Scrambling Analysis of 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
LIU Jing

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building
110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

Abstract

Ciliates are a class of organisms which undergo a genetic process called gene descrambling after mating. In order to better understand the problem, a literature review of past works has been presented in this thesis. This includes a brief summary of both the relevant biology and bioinformatics literature. Then, a formal definition of scrambling systems is developed which attempts to model the problem of sequence alignment between scrambled and descrambled genes. With this system, sequences can be classified into relevant functional segments. It also provides a framework whereby we can compare various ciliate sequence alignment algorithms. After that, a new method of predicting the various functional segments is studied. This method shows better coverage, and usually a better labelling score with certain parameters. Then we discuss several recent hypotheses as to how ciliates naturally descramble genes. An algorithm suite is developed to test these hypotheses. With the tests, we are able to computationally check which factors are potentially the most important. According to the current results with 247 pointer sequences of 13 micronuclear genes, examining repeats which are the same distance together with either the sequence or the size, as the real pointers, is almost always enough information to guide descrambling. Indeed, the real pointer sequence is the unique repeat 92.7% and 94.3% of the time within the 247 pointers, from the left and right respectively, using only the pointer distance and the pointer sequence information.

Acknowledgements

I would like to give a special thanks to my supervisor Ian McQuillan for his guidance during my research and study at the University of Saskatchewan. He is always energetic, intelligent, warm-hearted, and accessible. His endless support and willingness to help myself and his students made my Masters study full of surprise and happiness. I am so proud to be his first Masters student!

I am delighted to thank Professors Mark Keil, and Tony Kusalik for being on my thesis committee members, and Professor FangXing Wu for being my external examiner. Their time and advise is appreciated.

I have been happy to know Jan, a wonderful "Mom" to the department and students. Especially as an international student, her help and support have made me feel at home.

I would like to express my deepest gratitude to my parents Liu Jian and Hu Lisha for their unconditonal love throughout my life, and to my boyfriend Bao Lin for his love and support 24 hours a day, 7 days a week. They have made me feel happy and lucky everyday.

I would also like to thank all my friends and lab buddies. Their kindness, help, and inspiring ideas have made my study and work much easier and enjoyable.

Thanks to everyone who have helped and supported me, thanks to everyone who have given me advise, and thanks for my choice of taking a Masters Degree in the Department of Computer Science at the University of Saskatchewan.

Contents

Permission to Use i

Abstract ii

Acknowledgements iii

Contents iv

List of Tables vi

List of Figures vii

List of Abbreviations viii

1 Introduction 1
- 1.1 Motivation and objectives . . . 1
- 1.2 Thesis structure . . . 2

2 Preliminary works 3
- 2.1 Biological preliminaries . . . 3
  - 2.1.1 Introduction to ciliate biology . . . 3
  - 2.1.2 Gene structure and behaviour . . . 3
- 2.2 Descrambling hypotheses . . . 5
  - 2.2.1 Pointer-guided recombination . . . 5
  - 2.2.2 Template-guided recombination . . . 7
  - 2.2.3 RNAi . . . 7
  - 2.2.4 DNA structure . . . 7
- 2.3 Bioinformatics preliminaries . . . 9
  - 2.3.1 Sequence alignment . . . 9
  - 2.3.2 Smith-Waterman algorithm . . . 10
  - 2.3.3 BLAST . . . 13
  - 2.3.4 Ciliate-based bioinformatics micronuclear and macronuclear segment partitioning . . . 13

3 Formal modelling of Ciliate scrambling systems 15
- 3.1 Motivation . . . 15
- 3.2 Scrambling system . . . 15
- 3.3 Labelling a scrambling system . . . 19

4 New algorithm to partition micronuclear genes and macronuclear genes 22
- 4.1 Motivation . . . 22
- 4.2 Problem description . . . 22
- 4.3 Criteria to analyze the accuracy of the algorithms . . . 23
- 4.4 Algorithm description . . . 24
- 4.5 Algorithm analysis . . . 26
- 4.6 Result analysis . . . 27
  - 4.6.1 Analysis of the number of MDSs detected . . . 29
  - 4.6.2 Coverage analysis . . . 31
  - 4.6.3 Labelling score analysis . . . 31
  - 4.6.4 Program running time analysis . . . 35

