) $== 1$ then
                remove $\pi_0$ from $\pi$;
            end if
        end if
        Clear $s$;
    end while
    return segments;
end procedure
```

Table 4.5: Increasing subsequence statistics with Non-Contiguous Increasing System for renumbered unidirectional sequences with Algorithm 3.

Sequence patterns	**No. of Sequences**	**Avg. number of increasing subsequences**	**Max. number of increasing subsequences**	**Avg. length of increasing subsequences**	**Max. length of increasing subsequences**
Consecutive odd-even patterns	986	2	2	2.300	36
Consecutive even-odd patterns	985	2	2	2.305	37
Odd-even patterns	58	3.017	4	2.44	10
Even-odd patterns	60	2.883	4	2.901	26
More complex scrambled patterns	354	3.037	7	3.492	43

of increasing subsequences for renumbered unidirectional and renumbered extracted unidirectional scrambled sequences.

The results of Tables 4.3 and 4.4 in contrast to Tables 4.5 and 4.6 show that the average number of segments for all types (both consecutive and non-consecutive) of odd-even pattern and even-odd pattern sequences is less for CIDS, but Algorithm 3 for NIS has reduced the average number of segments from 3.37 to 3.037 and 3.83 to 3.57 for more complex scrambled pattern sequences of renumbered unidirectional and renumbered extracted unidirectional, respectively. However, we are more interested in comparisons of CIS to NIS, as both do not allow decreasing segments. Tables 4.1 and 4.2, in contrast to Tables 4.5 and 4.6, show that non-contiguous increasing system reduces the average number of segments for all types of sequences in comparison with contiguous increasing system. Algorithm 3 for NIS has decreased the average number of segments notably for even-odd pattern and more complex scrambled sequences of renumbered unidirectional dataset to 2.883 from 3.067 and 3.037 from 3.799, respectively. Furthermore, for the renumbered extracted unidirectional sequences, the average number of segments remains the same for even-odd pattern sequences, but decreased for more complex scrambled sequences to 3.57 from 4.37 (Tables 4.6 and 4.2). Note that Algorithm 3 for NIS gave the optimal number of segments for the consecutive even/odd sequences for both of the datasets, as the average number of segments for these sequences are 2, and any scrambled sequences must have at least 2 increasing segments. Moreover, Algorithm 3 for NIS has decreased the maximum number

Table 4.6: Increasing subsequence statistics with Non-Contiguous Increasing System for renumbered extracted unidirectional sequences with Algorithm 3.

Sequence patterns	**No. of Sequences**	**Avg. number of increasing subsequences**	**Max. number of increasing subsequences**	**Avg. length of increasing subsequences**	**Max. length of increasing subsequences**
Consecutive odd-even patterns	280	2	2	2.323	20
Consecutive even-odd patterns	302	2	2	2.488	25
Odd-even patterns	13	3	3	3.128	10
Even-odd patterns	19	3.105	3	4.423	21
More complex scrambled patterns	135	3.57	9	4.141	86

of required segments notably, as the maximum number of segments with Algorithm 3 is 7 for renumbered unidirectional sequences, and 9 for renumbered extracted unidirectional sequences, whereas with CIS, this number is 20 (19 with CIDS) and 35 (35 with CIDS), respectively. Thus, allowing subsequences instead of subwords seems to drastically reduce the maximum number of parallel operations.

However, there were some sequences where Algorithm 3 failed to give the smallest number of increasing subsequences, and therefore the algorithm is not optimal. For example, the sequence 3 8 4 9 5 10 1 6 2 7 has 3 increasing subsequences, but Algorithm 3 determines 4 increasing subsequences. The segments determined by Algorithm 3 are the following, listed in order (3 8 9), (4 6), (5 10), (1 2 7). That said, this sequence can be partitioned into the 3 increasing subsequences (3 8 9 10), (4 5 6 7), and (1 2). Another example of this scenario is: -8 -17 -7 -16 -6 -15 -5 -14 -4 -13 -3 -12 -2 -11 -1 -10 -19 -9 -18. This sequence can be partitioned into 3 increasing subsequences, but Algorithm 3 gave 4 for the number of segments, which is not optimal. The subsequences determined by Algorithm 3 are (-8 -7 -6 -5 -4 -3 -2 -1), (-17 -9), (-16 -15 -14 -13 -12 -11 -10), and (-19 -18). One of the optimal NIS partition of this sequence with minimum number of segments of 3 would be (-8 -7 -6 -5 -4 -3 -2 -1), (-17 -16 -15 -14 -13 -12 -11 -10 -9), and (-19 -18).

The above examples indicate that slightly modifying the algorithm to consider the next consecutive numbers might lead to better solutions. Thus, a new algorithm was implemented by updating the *CutOffPointsRemoval* algorithm to get the smaller size of a NIS partition. This new algorithm, *CutOfPointsRemovalUpdated*, is discussed in the next section.

4.5.2 Considering Consecutive Numbers with Eliminating Cut-off Points

As the following algorithm *CutOfPointsRemovalUpdated* (Algorithm 4) is a modified version of *CutOffPointsRemoval* algorithm (Algorithm 3), the main strategy for both is the same. The only difference is, Algorithm 4 adds an element in the increasing subsequence s from π, if it is a greater element than the last element of s, and taking the new element from π would eliminate a cut-off point in π, **or the new element is one larger than the last element of** s (i.e. $\text{new\_element\_of\_}\pi = \text{last\_element\_of\_}s + 1$). The rest of the constraints on determining increasing subsequences are exactly the same as Algorithm 3. Next, π is updated by deleting the elements of s from π, and the increasing subsequence s is the segment of π to participate in the shuffle operation. Similarly, these procedures are repeated for π until the length of π becomes 0.

Another difference with Algorithm 3 is that, Algorithm 4 might have more than one increasing subsequence that does not eliminate any cut-off points, because each time an increasing subsequence adds a consecutive element that does not eliminate any cut-off points, there is a possibility that the increasing subsequence might not eliminate any cut-off points. This might be the reason behind this egregious breakdown of Algorithm 4 (Table 4.7, 4.8). Algorithm 4 has increased the number of increasing subsequences, and the average number of increasing subsequences for many sequences in comparison with Algorithm 3. The average number of increasing subsequences became 3.933 from 2.883 for even-odd pattern sequences, 3.724 from 2.901 for odd-even pattern sequences, and 4.008 from 3.037 for more complex scrambled sequences for renumbered unidirectional sequences (Tables 4.5, 4.7). Similar types of increases were also noted with renumbered extracted unidirectional sequences (Tables 4.6, 4.8).

Moreover, Algorithm 4 gave a maximum number of increasing subsequences of 89 instead of 9 for Algorithm 3. The maximum number of increasing subsequences with Algorithm 3 was 3, 3, and 9 for even-odd patterned sequences, odd-even patterned sequences, and more complex scrambled sequences, respectively, whereas with Algorithm 4 these numbers are 11, 9, and 19, respectively (Tables 4.5, 4.7) for renumbered unidirectional sequences. In addition, Algorithm 3 can partition the consecutive odd-even and consecutive even-odd sequences into 2 increasing subsequences only, but with Algorithm 4, the maximum number of increasing subsequences for renumbered unidirectional consecutive odd-even and consecutive even-odd sequences are 35 and 32, respectively. And, for renumbered extracted unidirectional consecutive odd-even and consecutive even-odd sequences, these numbers are 19 and 20, respectively.

Algorithm 4 Minimum number of segments in NIS: cut-off points removal updated.

```
procedure CutOfPointsRemovalUpdated($\pi = \pi_1 \cdots \pi_n$)
    extend $\pi$ by adding $\pi_0 \leftarrow 0$ in $\pi$;            ▷ Thus $\pi = \pi_0\pi_1\pi_2 \cdots \pi_{n-1}\pi_n$ and $\pi_0 = 0$.
    segments $\leftarrow 0$;
    while $\pi$ is not empty do
        element $\leftarrow \pi_1$;
        index_of_element $\leftarrow 1$;
        for $i \leftarrow 1$ to $n-1$ do
            if $(\pi_i > \pi_{i+1})$ && $(element \leq \pi_i)$ && $(\pi_{i-1} < \pi_{i+1})$ then
                add $\pi_i$ in the subsequence $s$;
                element $\leftarrow \pi_i$;
                index_of_element $\leftarrow i$;
            end if
            if $(element + 1 == \pi_i)$ then
                element $\leftarrow \pi_i$;
                add $\pi_{index\_of\_element}$ in the subsequence $s$;
                index_of_element $\leftarrow i$;
                add $\pi_i$ in the subsequence $s$;
            end if
            if $(i == n-1)$ then
                if $(\pi_i > \pi_{i+1})$ && $(element < \pi_{i+1})$ then
                    add $\pi_n$ in $s$;
                else
                    if the first element after last cut-off point in $\pi >$ element then
                        add all of the elements from first element after last cut-off point to $n$ of $\pi$ in $s$;
                    end if
                end if
            end if
        end for
        if ($s$ is empty) && ($length(\pi) \neq 1$) then
            add $\pi_1$ to all elements before first cut-off point of $\pi$ in $s$;
        end if
        if $s$ is not empty then
            segments $\leftarrow$ segments $+ 1$;
            delete the elements of $s$ from $\pi$;
        else
```

Table 4.7: Increasing subsequence statistics with Non-Contiguous Increasing System for renumbered unidirectional sequences with Algorithm 4.

Sequence patterns	**No. of Sequences**	**Avg. number of increasing subsequences**	**Max. number of increasing subsequences**	**Avg. length of increasing subsequences**	**Max. length of increasing subsequences**
Consecutive odd-even patterns	986	2.665	35	1.726	23
Consecutive even-odd patterns	985	2.66	32	1.733	38
Odd-even patterns	58	3.724	9	1.976	7
Even-odd patterns	60	3.933	11	2.046	20
More complex scrambled patterns	354	4.008	19	2.645	29

```
36:         if $length(\pi) == 1$ then
37:             $remove\ \pi_0\ from\ \pi$;
38:         end if
39:     end if
40:     $Clear\ s$;
41: end while
42: return $segments$;
43: end procedure
```

Thus, the additional restriction of Algorithm 4 on increasing subsequences increases the number of subsequences immensely. For example, the following sequence: 2 4 6 8 10 12 14 16 18 1 3 5 7 9 11 13 15 17 19 has 2 increasing subsequences — one of the possible 2 increasing subsequences are 2 4 6 8 10 12 14 16 18 and 1 3 5 7 9 11 13 15 17 19. Though Algorithm 3 can partition this sequence into 2 increasing subsequences, Algorithm 4 partitions it into 10 increasing subsequences. This is because, at first, Algorithm 3 adds 2 in the first increasing subsequence s, and when it sees that there is one cut-off point in between 18 and 1, and taking 18 will not eliminate any cut-off points, it takes all the elements from 2 to 18 for the first increasing subsequence. But, as Algorithm 4 considers the consecutive numbers also, instead of taking the elements from

Table 4.8: Increasing subsequence statistics with Non-Contiguous Increasing System for renumbered extracted unidirectional sequences with Algorithm 4.

Sequence patterns	**No. of Sequences**	**Avg. number of increasing subsequences**	**Max. number of increasing subsequences**	**Avg. length of increasing subsequences**	**Max. length of increasing subsequences**
Consecutive odd-even patterns	280	2.696	19	1.723	15
Consecutive even-odd patterns	302	2.788	20	1.785	26
Odd-even patterns	13	4.462	10	2.103	6
Even-odd patterns	19	7.053	22	1.947	5
More complex scrambled patterns	135	5.356	89	2.760	28

2 to 18, it adds 3 to the first increasing subsequence. Thus, for Algorithm 4, $s_1 = 2\ 3$, and for Algorithm 3, $s_1 = 2\ 4\ 6\ 8\ 10\ 12\ 14\ 16\ 18$. In this way, the minimum number of increasing subsequences increases for Algorithm 4.

Therefore, cut-off eliminating algorithms (Algorithms 3 and 4) for non-contiguous systems could not achieve optimality. This is because subsequences can have elements that are not in continuous positions in π, thus the elements of subsequences may appear in between different cut-off points in π. That is, suppose, $\pi = 2\ \mathbf{5}\ \mathbf{4}\ \mathbf{3}\ 6\ \mathbf{7}\ \mathbf{1}\ 8\ 9$, thus, the cut-off points are c_1 in between 5, 4, c_2 in between 4, 3, and c_3 in between 7, 1. An increasing subsequence can have 2 5 6 7 8 9, and the remaining $\pi = 4\ 3\ 1$ with 2 cut-off points, 4 3 and 3 1. Thus, the increasing subsequences can have any element from the start to c_1 (2 5), c_2 to c_3 (6 7) and c_3 to the end of π (8 9). In contrast, subwords in CIS can have all the elements in between two consecutive cut-off points (3 6 7), or the start to the first cut-off point (2 5), or last cut-off point to the end of π (1 8 9). Hence, implementing algorithms based on cut-off points for NIS might not give optimality. Therefore, the following algorithm was implemented with a new idea of considering the longest increasing subsequence of permutation, π.