4.6.5 Comparison between the algorithms and parameters 35
4.6.6 Special cases 37

5 Genetic scrambling analysis based on biological hypotheses and a bioinformatics algorithm 38
5.1 Motivation 38
5.2 An algorithm suite based on the structure model 39
5.2.1 Definitions of basic terms used in the algorithm 39
5.2.2 Definition of the algorithm suite 40
5.3 Algorithm description 41
5.4 Result analysis 46

6 Conclusion 55

References 59

A PERL code of the new algorithm to partition micronuclear genes and macronuclear genes 60

B PERL code of evaluating A-L algorithm 71

C PERL code of the new algorithm suite to test the hypotheses 78
C.1 Program that uses window to take control of distance 78
C.2 Program that ignores distance 80

D Result of the new algorithm suite 83

List of Tables

2.1 Example of Smith-Waterman algorithm 10

4.1 Numerical identifier of each macronuclear gene and micronuclear gene pair 28
4.2 Number of MDSs detected by the various algorithms and parameters used 29
4.3 Coverage scores by the various algorithms and parameters used 31
4.4 Labelling scores by the various algorithms and parameters used 33
4.5 Program running time by the various parameters used 35

5.1 Statistical summary of the tested database 46
5.2 An example of results using the algorithm suite with parameter KnotDS and KnotDSe on micronuclear gene number 19 49
5.3 An example of results using the algorithm suite with parameter KnotD and KnotDSe3 on micronuclear gene number 19 50
5.4 An example of results using the algorithm suite with parameter KnotS and KnotSe on micronuclear gene number 19 51
5.5 Percentage of unique sequences in each micronuclear gene, each group, and the total 13 genes with each variant 52
5.6 Minimum slithering range of each micronuclear gene, each group, and the total 13 genes with each variant 53
5.7 Median slithering range of each micronuclear gene, each group, and the total 13 genes with each variant 54

D.1 Result of KnotDS, and KnotDSe on all 13 micronuclear genes 83
D.2 Result of KnotD, and KnotDSe3 on all 13 micronuclear genes 87
D.3 Result of KnotS, and KnotSe on all 13 micronuclear genes 91

List of Figures

2.1 Two of the most well-studied classes of ciliates 4
2.2 Conversion from a hypothetical micronuclear gene to a macronuclear gene 4
2.3 Hypothesized process of pointer guided recombination 6
2.4 A knot that represents a DNA sequence 8
2.5 Result of recombination with a direct and an inverted repeat 8
2.6 Hypothetical structure of a micronucleus before and after slithering 9
2.7 An example of a sequence alignment between sequences ATCGAATCA and ATGAATAA 10

3.1 Traditional definition of MDS 15
3.2 New definition of MDS 16
3.3 Micronuclear classification 16
3.4 Macronuclear classification 17
3.5 Labelling of a valid scrambling system 19

4.1 Number of MDSs detected by the various algorithms and parameters used 30
4.2 Coverage scores by the various algorithms and parameters used 32
4.3 Labelling scores by the various algorithms and parameters used 34
4.4 Program running time by the various parameters used 36

5.1 Hypothetical structure of a micronucleus before and after slithering 39
5.2 Window structure explanation 40
5.3 Sliding window explanation 40
5.4 Sample result of algorithm using variants KnotD and KnotDSe3 43
5.5 Sample result of algorithm using variants KnotDS and KnotDSe 44
5.6 Sample result of algorithm using variants KnotS and KnotSe 45

List of Abbreviations

MIC	micronuclear gene
MAC	macronuclear gene
MDS	macronuclear destined sequence
IES	internal eliminated sequence
A-L	MDSs detecting algorithm introduced in [3]
m	match score
mm	mismatch penalty
g	gap penalty
p50c5	the percentage penalty is 50%, and the cutoff score is 5
p50c10	the percentage penalty is 50%, and the cutoff score is 10
p0c5	the cutoff score is 5, no percentage penalty
p0c10	the cutoff score is 10, no percentage penalty
KnotD	the variant only considering the window distance constraint
KnotS	the variant only considering the window size constraint
KnotSe	the variant only considering the window sequence constraint
KnotDS	the variant considering both window distance and window distance constraint
KnotDSe	the variant considering both window distance and window sequence constraint
KnotDSe3	the variant considering both window distance and first three nucleotides of window sequence constraint

Chapter 1

Introduction

Ciliates are a group of organisms that, after mating, undergo massive genetic reorganization including a process known as gene descrambling. This process has captured the interest of evolutionary biology and bioinformatics researchers. The research area of this thesis involves sequence analysis of gene descrambling. The biological mechanisms involved with scrambling and descrambling are still not completely understood. Many biological experiments, and several biological hypotheses have been proposed, although there is no firm consensus as to which ones are true.

1.1 Motivation and objectives

The goals of this thesis are divided into three main categories. The first part is to present a review of the relevant previous literature, which will help the readers understand ciliate descrambling both from a biological perspective and a bioinformatics perspective. The second goal is to predict a partition of ciliate genes into known types of functional segments, which can be done by studying sequence alignment of scrambled genes with unscrambled genes. The third goal is to create an algorithm to test hypotheses as to how ciliates descramble biologically.

Aligning scrambled genes with unscrambled genes is different from the traditional sequence alignment problem. During scrambling, various segments are spliced out, and the remaining parts get rearranged. Hence, there is a more complex relationship between the various sections of the strings (although there are other ways of aligning strings besides using a left-to-right ordering, such as A-Bruijn alignments [25]). To identify those segments, a custom variant of a sequence alignment algorithm is needed.

There is only one existing algorithm to do this task [3], and no formal description of the problem has been developed. As a result, a formal model is needed which would provide a clear goal as to what the alignment algorithms should accomplish, and also to provide a way to assess the quality of alignments. This would also allow a comparison of various algorithms which perform this type of alignment.

There are many different hypotheses that currently exist to predict how the descrambling takes place, but there are not any computational methods for assessing the feasibility of any of these

hypotheses. Being able to computationally test the importance of different factors in descrambling, which are abstracted from biological hypotheses, would help biologists to establish or reject their feasibility, modify and refine hypotheses, and better design experiments to help find the actual mechanisms.

1.2 Thesis structure

In Chapter 2, a survey of the related preliminary works is given. In Chapter 3, a new formal model called ciliate scrambling systems is introduced. This attempts to model sequence alignment of ciliate descrambling. From there, in Chapter 4, we study the problem of computationally partitioning micronuclear genes into the functional regions. A labelling algorithm based on the Smith-Waterman local sequence alignment is created and discussed. In Chapter 5, we address the importance of the structure of the DNA in gene descrambling and whether or not the structure provides enough information to guide descrambling. Finally, in Chapter 6, we provide the conclusion of the thesis.

In summary, we are concerned with two perspectives. The first one is creating bioinformatics algorithms for detecting and labelling alignments for ciliates. The second one is attempting to test the feasibility of biological hypotheses.

Chapter 2

Preliminary works

2.1 Biological preliminaries

2.1.1 Introduction to ciliate biology

Ciliates are a group of organisms which belongs to superkingdom Eukaryota, superphylum Alveolata, and the Ciliophora phylum[1]. They are estimated to have originated 10^9 years ago, separating from the eukaryotic line before fungi appeared. They have diverged into a rich family containing many thousands of species. In Figure 2.1, we see two of the well-studied classes in the phylum: Spirotrichea and Oligohymenophorea [20]. The genetic distance within these classes is very large. For example, *Euplotes* (in Spirotrichea) and *Tetrahymena* (in Oligohymenaphorea) have a genetic distance between them which is approximately the same as the distance between corn and rat [9].

2.1.2 Gene structure and behaviour

There are some known genetic behaviours of ciliates which are unique. Their DNA is stored in two types of nuclei: a transcriptionally silent nucleus called the *micronucleus* (*MIC*), which is used during mating for sexual exchange of DNA, and a somatic nucleus called the *macronucleus* (*MAC*) which is trancriptionally active, supports vegetative cell growth and cell proliferation [20]. Thus, only the genes in the MAC get transcribed and translated. 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.