4.5.3 Longest Increasing Subsequences

The non-optimal results of Algorithms 3 and 4 give the indication that for non-contiguous increasing system, cut-off points might not have as much importance as they had in contiguous increasing system. Thus, an algorithm to "cover" the maximum increasing elements of π by increasing subsequences was created using graphs. Hence, the algorithm *LongestIncreasingSubsequence* (Algorithm 5) was implemented. This algorithm applies graph theoretic techniques to get the longest increasing subsequence in a permutation π. Initially, Algorithm 5 makes a directed graph $G = (V, E)$ from π with restrictions on the edges of the graph to determine the longest increasing subsequence. The longest increasing subsequence of π is the longest path in graph G from first node to the last node. Here, a vertex is made for each element of π, and there is an edge between π_i and π_j in G, if and only if $\pi_i < \pi_j$ and $i < j$ for all $i, j \in 1, 2, 3, \ldots, n$, with the first node π_1 and the last node π_n. Then, the algorithm determines the longest path optimally in graph G with the Bellman-Ford algorithm, which is an algorithm that computes longest paths from a single source vertex to all of the other vertices in a negative weighted directed acyclic graph [11]. Then, the vertices of the longest path are the elements of the first increasing subsequence s. Note that the order of vertices in the longest path is preserved in the increasing subsequences as well. Next, the algorithm deletes the elements of the increasing subsequence s from π, and repeats the above steps until π becomes empty. In the end, the minimum size of the NIS partitioning is the final number of increasing subsequences.

Algorithm 5 Minimum number of segments in NIS: longest increasing subsequence.

```
1: procedure LongestIncreasingSubsequence($\pi = \pi_1 \cdots \pi_n$)
2:     segments ← 0;
3:     while $length(\pi) > 0$ do
4:         extend $\pi$ by adding $\pi_0 \leftarrow 0$ and $\pi_{n+1} \leftarrow n+1$ in $\pi$;    ▷ Thus $\pi = \pi_0\pi_1\pi_2\cdots\pi_{n-1}\pi_n\pi_{n+1}$ and $\pi_0 = 0,\ \pi_{n+1} = n+1$.
5:         Make a directed acyclic graph, $G = (V, E)$ from $\pi$ where,
6: $V = \{\pi_0\pi_1, \pi_2, \ldots, \pi_{n-1}, \pi_n, \pi_{n+1}\}$ and an edge between $\pi_i$ and $\pi_j$ if $\pi_i < \pi_j$ and $i < j$, for all $i, j \in 0, 1, 2, 3, \ldots, n-1, n, n+1$;
7:         Find longest path in G from $\pi_0$ to $\pi_{n+1}$, add those vertices except $\pi_0$ and $\pi_{n+1}$ in subsequence $s$;
8:         segments ← segments + 1;
9:         delete the elements of $s$ from $\pi$;
10:        delete $\pi_0$ and $\pi_{n+1}$ from $\pi$;
11:        Clear $s$;
12:    end while
13:    return segments;
14: end procedure
```

Table 4.9 and 4.10 show the average number of increasing subsequences determined by Algorithm 5, along

Table 4.9: Increasing subsequence statistics with Non-Contiguous Increasing System for renumbered unidirectional sequences with Algorithm 5.

Sequence patterns	**No. of Sequences**	**Avg. number of increasing subsequences**	**Max. number of increasing subsequences**	**Avg. length of increasing subsequences**	**Max. length of increasing subsequences**
Consecutive odd-even patterns	986	2	2	2.301	36
Consecutive even-odd patterns	985	2	2	2.306	38
Odd-even patterns	58	2.966	4	2.483	11
Even-odd patterns	60	2.9	5	2.885	27
More complex scrambled patterns	354	2.992	6	3.545	44

with the maximum number of increasing subsequences, number of sequences in each sequence pattern, average length of increasing subsequences, and the maximum length of increasing subsequences for renumbered unidirectional and renumbered extracted unidirectional scrambled sequences. In comparison with Algorithm 4, Algorithm 5 has reduced both the maximum number of increasing subsequences and the average number of increasing subsequences for all types of sequences notably (Tables 4.7, 4.8, 4.9 and 4.10). There is a small increase in the maximum number of increasing subsequences for even-odd patterned sequences in comparison with Algorithm 3 — maximum number of increasing subsequences became 5 from 4 for renumbered unidirectional sequences (Tables 4.5, 4.9), and 4 from 3 for renumbered extracted unidirectional sequences (Tables 4.6, 4.10). The average number of increasing subsequences for the renumbered unidirectional even-odd pattern sequences increased slightly, from 2.883 to 2.9. However, Algorithm 5 has reduced the average number of increasing subsequences for the rest of the sequences (Tables 4.5, 4.6, 4.9 and 4.10).

Though the average size of NIS partitioning with Algorithm 5 (Tables 4.9, 4.10) is quite satisfactory, the increase in the average number of increasing subsequences for renumbered unidirectional even-odd sequences with Algorithm 5 over Algorithm 3 (2.9 from 2.883 respectively) signifies that there must be some sequences where Algorithm 5 could not give optimal results. There is such a sequence of even-odd pattern that was partitioned into 5 increasing subsequences by Algorithm 4, and can be partitioned into 3 increasing subsequences by Algorithm 3. However, Algorithm 5 partitioned it into 4 increasing subsequences as the smallest

Table 4.10: Increasing subsequence statistics with Non-Contiguous Increasing System for renumbered extracted unidirectional sequences with Algorithm 5.

Sequence patterns	**No. of Sequences**	**Avg. number of increasing subsequences**	**Max. number of increasing subsequences**	**Avg. length of increasing subsequences**	**Max. length of increasing subsequences**
Consecutive odd-even patterns	280	2	2	2.323	20
Consecutive even-odd patterns	302	2	2	2.488	26
Odd-even patterns	13	2.846	3	3.297	11
Even-odd patterns	19	3.105	4	4.424	22
More complex scrambled patterns	135	3.415	9	4.33	92

NIS partitioning, which is not optimal. The sequence is 4 8 2 6 1 3 5 7 9. This example sequence can be partitioned into 3 increasing subsequences (4 8 9), (2 6 7), and (1 3 7). But, Algorithm 5 gave 4 as the smallest size of NIS partitioning, (4 6 7 9), (2 3 5), (8), and (1). Thus, 4 is not the small NIS partitioning for this sequence, as it can be partitioned into 3 increasing subsequences also. Another example of this scenario is the following sequence: 4 8 10 2 5 7 9 1 3 6. Algorithm 5 partitioned this sequence into 4 increasing subsequences (4 5 7 9), (2 3 6), (8 10), and (1), whereas it can be partitioned into 3 increasing subsequences only, (4 8 10), (2 5 7 9), and (1 3 6).

Hence, in the above examples taking the longest increasing subsequence first changes the remaining options for further increasing subsequences. Therefore, the final number of increasing subsequence can increase. Thus, Algorithm 5 could not give the optimal results with the heuristic of repeatedly finding the longest increasing subsequences. Overall, algorithm *LongestIncreasingSubsequence* gives better results than algorithm *CutOfPointsRemovalUpdated* and algorithm *CutOffPointsRemoval*, and is the best among these 3 algorithms (Tables 4.5, 4.6, 4.7, 4.8, 4.9 and 4.10), although there are specific examples where others are better. Therefore, algorithm *LongestIncreasingSubsequence* also could not reach optimality for all of the sequences. Hence, the following final algorithm was implemented to try to determine the smallest size of NIS partitioning.

4.5.4 Considering Next Increasing Element in a Subsequence

The final algorithm is based on the idea of adding the next increasing element of the input permutation to an increasing subsequence to determine the optimal size of NIS partitioning. Indeed, the optimal partitioning for the examples above can be achieved by adding the next larger element of π to the subsequence instead of looking for the longest increasing subsequence. Therefore, the following algorithm (Algorithm 6) finds the number of increasing subsequences of π. At first, the algorithm adds the first element of π as the first element of increasing subsequence s. Then it finds the next larger element in π, adds it to increasing subsequence s, and continues doing this until reaching the end of π. Thus, the increasing subsequence s is the first increasing subsequence of π. Next, the algorithm deletes the elements of this increasing subsequence s from π, and repeatedly finds the next increasing subsequence s in a similar fashion until π becomes empty. In the end, the final number of increasing subsequences is the size of NIS partitioning of the input permutation determined by Algorithm 6.

Algorithm 6 Minimum number of segments in NIS: subsequence with next increasing elements.

```
1: procedure NextIncreasingElement($\pi = \pi_1 \cdots \pi_n$)
2:     segments ← 0;
3:     while $\pi$ is not empty do
4:         element ← $\pi_1$;
5:         add element in subsequence $s$;
6:         for $i \leftarrow 2$ to $n$ do
7:             if element < $\pi_i$ then
8:                 element ← $\pi_i$;
9:                 add element in subsequence $s$;
10:            end if
11:        end for
12:        if $s$ is not empty then
13:            segments ← segments + 1;
14:            delete the elements of $s$ from $\pi$;
15:        end if
16:        Clear $s$;
17:    end while
18:    return segments;
19: end procedure
```

Tables 4.11 and 4.12 show the average number of increasing subsequences determined by Algorithm 6, along with the maximum number of increasing subsequences, number of sequences in each sequence pattern, average length of increasing subsequences, and the maximum length of increasing subsequences for

Table 4.11: Increasing subsequence statistics with Non-Contiguous Increasing System for renumbered unidirectional sequences with Algorithm 6.

Sequence patterns	**No. of Sequences**	**Avg. number of increasing subsequences**	**Max. number of increasing subsequences**	**Avg. length of increasing subsequences**	**Max. length of increasing subsequences**
Consecutive odd-even patterns	986	2	2	2.301	36
Consecutive even-odd patterns	985	2	2	2.306	38
Odd-even patterns	58	2.948	3	2.497	11
Even-odd patterns	60	2.867	4	2.918	26
More complex scrambled patterns	354	2.918	6	3.634	43

renumbered unidirectional and renumbered extracted unidirectional scrambled sequences. The results of Algorithm 6 (Table 4.11 and 4.12) demonstrates that it gives the smallest average number of increasing subsequences with the smallest maximum number of increasing subsequences for all types of sequences among the implemented NIS algorithms (Table 4.13 and 4.14).

Moreover, a comparison among the detailed results of Algorithms 3, 5, and 6 shows that Algorithm 6 gives the best results for all of the renumbered unidirectional and renumbered extracted unidirectional sequences. This means, in all of the cases from renumbered unidirectional and renumbered extracted unidirectional sequences, this algorithm successfully gives the smallest size of NIS partitioning among Algorithms 3, 5, and 6. Algorithm 4 was excluded from this comparison, because it gave the highest NIS partitioning size for all types of sequences among the implemented NIS algorithms (Table 4.13 and 4.14) — and we are looking for the smallest NIS partitioning size.

Compared to Algorithm 6, the *LongestIncreasingSubsequence* algorithm (Algorithm 5) gives almost as small an NIS partitioning for the input sequences. For renumbered unidirectional sequences, there were 2 even-odd pattern sequences, 1 odd-even pattern sequence, and 25 more complex scrambled pattern sequences that Algorithm 5 failed to give as good a result as Algorithm 6. From renumbered extracted unidirectional sequences, there were 13 more complex scrambled pattern sequences that Algorithm 6 gave a better result than Algorithm 5. Thus, in total there were 41 sequences of 3192 sequences that Algorithm 6 partitioned

Table 4.12: Increasing subsequence statistics with Non-Contiguous Increasing System for renumbered extracted unidirectional sequences with Algorithm 6.

Sequence patterns	**No. of Sequences**	**Avg. number of increasing subsequences**	**Max. number of increasing subsequences**	**Avg. length of increasing subsequences**	**Max. length of increasing subsequences**
Consecutive odd-even patterns	280	2	2	2.323	20
Consecutive even-odd patterns	302	2	2	2.488	26
Odd-even patterns	13	2.846	3	3.297	11
Even-odd patterns	19	3.105	4	4.424	22
More complex scrambled patterns	135	3.274	7	4.515	87

Table 4.13: Comparison of average number of increasing subsequences for renumbered unidirectional sequences among implemented NIS algorithms.

	Average number of increasing subsequences			
Sequence patterns	Algorithm 3	Algorithm 4	Algorithm 5	Algorithm 6
Consecutive odd-even patterns	2	2.665	2	2
Consecutive even-odd patterns	2	2.66	2	2
Odd-even patterns	3.017	3.724	2.966	2.948
Even-odd patterns	2.883	3.933	2.9	2.867
More complex scrambled patterns	3.037	4.008	2.992	2.918

Table 4.14: Comparison of average number of increasing subsequences for renumbered extracted unidirectional sequences among implemented NIS algorithms.