The method by which ciliates develop a new macronucleus from the new micronucleus is not completely understood. However, when a gene in the MIC generates a MAC gene, certain segments of the MIC gene are removed. The remaining parts which do not get removed are called *macronuclear destined sequences* (*MDSs*). The segments which get removed are called *internal eliminated sequences* (*IESs*). A hypothetical micronuclear gene is drawn in Figure 2.2. Hence, accurate removal of IESs is necessary in the conversion. Moreover, in some genes of some types

[1] This is the classification according to the classification in the online UniProt taxonomy database – NEWT. They used to be classified in kingdom Protista, although this classification is no longer recognized [19].

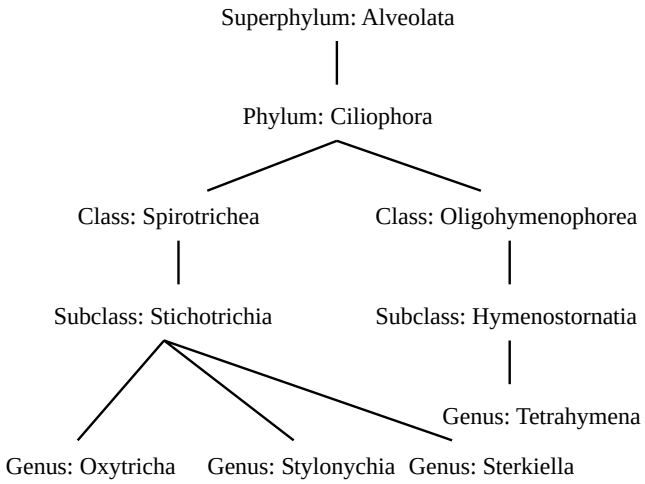

Figure 2.1: Pictured above are two of the most well-studied classes of ciliates (we have omitted part of the diagram for simplicity) [19].

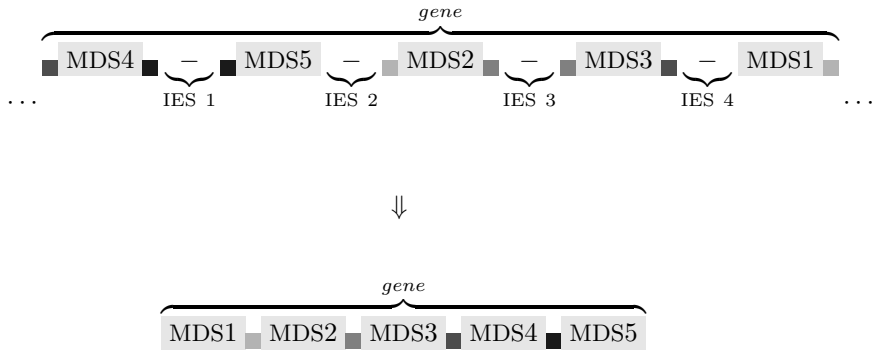

Figure 2.2: Pictured above is the conversion from a hypothetical micronuclear gene (on the top) to a macronuclear gene (on the bottom). The order of the MDSs in the micronuclear gene is permuted from the macronuclear gene. Matching pointers are shown with the same shade of grey, adjacent to MDSs. Only one copy of each pointer is retained in the macronuclear gene.

of ciliates, the order of MDSs is different between the macronuclear copy of each gene, and the micronuclear version. These are called scrambled genes [21, 22, 23]. For example, in Figure 2.2, the order of MDSs for this hypothetical micronuclear gene is $4, 5, 2, 3, 1$ and is $1, 2, 3, 4, 5$ in the MAC gene (we always label the MDSs of MAC genes in the "natural" ordering from 1 to n where n is the number of MDSs). Gene scrambling is only known to occur in the class Spirotrichea, and not in the class Oligohymenophorea, containing the model organism *Tetrahymena thermophila*, although IESs occur in both [20]. Because ciliates rearrange scrambled genes, we say that they *descramble* micronuclear genes [16].