	Average number of increasing subsequences			
Sequence patterns	Algorithm 3	Algorithm 4	Algorithm 5	Algorithm 6
Consecutive odd-even patterns	2	2.696	2	2
Consecutive even-odd patterns	2	2.788	2	2
Odd-even patterns	3	4.462	2.846	2.846
Even-odd patterns	3.105	7.053	3.105	3.105
More complex scrambled patterns	3.57	5.356	3.415	3.274

into a smaller size than Algorithm 5, which is 1.284% of the sequences. Similarly, there were 80 sequences (2.506%) that *CutOffPointsRemoval* algorithm (Algorithm 3) could not give as good a NIS partitioning in comparison with Algorithm 6. Of the 80 sequences, 34 more complex scrambled sequences were from renumbered extracted unidirectional dataset, 1 even-odd pattern sequence, 4 odd-even pattern sequences, and 41 more complex scrambled sequences were from renumbered unidirectional dataset. Though the performance of *LongestIncreasingSubsequence* algorithm is quite good, *NextIncreasingElement* algorithm gave the best results among the implemented algorithms for NIS partitioning. However, it is an open question as to whether Algorithm 6 is optimal or not.

The graphs in Figures 4.10 and 4.11 show the visualisations of the number of segments (size of NIS partitioning with Algorithm 6) versus the number of sequences achieving that number of segments for renumbered unidirectional and renumbered extracted unidirectional sequences, respectively. The number of segments was determined with the *"NextIncreasingElement"* algorithm, as it gives the best result among the implemented NIS algorithms. Moreover, the graphs again represent the satisfactory results of this algorithm, as most of the more complex scrambled sequences were partitioned into only 3 increasing subsequences by this algorithm, even/odd patterned sequences can be partitioned within 4 increasing subsequences, and the maximum number of increasing subsequences is only 7. Therefore, the excellent performance of *NextIncreasingElement* algorithm demonstrates the benefit of the non-contiguous increasing system.

4.6 Non-Contiguous Increasing and Decreasing System

Finally, this system allows shuffle on increasing and decreasing subsequences (i.e. non-contiguous) instead of increasing and decreasing subwords (i.e. contiguous) only. As discussed above, if $\pi = \pi_1 \cdots \pi_n$ is a given input permutation, then a NIDS partition of π is a sequence of subsequences $s_1, \ldots, s_m$, where s_i for each i,

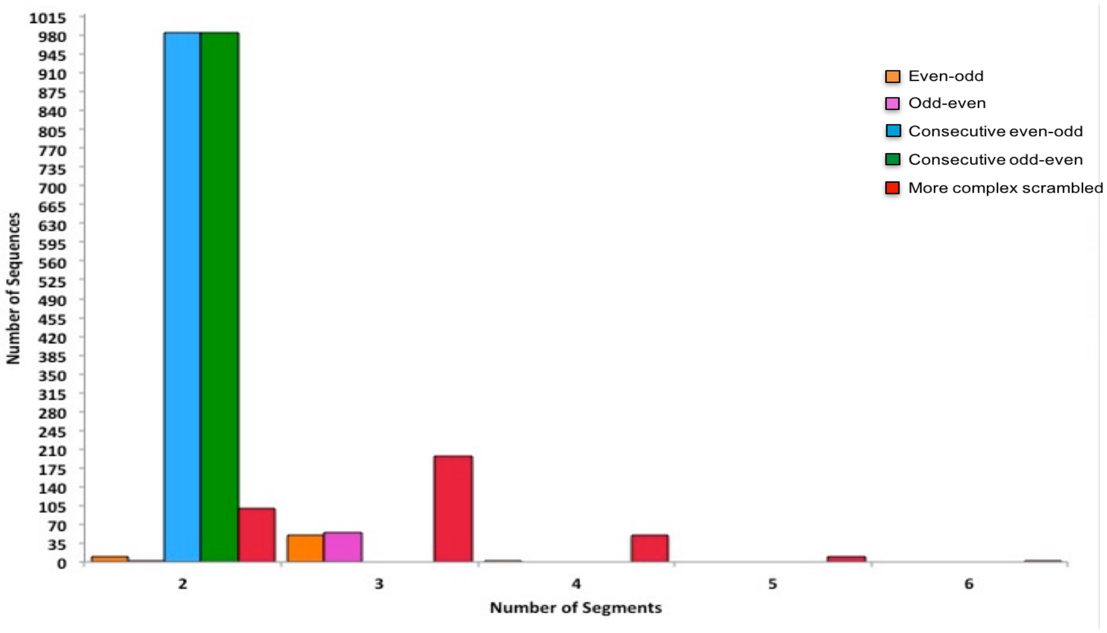


Figure 4.10: Relationship between the number of segments (increasing subsequences) and the number of sequences in the dataset achieving that for renumbered unidirectional sequences as determined by Algorithm 6.

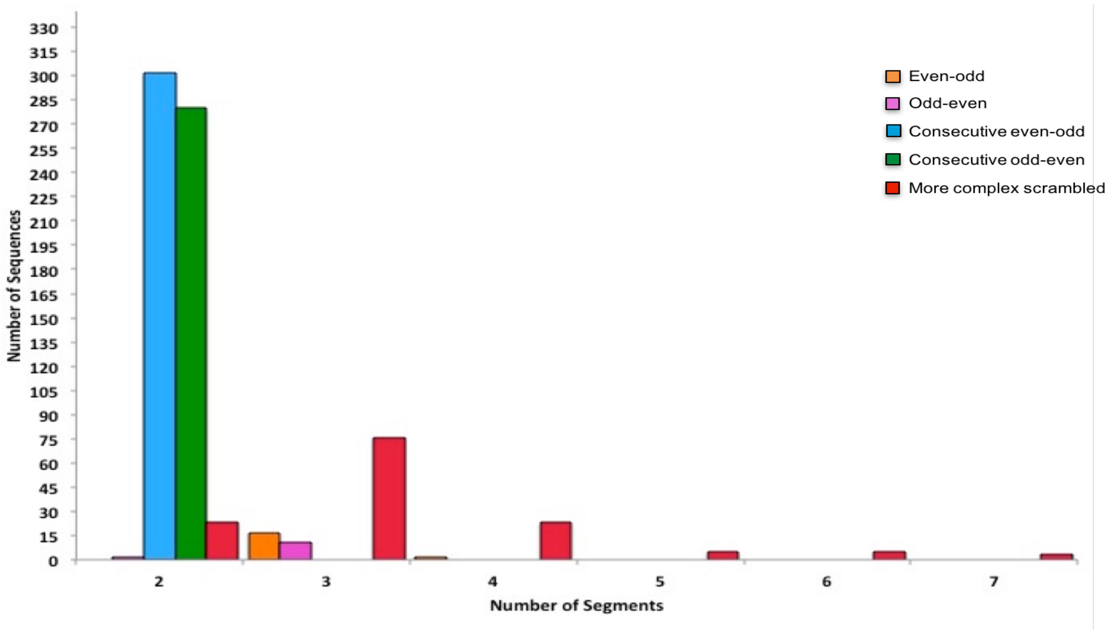
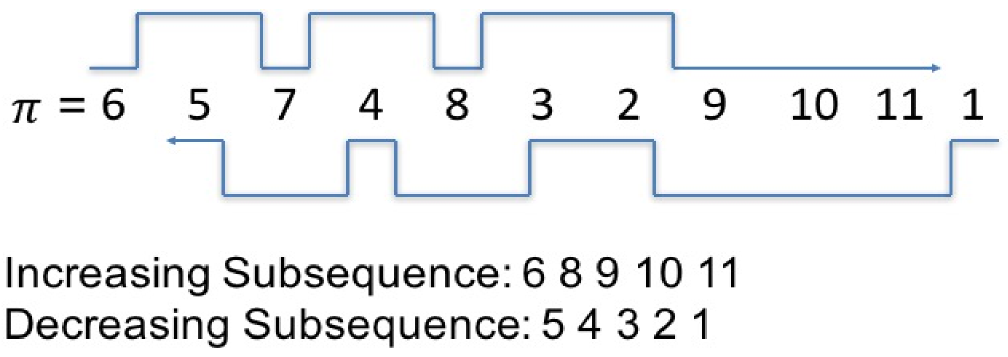


Figure 4.11: Relationship between the number of segments (increasing subsequences) and the number of sequences in the dataset achieving that for renumbered extracted unidirectional sequences as determined by Algorithm 6.



π = 6 5 7 4 8 3 2 9 10 11 1
Increasing Subsequence: 6 8 9 10 11
Decreasing Subsequence: 5 4 3 2 1

Figure 4.12: Example of a MIC chromosome fragment that has 1 increasing subsequence, and 1 decreasing subsequence to descramble.

$1 \leq i \leq m$, is either increasing or decreasing, such that $\pi \in s_1 \sqcup s_2 \sqcup \cdots \sqcup s_m$, and the identity permutation is in $\hat{s_1} \sqcup \cdots \sqcup \hat{s_m}$. Recall that, $\hat{s_i}$, for each i, $1 \leq i \leq m$ is s_i if s_i is positive and increasing, the reversal of s_i if s_i is positive and decreasing, the inversion of s_i if s_i is negative and increasing, and the reversal of ${s_i}^I$ if s_i is negative and decreasing. Figure 4.12 shows an example of increasing and decreasing subsequences in a permutation π, where the input permutation is in 6 8 9 10 11 $\sqcup$ 5 4 3 2 1, and the identity permutation is in 6 8 9 10 11 $\sqcup$ 1 2 3 4 5 (reversal of 5 4 3 2 1). The following sections discuss the implemented algorithms for determining the segments within the non-contiguous increasing and decreasing system. Note that the implemented algorithms only focused on calculating the number of segments, which corresponds to the number of shuffle applications plus one. The number of reversals and inversions are not calculated, and has no impact on results. Two separate algorithms were created using different techniques to try to reach optimality. The reason that multiple algorithms are presented is because an optimal algorithm has not been found for this system as well.

4.6.1 Longest Increasing and/or Decreasing Subsequences

The non-optimal results of the cut-off points eliminating algorithms (Algorithms 3 and 4) of NIS gives the indication that starting with reducing stop points in NIDS will not be a good idea. Moreover, the quite good performance of *LongestIncreasingSubsequence* algorithm for NIS, raises the question about whether an algorithm to "cover" the maximum increasing or decreasing elements of π by increasing and decreasing subsequences can achieve optimality or not. Hence, the algorithm *LongestSubsequence* (Algorithm 7) was implemented using graphs. Similar to Algorithm 5, this algorithm applies graph theoretic techniques to get the longest increasing or decreasing subsequences in a permutation π. Initially, Algorithm 7 makes two different directed graphs $G_I = (V_I, E_I)$ and $G_D = (V_D, E_D)$ from π with restrictions on the edges of the graph to determine both the longest increasing and decreasing subsequences. The longest increasing and decreasing subsequences of π is the longest path in graph G_I and G_D respectively from first node to the last node. Here, a vertex is made for each element of π in both graphs G_I and G_D. There is an edge between π_i

and π_j in G_I, if and only if $\pi_i < \pi_j$ and $i < j$ for all $i, j \in 1, 2, 3, \ldots, n$, with the first node π_1 and the last node π_n. Similarly, in graph G_D, there is an edge between π_i and π_j, if and only if $\pi_i > \pi_j$ and $i < j$ for all $i, j \in 1, 2, 3, \ldots, n$, with the first node π_1 and the last node π_n. Then, the algorithm determines the longest path optimally in graph G_I and G_D with the Bellman-Ford algorithm. The vertices of the longest paths of graphs G_I and G_D are the elements of first increasing subsequence s_I, and first decreasing subsequence s_D, respectively. The algorithm takes the bigger one between s_I and s_D as the first subsequence s (s is either increasing or decreasing). Next, the algorithm deletes the elements of increasing or decreasing subsequence s from π, and repeats the above steps until π becomes empty.

Table 4.15 and 4.16 show the average number of subsequences determined by Algorithm 7, along with the maximum number of increasing or decreasing subsequences, number of sequences in each sequence pattern, average length of subsequences, and the maximum length of subsequences for renumbered unidirectional and renumbered extracted unidirectional scrambled sequences. In comparison with Algorithm 6 for NIS partitioning, Algorithm 7 for NIDS has reduced the average number of segments for all types of sequences by considering the decreasing subsequences (Tables 4.11, 4.12, 4.15 and 4.16). Although, the maximum number of required segments remains the same with both of the Algorithm 6 for NIS and Algorithm 7 for NIDS, the average number of subsequences shows that considering decreasing subsequences reduces the average number of applications of shuffle. However, we are also interested in comparisons of CIDS to NIDS, as both allow decreasing segments. Tables 4.3, 4.4, 4.15, and 4.16 show that non-contiguous increasing and decreasing system gives almost the same average number of segments for consecutive even/odd sequences in comparison with contiguous increasing and decreasing system. However, Algorithm 7 for NIDS has decreased the average number of segments notably for odd-even pattern, even-odd pattern, and more complex scrambled sequences of the both datasets. For renumbered unidirectional dataset the average number of segments for odd-even pattern, even-odd pattern, and more complex scrambled sequences reduced to 2.621 from 2.775, 2.617 from 2.766, and 2.508 from 3.37, respectively (Tables 4.3 and 4.15). And, for the renumbered extracted unidirectional sequences, the average number of segments came down to 2.615 from 2.846, 2.737 from 3, and 2.704 from 3.83 for odd-even pattern, even-odd pattern, and more complex scrambled sequences, respectively (Tables 4.4 and 4.16). Moreover, Algorithm 7 for NIDS has decreased the maximum number of required segments in comparison with CIDS, as the maximum number of segments with Algorithm 7 is 6 for renumbered unidirectional sequences, and 7 for renumbered extracted unidirectional sequences, whereas with CIDS, this number is 19 and 35, respectively. Thus, the consideration of subsequences instead of subwords reduce the number of required segments.

Though Algorithm 7 for NIDS partitioning reduces the average number of segments for all types of sequences (Tables 4.15, 4.16), there were some sequences where Algorithm 7 failed to give smallest number of subsequences, and therefore the algorithm is not optimal. For example, the sequence $-16\ -12\ -10\ -8\ -6\ -4\ -14\ -11\ -5\ -2\ -15\ -13\ -9\ -7\ -3\ -1$ has 3 increasing subsequences, $(-16\ -12\ -10\ -8\ -6\ -4\ -3\ -1)$, $(-14\ -11\ -5\ -2)$, and $(-15\ -13\ -9\ -7)$. But, Algorithm 7 gave 4 as the size of NIDS partitioning,

Algorithm 7 Minimum number of segments in NIDS: longest increasing or decreasing subsequence.

```
1: procedure LongestSubsequence($\pi = \pi_1 \cdots \pi_n$)
2:     $segments \leftarrow 0$;
3:     while $length(\pi) > 0$ do
4:         $\pi_I \leftarrow \pi$;
5:         $\pi_D \leftarrow \pi$;
6:         extend $\pi_I$ by adding $\pi_{I_0} \leftarrow 0$ and $\pi_{I_{n+1}} \leftarrow n+1$ in $\pi_I$;    ▷ Thus $\pi_I = \pi_{I_0}\pi_1\pi_2\cdots\pi_{n-1}\pi_n\pi_{I_{n+1}}$
   and $\pi_{I_0} = 0,\ \pi_{I_{n+1}} = n+1$.
7:         extend $\pi_D$ by adding $\pi_{D_0} \leftarrow n+1$ and $\pi_{D_{n+1}} \leftarrow 0$ in $\pi_D$;    ▷ Thus
   $\pi_D = \pi_{D_0}\pi_1\pi_2\cdots\pi_{n-1}\pi_n\pi_{D_{n+1}}$ and $\pi_{D_0} = n+1,\ \pi_{D_{n+1}} = 0$.
8:         Make a directed acyclic graph, $G_I = (V_I, E_I)$ from $\pi_I$ where,
9: $V_I = \{0, \pi_1, \pi_2, \ldots, \pi_{n-1}, \pi_n, n+1\}$ and an edge between $\pi_{I_i}$ and $\pi_{I_j}$ if $\pi_{I_i} < \pi_{I_j}$ and $i < j$, for all $i, j \in$
   $0, 1, 2, 3, \ldots, n-1, n, n+1$;
10:        Make a directed acyclic graph, $G_D = (V_D, E_D)$ from $\pi_D$ where,
11: $V_D = \{n+1, \pi_1, \pi_2, \ldots, \pi_{n-1}, \pi_n, 0\}$ and an edge between $\pi_{D_i}$ and $\pi_{D_j}$ if $\pi_{D_i} > \pi_{D_j}$ and $i <$
   $j$, for all $i, j \in 0, 1, 2, 3, \ldots, n-1, n, n+1$;
12:        Find the longest path in $G_I$ from $\pi_{I_0}$ to $\pi_{I_{n+1}}$ and add those vertices in increasing subsequence $s_I$;
13:        Find the longest path in $G_D$ from $\pi_{D_0}$ to $\pi_{D_{n+1}}$ and add those vertices in decreasing subsequence $s_D$;
14:        if $length(s_I) > length(s_D)$ then
15:            add the elements of $s_I$ except $\pi_{I_0}$ and $\pi_{I_{n+1}}$ in the subsequence $s$;
16:            $segments \leftarrow segments + 1$;
17:            delete the elements of $s_I$ from $\pi$;
18:            Clear $s,\ s_I$;
19:        else
20:            add the elements of $s_D$ except $\pi_{D_0}$ and $\pi_{D_{n+1}}$ in the subsequence $s$;
21:            $segments \leftarrow segments + 1$;
22:            delete the elements of $s_D$ from $\pi$;
23:            Clear $s,\ s_D$;
24:        end if
25:    end while
26:    return $segments$;
27: end procedure
```

Table 4.15: Subsequence statistics with Non-Contiguous Increasing and Decreasing System for renumbered unidirectional sequences with Algorithm 7.

Sequence patterns	**No. of Sequences**	**Avg. number of subsequences**	**Max. number of subsequences**	**Avg. length of subsequences**	**Max. length of subsequences**
Consecutive odd-even patterns	986	1.595	2	2.884	36
Consecutive even-odd patterns	985	1.595	2	2.891	38
Odd-even patterns	58	2.621	3	2.809	11
Even-odd patterns	60	2.617	4	3.197	27
More complex scrambled patterns	354	2.508	6	4.227	44

$(-16\ -12\ -10\ -8\ -6\ -4\ -2\ -1)$, $(-14\ -11\ -9\ -7\ -3)$, $(-15\ -13)$, and (-5). Another example of this scenario is: 4 8 10 2 5 7 9 1 3 6. This sequence can be partitioned into 3 subsequences, but Algorithm 7 gave 4 for the number of segments, which is not optimal. The subsequences determined by Algorithm 7 are (4 5 7 9), (8 2 1), (10 3), and (6). One of the optimal NIDS partitionings of this sequence with 3 segments would be (4 8 10), (2 5 7 9), and (1 3 6).

Hence, the above examples indicate that the greedy algorithm is not optimal. In the above examples, taking the longest subsequence first changes the remaining options for further subsequences, thus the final number of subsequences can increase. These examples prove that taking the longest subsequence is not always the optimal solution. In addition, the best results from the *NextIncreasingElement* algorithm emphasizes the idea of achieving optimality by adding the next increasing, or decreasing elements in a subsequence. Therefore, the following section discusses this final idea.

4.6.2 Considering Next Increasing or Decreasing Element in a Subsequence

The optimal partitioning for the examples above give the idea that adding the next increasing, or decreasing elements of the input permutation to an increasing or decreasing subsequence instead of looking for the longest increasing or decreasing subsequence might give a better size of a NIDS partitioning. In addition, the smallest average number of subsequences with the *NextIncreasingElement* algorithm, among the

Table 4.16: Subsequence statistics with Non-Contiguous Increasing and Decreasing System for renumbered extracted unidirectional sequences with Algorithm 7.

Sequence patterns	**No. of Sequences**	**Avg. number of subsequences**	**Max. number of subsequences**	**Avg. length of subsequences**	**Max. length of subsequences**
Consecutive odd-even patterns	280	1.536	2	3.026	20
Consecutive even-odd patterns	302	1.483	2	3.355	26
Odd-even patterns	13	2.615	3	3.588	11
Even-odd patterns	19	2.737	4	5.019	22
More complex scrambled patterns	135	2.704	7	5.468	92

implemented NIS algorithms, demonstrates this fact also. Therefore, the following algorithm *NextIncreasingOrDecreasingElement* (Algorithm 8) was implemented with the heuristic of adding the next increasing, or decreasing elements of π to the subsequences. At first, the algorithm adds the first element of π as the first element of increasing subsequence s_I and decreasing subsequence s_D. Then it finds the next larger element in π, adds it to increasing subsequence s_I, and continues doing this until reaching the end of π. Similarly, it also finds the next smaller element in π, adds it to the decreasing subsequence s_D, and continues doing this until reaching the end of π. Now, the largest subsequence between s_I and s_D is the first increasing or decreasing subsequence s of π respectively. Next, the algorithm deletes the elements of this increasing or decreasing subsequence s from π, and repeatedly finds the next increasing or decreasing subsequence s in a similar fashion until π becomes empty. In the end, the final number of subsequences is the size of the NIDS partitioning of the input permutation determined by Algorithm 8.

Tables 4.17 and 4.18 show the average number of subsequences determined by Algorithm 8, along with the maximum number of subsequences, number of sequences in each sequence pattern, average length of subsequences, and the maximum length of subsequences for renumbered unidirectional and renumbered extracted unidirectional scrambled sequences. In comparison with Algorithm 7, Algorithm 8 gives the same average number of subsequences for the consecutive even/odd sequences of both of the datasets. Interestingly, Algorithm 8 does not reduce the average number of subsequences for any type of sequences, instead it increases

Algorithm 8 Minimum number of segments with NIDS: subsequence with next increasing or decreasing elements.

```
procedure NextIncreasingOrDecreasingElement($\pi = \pi_1 \cdots \pi_n$)
    $segments \leftarrow 0$;
    while $\pi$ is not empty do
        $element \leftarrow \pi_1$;
        add element in an increasing subsequence $s_I$;
        add element in a decreasing subsequence $s_D$;
        for $i \leftarrow 2$ to $n$ do
            if $element < \pi_i$ then
                $element \leftarrow \pi_i$;
                add element in increasing subsequence $s_I$;
            end if
        end for
        for $i \leftarrow 2$ to $n$ do
            if $element > \pi_i$ then
                $element \leftarrow \pi_i$;
                add element in decreasing subsequence $s_D$;
            end if
        end for
        if $length(s_I) > length(s_D)$ then
            subsequence $s \leftarrow s_I$;
            $segments \leftarrow segments + 1$;
        else
            subsequence $s \leftarrow s_D$;
            $segments \leftarrow segments + 1$;
        end if
        delete the elements of $s$ from $\pi$;
        Clear $s,\ s_I,\ s_D$;
    end while
    return $segments$;
end procedure
```

Table 4.17: Subsequence statistics with Non-Contiguous Increasing and Decreasing System for renumbered unidirectional sequences with Algorithm 8.

Sequence patterns	**No. of Sequences**	**Avg. number of subsequences**	**Max. number of subsequences**	**Avg. length of subsequences**	**Max. length of subsequences**
Consecutive odd-even patterns	986	1.595	2	2.884	36
Consecutive even-odd patterns	985	1.595	2	2.891	38
Odd-even patterns	58	2.65	3	2.685	11
Even-odd patterns	60	2.741	4	3.157	26
More complex scrambled patterns	354	2.522	5	4.203	43

the average number of subsequences for the more complex scrambled sequences of renumbered extracted unidirectional sequences, 2.740 from 2.704. In addition, for the renumbered unidirectional sequences, the average number of subsequences goes to 2.65 from 2.621 for odd-even sequences, 2.741 from 2.617 for even-odd sequences, and 2.522 from 2.508 for more complex scrambled sequences.

Hence, the increase in the average number of subsequences with Algorithm 8 over Algorithm 7 signifies that there must be some sequences where Algorithm 8 could not give best results. There is such a sequence of more complex scrambled pattern that can be partitioned into 2 subsequences, whereas Algorithm 8 partitioned it into 3 subsequences as the smallest NIDS partitioning, which is not optimal. The sequence is 6 1 5 3 2 4. This example sequence can be partitioned into 2 subsequences (6 5 3 2) and (1, 4). But, Algorithm 8 gave 3 as the smallest size of NIDS partitioning, (6 1), (5 3 2), and (4). Another example of this scenario is the following sequence: 6 2 4 9 8 1 3 5 7. Algorithm 8 partitioned this sequence into 4 subsequences (6 2 1), (4 8 3), (9 5), and (7), whereas it can be partitioned into 3 subsequences only, (2 4 5 7), (9 8 1), and (6, 3).

Thus, among the implemented NIDS algorithms, Algorithm 7 gives better results. However, both of the implemented algorithms are not optimal. The examples where taking the longest subsequences give better results, Algorithm 8 fails, and similarly the examples where taking the longest subsequences increase the final size of NIDS partitioning, Algorithm 7 fails. A comparison between the results of Algorithm 7 and 8 shows that both of the algorithms give the same size of NIDS partitioning for the all types of sequences,

Table 4.18: Subsequences statistic with Non-Contiguous Increasing and Decreasing System for renumbered extracted unidirectional sequences with Algorithm 8.

Sequence patterns	**No. of Sequences**	**Avg. number of subsequences**	**Max. number of subsequences**	**Avg. length of subsequences**	**Max. length of subsequences**
Consecutive odd-even patterns	280	1.536	2	3.026	20
Consecutive even-odd patterns	302	1.483	2	3.355	26
Odd-even patterns	13	2.615	3	3.588	11
Even-odd patterns	19	2.737	4	5.019	22
More complex scrambled patterns	135	2.740	7	5.394	87

except for the more complex scrambled sequences for the renumbered extracted unidirectional dataset. From the renumbered extracted unidirectional dataset, there were 14 more complex scrambled sequences where Algorithm 7 gives better results, and 6 more complex scrambled sequences where Algorithm 8 gives better results. All of these 14 sequences were partitioned by 1 less segment by Algorithm 7 in comparison with Algorithm 8, 3 sequences were partitioned by 1 less segment, and 3 sequences were partitioned by 2 less segments by Algorithm 8 in comparison with Algorithm 7. Similarly, from the renumbered unidirectional dataset, 2 even-odd sequences, 7 odd-even sequences, and 20 more complex scrambled sequences were partitioned by 1 less segment by Algorithm 7 in comparison with Algorithm 8. Further, 13 and 1 more complex scrambled sequences were partitioned by 1 and 2 less segments, respectively, by Algorithm 8 in comparison with Algorithm 7. In summary, there were 43 sequences among the 3192 sequences where Algorithm 7 gave a better result than Algorithm 8, which is 1.35% of the sequences. In contrast, there were 20 sequences where Algorithm 7 failed to give a better result than Algorithm 8. Therefore, in comparison with the *NextIncreasingOrDecreasingElement* algorithm, the *LongestSubsequence* algorithm failed to give better results for 0.62% of the sequences, and it gave the best results among the implemented NIDS partitioning algorithms.

The graphs in Figures 4.13 and 4.14 show the visualisations of the number of segments (size of NIDS partitioning with Algorithm 7) versus the number of sequences achieving that number of segments for renumbered unidirectional and renumbered extracted unidirectional sequences, respectively. The number of segments

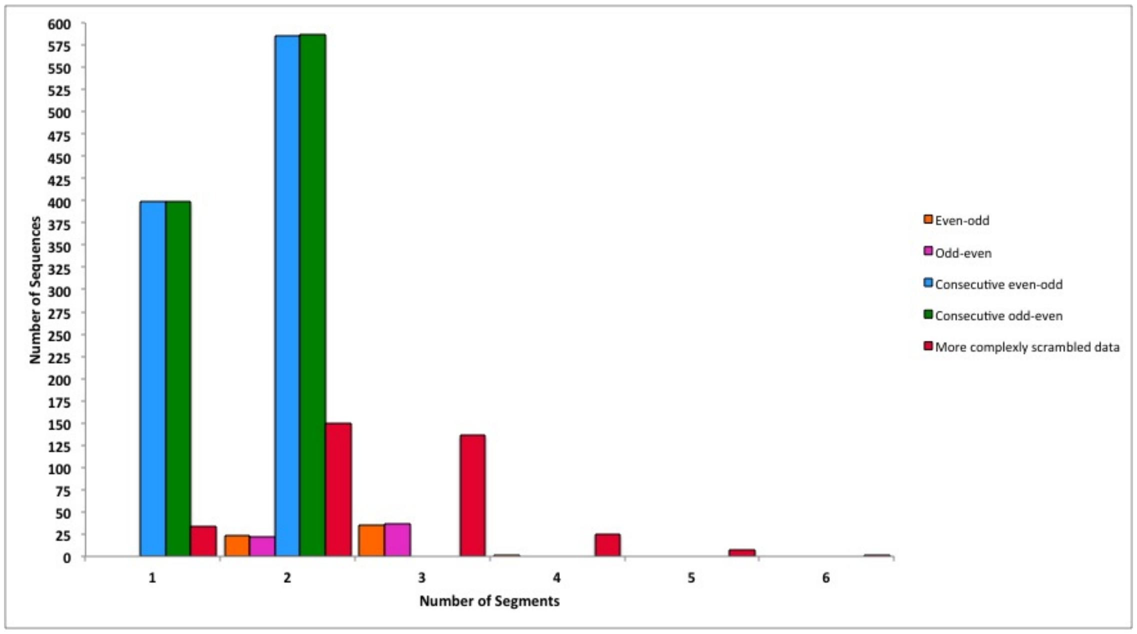


Figure 4.13: Relationship between the number of segments (increasing and decreasing subsequences) and the number of sequences in the dataset achieving that for renumbered unidirectional sequences as determined by Algorithm 7.

was determined with the *"LongestSubsequence"* algorithm, as it gives the best result among the implemented NIDS algorithms. The graphs show that similar to CIDS, this system also has the minimum number of subsequences as 1 for both of the datasets, and the most of the sequences were partitioned by less than 5 subsequences (Figures 4.13 and 4.14) by Algorithm 7 for NIDS. Thus, the good performance of Algorithm 6 (Figures 4.10 and 4.11), and 7 computationally inferred the benefit of the non-contiguous systems.

4.7 Result Analysis

The patterns of the order of MDSs in the micronuclear chromosome fragments show evidence of some biological operations that can be computationally described with shuffle operations. Therefore, we studied four similar systems involving shuffle to see how the minimum number of applications of shuffle differs between the types. However, the question remains — which system is most likely? This question can be addressed from both biological and computational perspectives. Though the previous sections discuss the significance of each type of system on the minimum number of applications of shuffle based on the computational results, the following section discusses conclusions.

4.7.1 Comparison of the Systems

As we do not know the exact mechanism by which descrambling takes place, each system represents a different computational descrambling scenario involving the minimum number of applications of shuffle.

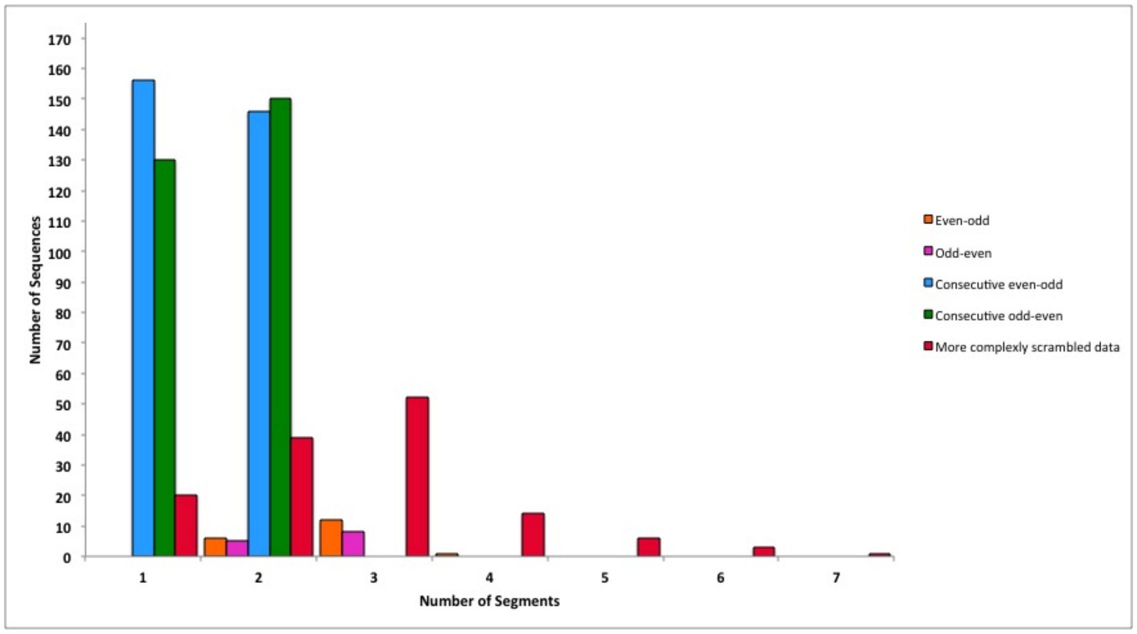


Figure 4.14: Relationship between the number of segments (increasing and decreasing subsequences) and the number of sequences in the dataset achieving that for renumbered extracted unidirectional sequences as determined by Algorithm 7.

Hence, different types of systems will result in different types of segments, and different types of segments will result in different minimum numbers of moves. Thus, a comparison among the computational results of these four systems might act as a starting point to draw a conclusion about the most-likely system. Table 4.19 and 4.20 show the average number of segments determined by each system, along with the maximum number of segments for the renumbered unidirectional and renumbered extracted unidirectional scrambled sequences, respectively. Algorithms 6 and 7 have been considered for NIS and NIDS respectively, as they gave the best minimum number of segments among the implemented algorithms for NIS and NIDS for the input datasets. The results show that decreasing segments reduce the average number of segments, but that does not imply that it occurs biologically.

During gene assembly, the micronuclear chromosomes become organised in coiled, lampbrush patterns or loop-like structures that might be a necessary prerequisite for later IES elimination and MDS rearrangement [28, 29]. Moreover, the functional macronuclear molecules may be produced through multiple descrambling pathways, where each pathway can descramble a number of MDSs through multiple parallel inversion and transposition events [30]. However, a decreasing pattern such as 4 3 2 1 would require that MDS 2 move to the right of MDS 1, then MDS 3 move to the right of MDS 2, then MDS 4 to the right of MDS 3. Each change seems to require a separate structure, and therefore it is likely not realistic to allow for such moves to occur "for free" as they do for CIDS and NIDS. Furthermore, these systems do not remarkably reduce the number of moves required, especially after eliminating simple sequences such as 2 1. Therefore, in spite of giving fewer number of applications of shuffle, these two systems are likely not realistic.

Table 4.19: Comparison of the average number of segments and the maximum number of segments for different types of systems for the renumbered unidirectional sequences.

	Contiguous Increasing System		Contiguous Increasing and Decreasing System		Non-Contiguous Increasing System with *NextIncreasingElement* algorithm		Non-Contiguous Increasing and Decreasing System with *LongestSubsequence* algorithm	
Sequence Patterns	Avg. no. of segments	Max. no. of segments	Avg. no. of segments	Max. no. of segments	Avg. no. of segments	Max. no. of segments	Avg. no. of segments	Max. no. of segments
Consecutive odd-even patterns	2	2	1.595	2	2	2	1.595	2
Consecutive even-odd patterns	2	2	1.594	2	2	2	1.595	2
Odd-even patterns	3.086	4	2.775	4	2.984	3	2.621	3
Even-odd patterns	3.067	4	2.766	4	2.867	4	2.617	4
More complex scrambled patterns	3.799	20	3.37	19	2.918	6	2.508	6

Among CIS and NIS, Algorithm 6 for non-contiguous system gives a smaller number of applications of shuffle. Even though the results for CIS are optimal, and the results of Algorithm 6 for NIS may not be optimal, the number of segments with Algorithm 6 for NIS is considerably less than the number of segments in CIS. In particular, the maximum number of segments in CIS is 20 and 35 for the renumbered unidirectional sequences and renumbered extracted unidirectional sequences respectively, whereas with Algorithm 6 for NIS, this number is 6 and 7, respectively. This is sufficiently smaller. Moreover, each pathway can descramble a section of MDSs that might or might not reside beside each other, as the chromosomes fold mostly in coiled, lampbrush, or loop-like structures, thus alignment of the non-contiguous MDSs subsequently by the structure might not be impractical. Hence, the non-contiguous increasing system would have advantages as they consider both contiguous and non-contiguous MDSs (i.e. the elements of a subsequence can be

Table 4.20: Comparison of the average number of segments and the maximum number of segments for different types of systems for the renumbered extracted unidirectional sequences.

	Contiguous Increasing System		Contiguous Increasing and Decreasing System		Non-Contiguous Increasing System with *NextIncreasingElement* algorithm		Non-Contiguous Increasing and Decreasing System with *LongestSubsequence* algorithm	
Sequence Patterns	Avg. no. of segments	Max. no. of segments	Avg. no. of segments	Max. no. of segments	Avg. no. of segments	Max. no. of segments	Avg. no. of segments	Max. no. of segments
Consecutive odd-even patterns	2	2	1.535	2	2	2	1.536	2
Consecutive even-odd patterns	2	2	1.483	2	2	2	1.483	2
Odd-even patterns	3.077	4	2.846	3	2.846	3	2.615	3
Even-odd patterns	3.105	4	3	4	3.105	4	2.737	4
More complex scrambled patterns	4.37	35	3.83	35	3.274	7	2.704	7

contiguous, or non-contiguous in an input permutation). In addition, NIS gives the minimum number of segments, which results in the minimum number of moves for descrambling. Therefore, the non-contiguous increasing system has advantages. However, the feasibility of any such hypothesis derived from parsimony needs to be validated experimentally.

4.7.2 Optimality Analysis

The comparison among the results of the *NextIncreasingElement* algorithm of NIS (Algorithm 6), and the other implemented algorithms for NIS show that Algorithm 6 has successfully given the best results out of all the algorithms considered for all of the input sequences. Moreover, another comparison between the results of Algorithm 6 and the optimal results of CIS did not give any example sequence that Algorithm 6 failed to give a better result than the optimal CIS algorithm (the elements of a subsequence can be contiguous, or non-contiguous in an input permutation, thus the segments for NIS can be a subword too). However, it is an open question as to whether Algorithm 6 is optimal or not. As no example input sequences were found where Algorithm 6 failed to give a better result, this section discusses the optimality of Algorithm 6 for different types of input sequences.

Consecutive odd-even and consecutive even-odd sequences:

There are 986 consecutive odd-even and 985 consecutive even-odd sequences in the renumbered unidirectional dataset, and 280 consecutive odd-even and 302 consecutive even-odd sequences in the renumbered extracted unidirectional dataset. Thus, there are a total of 1266 consecutive odd-even, and 1287 consecutive even-odd sequences among the 3192 scrambled input sequences. Algorithm 6 partitioned all of these 2553 consecutive odd/even sequences into 2 increasing segments, which is optimal. This is because in NIS partitioning, only the unscrambled sequences will have 1 segment. Therefore, Algorithm 6 gives the optimal size of NIS partitioning for all of these 2553 consecutive odd/even sequences.

Odd-even and even-odd sequences:

There are 58 odd-even and 60 even-odd sequences in the renumbered unidirectional dataset, and 13 odd-even and 19 consecutive even-odd sequences in the renumbered extracted unidirectional dataset. Thus, there are a total of 71 odd-even, and 79 even-odd sequences among the 3192 scrambled input sequences. Note that these odd-even and even-odd sequences are not in the consecutive odd-even and consecutive even-odd patterns. Table 4.21 represents the frequency of the size of an NIS partitioning with Algorithm 6, and the number of sequences achieving that for these 71 odd-even, and 79 even-odd scrambled input sequences.

Table 4.21: Size of NIS partitioning with Algorithm 6, and the number of sequences achieving that for the odd-even, and even-odd scrambled input sequences.

Input datasets	**Size of NIS partitioning with Algorithm 6**	**No. of odd-even sequences**	**No. of even-odd sequences**
Renumbered unidirectional sequences (71)	2	3	9
	3	55	50
	4	0	1
Renumbered extracted unidirectional sequences (79)	2	2	0
	3	11	17
	4	0	2

Thus, among the 150 odd/even scrambled input sequences, Algorithm 6 partitioned 14 sequences optimally, as the size of NIS partitioning for these sequences is 2. The Algorithm 6 partitioned 133 odd/even pattern sequences into 3 segments (Table 4.21). Hence, for these 133 odd/even sequences we do not know the optimal answers. However, we know that, if $s_{optimal}(\pi)$ is the optimal size of NIS partitioning, and

$s(\pi)$ is the size of NIS partitioning determined by Algorithm 6, then for these 133 odd/even sequences, $s(\pi) \leq s_{optimal}(\pi) + 1$.

More complex scrambled sequences

There are 354 categorized as more complex scrambled sequences in the renumbered unidirectional dataset, and 135 more complex scrambled sequences in the renumbered extracted unidirectional dataset. Thus, there are a total of 489 more complex scrambled sequences among the 3192 scrambled input sequences. Table 4.22 represents the frequency of the size of NIS partitioning with Algorithm 6, and the number of sequences achieving that for these 489 more complex scrambled input sequences.

Table 4.22: Size of NIS partitioning with Algorithm 6, and the number of sequences achieving that for the more complex scrambled input sequences.

Input datasets	**Size of NIS partitioning with Algorithm 6**	**No. of more complex scrambled sequences**
Renumbered unidirectional sequences (354)	2	99
	3	196
	4	49
	5	9
	6	1
Renumbered extracted unidirectional sequences (135)	2	23
	3	76
	4	23
	5	5
	6	5
	7	3

Similarly, among the total 489 more complex scrambled input sequences, Algorithm 6 partitioned 122 sequences optimally, as the size of NIS partitioning for these sequences is 2 (Table 4.22). Algorithm 6 partitioned total 272 more complex scrambled sequences into 3 segments (Table 4.22). Hence, for these 272 more complex scrambled sequences we do not know the optimal answers, but they are at most one more than the optimal answer.

Therefore, the optimality analysis of the *NextIncreasingElement* algorithm on the input sequences can be summarized as following:

- Of the 3192 scrambled input sequences, 2553 (consecutive odd/even sequences), 14 (odd/even sequences), 122 (more complex scrambled sequences), totalling 2689 sequences were partitioned optimally by Algorithm 6, as the size of NIS partitioning for these sequences by Algorithm 6 is 2.

- Of the 3192 scrambled input sequences, 133 (odd/even sequences), 272 (more complex scrambled sequences), totaling 405 sequences were partitioned into 3 segments by Algorithm 6. Thus, for these 405 sequences the size of NIS partitioning by Algorithm 6 $s(\pi)$ is, $s(\pi) \leq s_{optimal}(\pi) + 1$.

- Hence, for the remaining 98 scrambled input sequences, the size of NIS partitioning by Algorithm 6 $s(\pi)$ is, $s(\pi) \leq s_{optimal}(\pi) + 5$. This is because the maximum number of segments with Algorithm 6 for NIS is 7, and any scrambled sequence cannot be partitioned into less than 2 increasing subsequences.

Note that we have 2443 unscrambled sequences in the renumbered extracted unidirectional dataset, and 18,315 unscrambled MIC MDS sequences. As these sequences are unscrambled, they were excluded from the input sequences of the implemented algorithms, and result analysis.

In summary, Algorithm 6 for NIS partitioned at least 84.24% of the scrambled input sequences optimally. Also, the size of the NIS partitioning $s(\pi)$, for 12.68% of the input sequences satisfies, $s(\pi) \leq s_{optimal}(\pi) + 1$. Thus, of the 3192 scrambled input sequences, the size of NIS partitioning by Algorithm 6 $s(\pi)$, for 96.93% sequences is, $s(\pi) \leq s_{optimal}(\pi) + 1$.

4.8 Discussion

The descrambling order determination study focused on determining the required minimum number of moves to descramble the section of a MIC chromosome corresponding to one MAC chromosome. Thus, the four similar systems of moves were studied involving shuffle to see how the minimum number of operations differs between the types. Moreover, determining the minimum number of applications of shuffle to descramble a permutation involved determining the minimum number of segments on which the shuffle is applied. Hence, the four systems allow different types of segments, and algorithms were implemented to determine the minimum number of segments for each system. As described above, among the considered systems the non-contiguous increasing system gives the best results, and the descrambling order from this system showed that almost all of the input sequences can be descrambled by very few moves.

Recall that the bidirectional sequences were separated into its two sub-subsequences: one holding the order of MDSs that are in the non-inverted order, and the other holding the order of MDSs that are inverted. The non-contiguous systems (NIS and NIDS) determines the minimum number of applications of shuffle to descramble its two subsequences, separately. If one instead wanted to combine the two subsequences, then it requires one extra application of shuffle, because the identity permutation of the bidirectional sequences is in the shuffle of its two descrambled subsequences. However, this is not the case for the contiguous systems (CIS and CIDS). For example, this sequence $1\ -6\ 2\ -5\ 3\ -4$ has 5 cut-off points, because an individual segment cannot contain both positive and negative numbers. So the correct answer for CIS is 6, and this is optimal. But if positive and negative parts are separated out, then it is $1\ 2\ 3$ and $-6\ -5\ -4$, each having no cut-off points. Thus, for the contiguous systems we have only solved the non-inverted parts and inverted parts separately, and their optimal combination is left as future work.

The small minimum number of segments regained for 96.63% of the scrambled input sequences indicates that each micronuclear chromosome fragment can be transformed into the corresponding macronuclear chromosome fragment by very few moves, where each move represents one or multiple parallel descrambling operations. Thus, if the structure is providing the information about how the MDSs are aligned, then it really requires very few parallel operations in descrambling. This is related to the idea of parsimony, whereby the smallest number of steps is likely close to the actual number of operations that occurred during the descrambling of the gene assembly process. For example, the number of applications of shuffle needed for this system is far lower than the number of MDSs, and therefore the principle of parsimony dictates that there is significant computational advantages to such a system in terms of simplicity to descramble. The results also indicate that the patterns of the scrambled MDSs in the micronuclear chromosome fragments is providing some information which makes the MDS rearrangement during gene assembly much simpler than it seems. In other words, the results show that the complications of MDS descrambling in ciliates might be overestimated, and the complex arrangement of the large number of MDSs of miconuclear chromosome fragments do not need a large number of steps to descramble.

The results analysis shows that 96.63% of the scrambled micronuclear chromosome fragments of *Oxytricha trifallax* can be partitioned into 2 to 3 segments, and therefore can be descrambled by only 1 or 2 applications of shuffle. This smallest number of steps lends theoretical evidence that some structural component is enforcing the shuffle-like behaviour, by properly aligning segments in an interleaving fashion, and then parallel recombination is taking place for MDS rearrangement and IES elimination. Indeed, the sheer number of micronuclear chromosome fragments that can be rearranged with very few applications of shuffle yields evidence that this type of behaviour is occurring.

Chapter 5

MDS and IES Identification with HMMs

The second objective of the thesis is to classify each nucleotide of every micronuclear chromosome segment as an MDS or IES. To do this, the well-known supervised machine learning classification model, hidden Markov models (HMMs) were used. Supervised machine learning techniques involve the use of training data, where the correct classification is known (i.e. sequences where the MDSs and IESs are already known). There were 21,157 labelled sequences of micronuclear contig subwords as the input sequences for the classifying MDSs and IESs by HMMs problem. For this work, the correct answers were assured to be the results of the preprocessing of the input data described in Section 3.2. Now, the following sections briefly describe the implemented HMMs, their accuracy, performance, and effectiveness. This work represents only a preliminary investigation into this topic.

5.1 Methodology

The sequences of MDSs and IESs have distinct statistical properties. Analysis of different k-mers from Section 3.2 showed that the previous nucleotides contribute knowledge in predicting the next most probable nucleotide. For this application, the sequence of nucleotides is known, and the objective is to predict whether each nucleotide is part of an MDS or IES. The observed sequences and its corresponding states (MDS or IES) contribute in predicting the state of the next nucleotides. Moreover, the distribution of state transitions can be inferred from the data, and the emissions are only visible here, but the states are hidden, and the aim is to predict the classes from the observed data set. Hence, the problem is quite similar with those problems that are solved by hidden Markov models. In Section 2.4, the significance of HMMs in classifying biological sequences was discussed, together with a discussion of similar biological problems such as classifying a sequence segment into coding and non-coding regions [5]. Considering the nature of the data, HMMs, are a good algorithm to try for classifying MIC chromosome fragments into MDS and IES. The use of other machine learning techniques for this problem is beyond the scope of this thesis.

For determining the class, often the frequency of each nucleotide in each class is counted. Hence, the probability of each emission from each class can be calculated through these frequencies by the Bayesian probabilistic theorems. In the same way, the frequencies and the probabilities of transitions from each class to every other class (i.e. from class IES to class MDS, from class MDS to class MDS etc.) is calculated.

These frequencies correspond to a probabilistic HMM of order one. This assumes that the sequences have been produced by a process that chooses any of the four nucleotides of the sequence, where the probability of choosing any of the four nucleotides at a particular position depends on the nucleotide chosen for the previous position.

The accuracy of the classification is verified through 10-fold cross validation. 10-fold cross validation partitions data into 10 randomly chosen subsets (or folds) of roughly equal size (off by at most one if the size is not divisible by 10). Then, one subset is used to validate the model, and the remaining subsets are used to train the model. This process is repeated 10 times so that each subset is used exactly once for validation. 10-fold cross validation is used, as it has become a standard method [26]. Moreover, extensive tests on numerous datasets with different learning techniques, have shown that 10 is approximately the right number of folds to get the best estimate of error, and there is also some theoretical evidence to support this choice [45].

Specifically, the micronuclear chromosomes fragments were first classified by a first-order supervised HMM. As HMMs predict the states in two steps — training and classifying — thus the labelled input data was divided into 10 randomly chosen subsets of approximately equal size to make the training and testing data. Of the 10 subsets, 9 subsets randomly chosen were used as the training data, and the last one was the testing data. The model calculated the initial, transition, and emission matrices by applying Bayesian probabilistic theorems from the observed sequences of emissions and states of the training data. Hence, the initial matrix holds the probability of the first nucleotide of a sequence being an MDS or IES, the transition matrix contained the probabilities of class transitions, and the emission matrix had the probabilities of each base occurring in each class. For the first-order HMM, the emission bases were A, C, G, and T. Then, the model classified the testing sequences according to the probabilities of the initial, transition, and emission matrices with Viterbi's Algorithm. Viterbi's Algorithm is a very well known machine learning algorithm that determines the most likely sequence of states that can produce the sequence, where a sequence of observations and a model is given [31]. This procedure was repeated 10 times to ensure that each subset had been used exactly once for testing. The accuracy of the model was measured by comparing the classified sequences with the labelled sequences (results have been described in the next section). For implementing and analyzing the HMMs, the statistical and machine learning toolbox of MATLAB software were used.

Data analysis with different k-mers showed that higher values of k for k-mers has an impact on the probability distributions. Hence, it is quite obvious to look at the higher order (order n) HMMs by looking at the statistics of the corresponding k-mers. Thus, to observe the effect of prior knowledge regarding the previous neighbouring nucleotides, and improve efficiency, higher order HMMs were implemented. Higher order HMMs were implemented by implementing first-order HMMs with k-mers emissions, where $k = 2, 3, 4, 5$. Hence, for the second-order HMM, the number of emissions was 16 (all pairs of nucleotides), for the third-order HMM the number of emissions was 64 (all 3-tuples), for the fourth-order HMM the number of emissions was 256 (all 4-tuples), and for the fifth-order HMM this number was 1024 (all 5-tuples). However, the states were constant for all of the implemented HMMs, two states were used, MDS and IES. Finally, the effectiveness of the model was evaluated by measuring the percentage of correctly classified MDSs.

5.2 Results

For each order HMM there were 10 sets of results, because the accuracy of the HMMs were verified by 10-fold cross validation. For each set, the precision, specificity, accuracy, and sensitivity of the classification, were calculated. The sensitivity and specificity are statistical measures of the performance of a binary classification test (binary classification is the task of classifying the elements of a given set into two groups on the basis of a classification rule). Sensitivity (also known as recall) measures the proportion of true positives relative to all positives, which we associated with the proportion of MDSs that are correctly identified as MDSs, and specificity measures the proportion of true negatives relative to all negatives, which we associated with the proportion of IESs that are correctly identified as IESs. The accuracy is the proportion of true results (both true MDSs and true IESs) among the total number of cases examined. Table 5.1 shows that a first-order HMM could classify the micronuclear chromosome fragments with an average of 61.2% accuracy.

Table 5.1: Result analysis for first-order HMM (t_p = true positive, t_n = true negative, f_p = false positive, and f_n = false negative).

	Precision $(\frac{t_p}{t_p+f_p})$	**Recall or Sensitivity** $(\frac{t_p}{t_p+f_n})$	**Specificity** $(\frac{t_n}{t_n+f_p})$	**Accuracy** $(\frac{t_p+t_n}{t_p+t_n+f_p+f_n})$
Results for each of the 10 folds	0.7253	0.4917	0.5977	0.6121
	0.7224	0.491	0.5983	0.6131
	0.7331	0.4947	0.5999	0.6137
	0.7314	0.4945	0.6075	0.6095
	0.7281	0.4915	0.599	0.6108
	0.7263	0.488	0.6034	0.6122
	0.7076	0.4921	0.6036	0.6096
	0.7437	0.4981	0.5943	0.616
	0.7258	0.4788	0.6103	0.608
	0.7442	0.4843	0.6084	0.6144
Average	0.72879	0.49047	0.60224	0.61194

In a classification task, the precision for a class is the number of true positives (i.e. the number of items correctly labeled as belonging to the positive class) divided by the total number of elements labeled as belonging to the positive class (i.e. the sum of true positives and false positives, which are items incorrectly labeled as belonging to the class). Thus, precision is the measure of correctly classified true MDSs. Table 5.1 shows that the average precision for a first-order HMM is 72.87%, and the average sensitivity of this classification is 49.04%.

Table 5.2: Result analysis for second-order HMM (t_p = true positive, t_n = true negative, f_p = false positive, and f_n = false negative).

	Precision $(\frac{t_p}{t_p+f_p})$	**Recall or Sensitivity** $(\frac{t_p}{t_p+f_n})$	**Specificity** $(\frac{t_n}{t_n+f_p})$	**Accuracy** $(\frac{t_p+t_n}{t_p+t_n+f_p+f_n})$
Results for each of the 10 folds	0.7356	0.4977	0.6241	0.6201
	0.7286	0.4921	0.6352	0.6235
	0.7233	0.4981	0.6314	0.6229
	0.7354	0.5115	0.6147	0.6294
	0.7352	0.4959	0.6397	0.6277
	0.7213	0.4972	0.6163	0.6222
	0.7291	0.5003	0.6209	0.6243
	0.7312	0.4968	0.6234	0.6243
	0.7347	0.4915	0.6307	0.6243
	0.7287	0.4898	0.6417	0.6231
Average	0.73031	0.49709	0.62781	0.62418

To improve the precision and accuracy, second-, third-, fourth-, and fifth-order HMMs were implemented, and Tables 5.2, 5.3, 5.4, and 5.5 give the result of the implemented higher order HMMs respectively. The average accuracy, precision, sensitivity, and specificity, with second-order HMM is 62.41%, 73.03%, 49.71%, and 62.78% respectively. Thus, the results of the second-order HMM shows that the knowledge of previous bases has a role in increasing precision and accuracy. Though none of the first-, or second-order HMM could classify the MDSs with good precision. The average accuracy, precision, sensitivity, and specificity with the third-order HMM is 62.68%, 73.03%, 49.61%, and 64.47% respectively. Thus, the results (in terms of accuracy, precision, and sensitivity) of the second-, and third-order HMM are similar — the precision remains the same, the accuracy, and the sensitivity decrease slightly with the third-order HMM. But, the third-order HMM has increased the average specificity notably, 64.47% from 62.78%.

Tables 5.4 and 5.5 show that the knowledge of the previous four, and five neighbouring bases still did not give particularly good results. Though, the accuracy increased (63.10%, and 66.56% respectively), as did the sensitivity (50.23%, and 59.2% respectively). However, the precision — 72.75% with fourth-order HMM, and 71.45% with fifth-order HMM — decreased in comparison to the second-, and third-order HMMs (Tables 5.2, and 5.3). Moreover, the specificity remains almost the same with the fourth-order HMM (64.79%), but decreased with the fifth-order HMM (58.57%). Thus, none of the implemented HMM could achieve the desired level of precision, and could not classify any of the data folds with above 75% precision. Moreover among all the implemented HMMs, the fifth-order HMM achieved the highest sensitivity of 59.2%, but the precision did not go above 72.1%. While this sensitivity value was much higher with the fifth-order HMM compared to lower orders, a sensitivity of 59.2% demonstrates that only 59.2% of the MDSs were classified as MDS. Also, the precision indicates that only 71.45% of classified MDSs were true MDS. Therefore, the

Table 5.3: Result analysis for third-order HMM (t_p = true positive, t_n = true negative, f_p = false positive, and f_n = false negative).

	Precision ($\frac{t_p}{t_p+f_p}$)	**Recall or Sensitivity** ($\frac{t_p}{t_p+f_n}$)	**Specificity** ($\frac{t_n}{t_n+f_p}$)	**Accuracy** ($\frac{t_p+t_n}{t_p+t_n+f_p+f_n}$)
Results for each of the 10 folds	0.7288	0.498	0.6373	0.6253
	0.7232	0.5022	0.6379	0.6314
	0.7321	0.4945	0.6437	0.623
	0.7272	0.4957	0.6478	0.6216
	0.7306	0.4834	0.6567	0.6277
	0.7327	0.4993	0.6416	0.6253
	0.7327	0.4993	0.6416	0.6253
	0.7404	0.4994	0.6451	0.6267
	0.7304	0.4983	0.6445	0.6243
	0.7226	0.4966	0.6458	0.6348
Average	0.73007	0.49667	0.6442	0.62654

classification was not a particularly good one.

5.3 Discussion

The results showed that none of the implemented HMMs could successfully classify a good percentage of MDSs and IESs. Of the implement models, the second- and third-order HMMs could determine the highest number of true MDSs, at 73.03%. Thus, the knowledge of the previous four, and five bases could not add enough information for the correct determination of more MDSs. Moreover, the accuracy of the classifications is very poor (Tables 5.1, 5.2, 5.3, 5.4, and 5.5). However, the fifth-order HMM has the highest accuracy among the implemented models, but that is only 66.56%. Note that the fifth-order HMM has increased the accuracy, but decreased the precision (71.45%) versus the second-, and third-order HMMs. Thus, just to check whether or not this characteristic remains the same for higher order HMMs, a sixth-order HMM was implemented. Table 5.6 shows the precision, specificity, sensitivity, and accuracy of the classification done by the implemented sixth-order HMM.

The average precision, accuracy and specificity of classification with sixth-order HMM is 59.95%, 60.65%, and 2.05% respectively, thus the specificity has decreased incredibly. Note that the sixth-order HMM increased the sensitivity, 99.44%. Thus, the increased sensitivity, and decreased specificity denotes that the sixth-order HMM is actually predicting almost all of the nucleotides as MDS. Thus, the classification with lower order HMMs was far better than the classification with the sixth-order HMM, but they also could not give a particularly good classification. Hence, the results (Tables 5.1, 5.2, 5.3, 5.4, 5.5, and 5.6) showed that an attempt of classifying the micronuclear genome with an HMM is likely not a good idea, as the successful

Table 5.4: Result analysis for fourth-order HMM (t_p = true positive, t_n = true negative, f_p = false positive, and f_n = false negative).

	Precision $(\frac{t_p}{t_p+f_p})$	**Recall or Sensitivity** $(\frac{t_p}{t_p+f_n})$	**Specificity** $(\frac{t_n}{t_n+f_p})$	**Accuracy** $(\frac{t_p+t_n}{t_p+t_n+f_p+f_n})$
Results for each of the 10 folds	0.7215	0.5185	0.6361	0.6395
	0.7291	0.5174	0.6318	0.6278
	0.7314	0.5055	0.6437	0.6378
	0.7229	0.5009	0.658	0.6278
	0.7306	0.4913	0.6509	0.6267
	0.7255	0.4904	0.6519	0.6317
	0.7374	0.4971	0.6587	0.6305
	0.7198	0.4924	0.647	0.6266
	0.7367	0.5097	0.6501	0.6361
	0.7204	0.4993	0.6512	0.6259
Average	0.72753	0.50225	0.64794	0.63104

biological sequences classification with HMMs has some additional knowledge that helps the almost accurate classification. For example, the knowledge regarding the start codons, and stop codons (described in Section 2.4) gives some additional information for predicting genes. For the exon and intron determination, information regarding spliceosomal introns helps classifiction [5, 39]. Spliceosomal introns are characterized by specific intron sequences located at the boundaries between introns and exons [16]. Similarly, CpG islands have a high probability of CG dinucleotides, in comparison with other regions of a biological sequence [18]. Thus, to get an effective classification of biological sequences with an HMM, it is important to incorporate some additional knowledge to the training parameters.

Therefore, it is more likely that an HMM will fail to accurately classify micronuclear chromosomes without having some predefined knowledge about the nucleotide patterns of MDSs or IESs. Though there might be other machine learning techniques that might work better than HMMs, and adding the pointer state might improve the results to some extent, but for identifying MDSs, it is unlikely that any would be good enough. The process of rearranging the MDSs and eliminating the IESs has to be robust, since it is done accurately biologically. Indeed, it is even known that ciliates need the old MAC for the gene assembly procedure, and therefore, it is not necessary for ciliates to only use MIC sequence information to identify IESs and MDSs [30, 32]. Thus, more investigation regarding micronuclear chromosome classification with machine learning was left out of the scope of this thesis.

Table 5.5: Result analysis for fifth-order HMM (t_p = true positive, t_n = true negative, f_p = false positive, and f_n = false negative).

	Precision $(\frac{t_p}{t_p+f_p})$	**Recall or Sensitivity** $(\frac{t_p}{t_p+f_n})$	**Specificity** $(\frac{t_n}{t_n+f_p})$	**Accuracy** $(\frac{t_p+t_n}{t_p+t_n+f_p+f_n})$
Results for each of the 10 folds	0.7165	0.5874	0.5956	0.6668
	0.7209	0.6023	0.5815	0.673
	0.7154	0.5862	0.5857	0.6672
	0.7118	0.5891	0.582	0.6637
	0.7171	0.5933	0.5876	0.6642
	0.7099	0.5908	0.5812	0.6572
	0.7185	0.5996	0.5825	0.6729
	0.7154	0.6032	0.5726	0.6705
	0.7071	0.5821	0.5977	0.6602
	0.7129	0.5847	0.5912	0.6612
Average	0.71455	0.59187	0.58576	0.66569

Table 5.6: Result analysis for sixth-order HMM (t_p = true positive, t_n = true negative, f_p = false positive, and f_n = false negative).

	Precision $(\frac{t_p}{t_p+f_p})$	**Recall or Sensitivity** $(\frac{t_p}{t_p+f_n})$	**Specificity** $(\frac{t_n}{t_n+f_p})$	**Accuracy** $(\frac{t_p+t_n}{t_p+t_n+f_p+f_n})$
Results for each of the 10 folds	0.5946	0.9936	0.0196	0.6018
	0.6011	0.9952	0.0178	0.6073
	0.5902	0.9936	0.0221	0.5975
	0.6046	0.9942	0.0229	0.6115
	0.5928	0.9937	0.0221	0.6015
	0.5962	0.9936	0.0216	0.6035
	0.6045	0.9958	0.0204	0.6111
	0.6066	0.9954	0.0177	0.6127
	0.5922	0.995	0.0199	0.6003
	0.5995	0.9944	0.0216	0.6065
Average	0.59823	0.99445	0.02057	0.60537

Chapter 6
Conclusions and Future Directions

The process of gene assembly is a fascinating example of extensive computations occurring in nature, and is interesting from both biological and computational viewpoints. The manner in which ciliates perform their gene assembly process is a topic of great interest to biologists and natural computing researchers. Since the descrambling of MDSs of micronuclear chromosomes is a significant phase of gene assembly, a topic of interest is to predict the order in which operations are applied to descramble micronuclear chromosomes. This work uses the principle of parsimony to help prediction, whereby the smallest sequence of operations is likely close to the actual number of operations that occurred during the descrambling. Separately, the other topic of this thesis is to use machine learning to classify micronuclear DNA sequences into MDSs and IESs without using macronuclear DNA. It is noteworthy that both of these studies are the first attempt in the literature to predict the descrambling order from MDS sequences, and to decipher MDSs and IESs from micronuclear data alone.

6.1 Descrambling Order Analysis

In 2008 Matthias et al. found that there are occurrences of multiple parallel inversion and transposition events through each pathways during gene descrambling which results in a permutation of the MDSs [30]. Additionaly, the order of MDSs in the newly assembled 22,354 *Oxytricha trifallax* MIC chromosome fragments provides evidence that multiple parallel recombinations occur, where the structure of the chromosomes allows for interleaving between two sections of the developing macronuclear chromosome in a manner that can be captured with the shuffle operation. Thus, the descrambling order determination study focused on determining the required minimum number of applications of the shuffle operation to descramble the sections of a MIC chromosome corresponding to one MAC chromosome, where each shuffle operation is a computational representation of certain parallel transposition events.

As determining the minimum number of applications of shuffle to descramble a MIC chromosome fragment involves determining the minimum number of segments on which shuffle is applied, four similar systems involving shuffle were studied, where each system allows different types of segments. Two optimal algorithms for each of the first two systems have been implemented. And, for the third and fourth systems, four different heuristic algorithms, and two different heuristic algorithms, respectively, have been implemented. Among the considered systems, the non-contiguous increasing system gives the best results, which allows the increasing subsequences to participate in the shuffle operation. This system showed that most of the input sequences can be descrambled by very few applications of shuffle. Indeed, the analysis shows that 96.63% of the scrambled micronuclear chromosome fragments of *Oxytricha trifallax* can be partitioned into 2 to 3 segments, and thus can be descrambled by only 1 or 2 applications of shuffle. However, optimality of this algorithm remains an open question.

Therefore, this study provides theoretical evidence that a structural component is partially enforcing this shuffle-like behaviour by properly aligning segments in an interleaving fashion, and then parallel recombination is taking place for the MDS rearrangement and IES elimination. Indeed, examples from different ciliate species that have scrambled genes (for example, Figures 1.5b, 1.5c and 1.6) seem to also apply, although the full genomes have not been analyzed. This could potentially inspire new experiments to verify that such a shuffle-like parallel recombination takes place.

6.2 MDS and IES Identification with HMMs

The second topic of the thesis, classifying micronuclear chromosomes is a matter of determining the right "class label", i.e. MDS or IES, on each nucleotide. To do this, the labelled input data was fed into a well-known supervised machine learning classification model, hidden Markov models. A first-, second-, third-, fourth-, fifth-, and sixth-order HMM have been implemented, and the accuracy of the classification was verified through 10-fold cross validation. The results showed that none of the implemented HMMs could accurately classify a good percentage of MDSs and IESs, and of the implement models, the second and third order HMMs could determine the highest number of true MDSs, at 73.03%. Moreover, the increased sensitivity (99.44%), and drastically decreased specificity (2.05%) shows that the sixth order HMM is actually predicting almost all of the nucleotides as MDS. Hence, it is an open question regarding what properties of the data cause sensitivity to collapse (i.e. what is causing it to predict everything as MDS).

Therefore, this study shows that it is more likely that an HMM will fail to accurately classify micronuclear chromosomes without having some additional knowledge. Moreover, it is known that ciliates need the old MAC for the gene assembly procedure [30], and it is therefore unlikely that this classification can be done with only MIC sequences. However, this work was done as a preliminary investigation into this topic, and therefore more investigation regarding micronuclear chromosome classification with machine learning was left out of scope.

6.3 Future Directions

Some potential future works of this study are listed as following:

- Optimality analysis of NIS and NIDS.

- Investigating the practicality and potential biological mechanisms by which these systems could occur.
- Investigating other data giving structural information such as HiC, which could be potentially used to clarify certain interactions that occur between different parts of the chromosomes.

References

[1] Suhartati Agoes. A hidden Markov model for identification of exons in DNA of genes *Plasmodium falciparum. International Journal of Electrical & Computer Sciences*, 11:33–36, 2011.

[2] Arnie Berg. Exploring the Behaviour of the Hidden Markov Model on CpG Island Prediction. Master's thesis, University of Saskatchewan Saskatoon, 2013.

[3] Hans-Joachim Böckenhauer and Dirk Bongartz. *Algorithmic Aspects of Bioinformatics (Natural Computing Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[4] Wilfried Brauer, Hartmut Ehrig, Juhani Karhumäki, and Arto K Salomaa. *Formal and Natural Computing: Essays Dedicated to Grzegorz Rozenberg*, volume 2300. Springer-Verlag, Berlin Heidelberg, 1st edition, 2002.

[5] Christopher B Burge and Samuel Karlin. Finding the genes in genomic DNA. *Current Opinion in Structural Biology*, 8(3):346–354, 1998.

[6] Jonathan Burns, Denys Kukushkin, Kelsi Lindblad, Xiao Chen, Nataša Jonoska, and Laura F Landweber. ⟨ mds_ies_db⟩: a database of ciliate genome rearrangements. *Nucleic Acids Research*, page gkv1190, 2015.

[7] Andre RO Cavalcanti, Thomas H Clarke, and Laura F Landweber. MDS_IES_DB: a database of macronuclear and micronuclear genes in spirotrichous ciliates. *Nucleic Acids Research*, 33(suppl 1):D396–D398, 2005.

[8] Andre RO Cavalcanti and Laura F Landweber. Gene unscrambler for detangling scrambled genes in ciliates. *Bioinformatics*, 20(5):800–802, 2004.

[9] Andre RO Cavalcanti and Laura F Landweber. Insights into a biological computer: detangling scrambled genes in ciliates. In *Nanotechnology: Science and Computation*, pages 349–359. Springer, 2006.

[10] Xiao Chen, John R Bracht, Aaron David Goldman, Egor Dolzhenko, Derek M Clay, Estienne C Swart, David H Perlman, Thomas G Doak, Andrew Stuart, Chris T Amemiya, Robert P Sebra, and Laura F Landweber. The architecture of a scrambled genome reveals massive levels of genomic rearrangement during development. *Cell*, 158(5):1187–1198, 2014.

[11] Thomas H Cormen. *Introduction to Algorithms*. MIT press, 2009.

[12] Sean R Eddy. What is a hidden Markov model? *Nature Biotechnology*, 22(10):1315–1316, 2004.

[13] Andrzej Ehrenfeucht, Tero Harju, Ion Petre, David M Prescott, and Grzegorz Rozenberg. *Computation in Living Cells: Gene Assembly in Ciliates*. Natural Computing. Springer-Verlag, Berlin Heidelberg, 2004.

[14] Andrzej Ehrenfeucht, David M Prescott, and Grzegorz Rozenberg. Computational aspects of gene (un)scrambling in ciliates. In *Evolution as Computation*, pages 216–256. Springer, 2002.

[15] Zoubin Ghahramani. An introduction to hidden Markov models and Bayesian networks. *International Journal of Pattern Recognition and Artificial Intelligence*, 15(01):9–42, 2001.

[16] C Guthrie and B Patterson. Spliceosomal snRNAs. *Annual Review of Genetics*, 22(1):387–419, 1988.

[17] John Henderson, Steven Salzberg, and Kenneth H Fasman. Finding genes in DNA with a hidden Markov model. *Journal of Computational Biology*, 4(2):127–141, 1997.

[18] Daniel Hogan and Tony Kusalik. Stability of Hidden Markov Models When Applied to CpG-Island Prediction. Presented at 19th Annual International Conference on Research in Computational Molecular Biology (RECOMB), 2015.

[19] Yen-Lin Huang, Chen-Cheng Huang, Chuan Yi Tang, and Chin Lung Lu. SoRT2: a tool for sorting genomes and reconstructing phylogenetic trees by reversals, generalized transpositions and translocations. *Nucleic Acids Research*, page gkq520, 2010.

[20] Liu Jing. Scrambling analysis of ciliates. Master's thesis, University of Saskatchewan, Saskatoon, 2009.

[21] Neil C Jones and Pavel Pevzner. *An Introduction to Bioinformatics Algorithms*. MIT press, Cambridge, MA, 2004.

[22] J Mark Keil, Jing Liu, and Ian McQuillan. Algorithmic properties of ciliate sequence alignment. *Theoretical Computer Science*, 411(6):919–925, 2010.

[23] Laura F Landweber, Tai-Chih Kuo, and Edward A Curtis. Evolution and assembly of an extremely scrambled gene. *Proceedings of the National Academy of Sciences*, 97(7):3298–3303, 2000.

[24] Vincent Magrini, Patrick Minx, Robert S Fulton, Amy Ly, Sean McGrath, Kevin Haub, Richard K Wilson, Elaine R Mardis, et al. The *Oxytricha trifallax* macronuclear genome: A complex eukaryotic genome with 16,000 tiny chromosomes. *Public Library of Science Biology (PLoS Biol)*, 11(1):e1001473, 2013.

[25] MD Sowgat Ibne Mahmud. Formal model and simulation of the gene assembly process in ciliates. Master's thesis, University of Saskatchewan, Saskatoon, 2013.

[26] Geoffrey McLachlan, Kim-Anh Do, and Christophe Ambroise. *Analyzing Microarray Gene Expression Data*, volume 422. John Wiley & Sons, 2005.

[27] GF Meyer and HJ Lipps. Chromatin Elimination in the Hypotrichous Ciliate *Stylonychia mytilus*. *Chromosoma*, 77(3):285–297, 1980.

[28] GF Meyer and HJ Lipps. The Formation of Polytene Chromosomes during Macronuclear Development of the Hypotrichous Ciliate *Stylonychia mytilus*. *Chromosoma*, 82(2):309–314, 1981.

[29] GF Meyer and HJ Lipps. Electron microscopy of surface spread polytene chromosomes of *Drosophila* and *Stylonychia*. *Chromosoma*, 89(2):107–110, 1984.

[30] Matthias Möllenbeck, Yi Zhou, Andre RO Cavalcanti, Franziska Jönsson, Brian P Higgins, Wei-Jen Chang, Stefan Juranek, Thomas G Doak, Grzegorz Rozenberg, Hans J Lipps, et al. The pathway to detangle a scrambled gene. *Public Library of Science One (PLoS One)*, 3(6):e2330, 2008.

[31] Kevin P Murphy. *Machine Learning: a Probabilistic Perspective*. The MIT press, Cambridge, MA, 2012.

[32] Mariusz Nowacki, Vikram Vijayan, Yi Zhou, Klaas Schotanus, Thomas G Doak, and Laura F Landweber. RNA-mediated epigenetic programming of a genome-rearrangement pathway. *Nature*, 450(7172), 2007.

[33] David M Prescott. The unusual organization and processing of genomic DNA in hypotrichous ciliates. *Trends in Genetics*, 8(12):439–445, 1992.

[34] David M Prescott. The DNA of ciliated protozoa. *Microbiological Reviews*, 58(2):233, 1994.

[35] David M Prescott. Genome gymnastics: unique modes of DNA evolution and processing in ciliates. *Nature Reviews Genetics*, 1(3):191–198, 2000.

[36] David M Prescott, Andrzej Ehrenfeucht, and Grzegorz Rozenberg. Molecular operations for DNA processing in hypotrichous ciliates. *European Journal of Protistology*, 37(3):241–260, 2001.

[37] David M Prescott, Jason D Prescott, and Ryan M Prescott. Coding properties of macronuclear DNA molecules in *Sterkiella nova* (*Oxytricha nova*). *Protist*, 153(1):71–77, 2002.

[38] David M Prescott and Grzegorz Rozenberg. How ciliates manipulate their own DNA—a splendid example of natural computing. *Natural Computing*, 1(2-3):165–183, 2002.

[39] Francisco Rodríguez-Trelles, Rosa Tarrío, and Francisco J Ayala. Origins and evolution of spliceosomal introns. *Annual Review of Genetics*, 40:47–76, 2006.

[40] U Sekar. Applications of Graph Theory in Computer Science. *International Journal of Electronics Communication and Computer Engineering*, 4:2278–4209, 2013.

[41] Mario Stanke and Stephan Waack. Gene prediction with a hidden Markov model and a new intron submodel. *Bioinformatics*, 19(suppl 2):ii215–ii225, 2003.

[42] Wing-Kin Sung. *Algorithms in Bioinformatics: A Practical Introduction.* Chapman & Hall/CRC Mathematical and Computational Biology. Chapman & Hall/CRC Press, London, UK, 1st edition, 2009.

[43] Glenn Tesler. GRIMM: genome rearrangements web server. *Bioinformatics*, 18(3):492–493, 2002.

[44] Douglas Brent West. *Introduction to Graph Theory*, volume 2. Prentice Hall Upper Saddle River, 2001.

[45] Ian H Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques.* Morgan Kaufmann, 2nd edition, 2005.

[46] Hao Wu, Brian Caffo, Harris A Jaffee, Rafael A Irizarry, and Andrew P Feinberg. Redefining CpG islands using hidden Markov models. *Biostatistics*, page kxq005, 2010.

[47] Byung-Jun Yoon. Hidden Markov models and their applications in biological sequence analysis. *Current Genomics*, 10(6):402, 2009.