In the diploid MIC, genes are separated from each other by large amounts of spacer DNA on large chromosomes. However, in the conversion to a new macronucleus, all chromosomes become gene-sized (one gene per chromosome) and each is amplified to between one and several thousand copies. Thus, the macronucleus is polyploid [20].

In both scrambled and non-scrambled genes, MDSs share some similar features. A sequence of nucleotides at the end of an MDS is repeated at the beginning of the next consecutive MDS (consecutive in terms of the ordering in the MAC). These sequences are called *pointers*. The lengths are known to vary from between 2 to 20 nucleotides. However, a single repeat of the pointer is maintained in the macronuclear gene (see Figure 2.2).

2.2 Descrambling hypotheses

The mechanisms used by ciliates to descramble genes are still not completely understood. However, there have been numerous hypotheses as to how the descrambling occurs. We will outline some of them here. The models try to address descrambling from different perspectives and at different levels of abstraction, each perhaps only capturing part of the process. As such, they do not necessarily conflict with each other.

2.2.1 Pointer-guided recombination

Traditionally, it was hypothesized that descrambling was guided with the help of pointers. That is, matching pointers could become aligned together whereby recombination between the pointers would be possible in the process. However, this hypothesis on its own assumes that the pointers can provide enough information to guide the alignment properly. That is, there must be some repeat sequence that occurs, and then an alignment between those two sites would be possible. For example, in Figure 2.3, the repeat sequence X occurs twice. Then, the two copies of X could become aligned together, at which point recombination would be possible resulting in the excision of the IES.

There have been two major computational models proposed in this category which attempt to

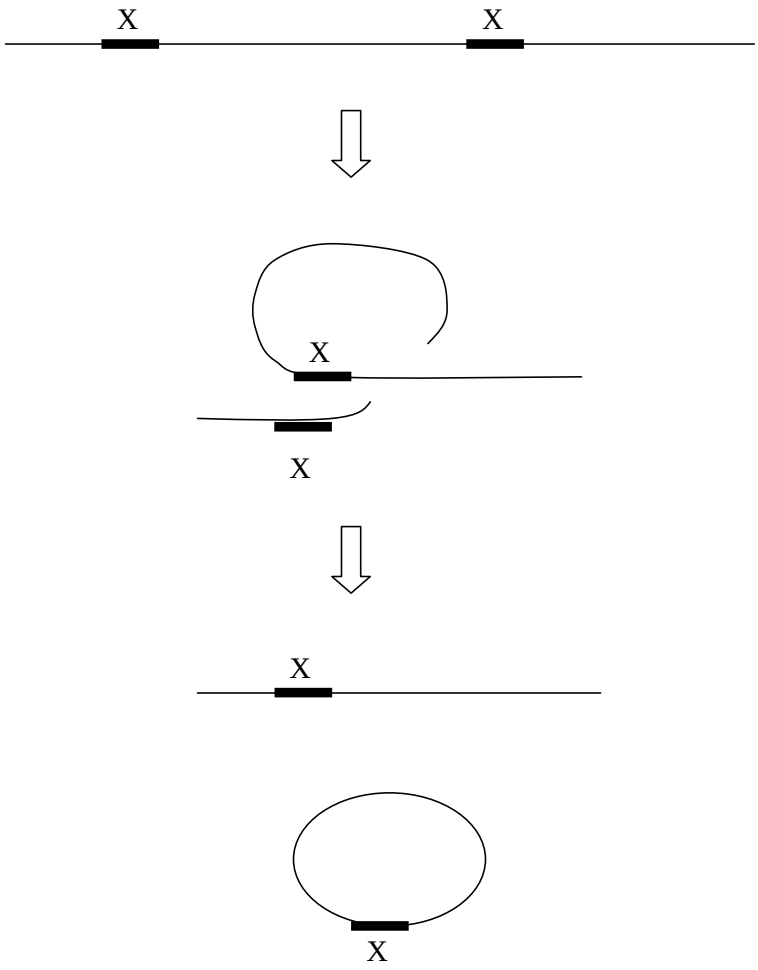

Figure 2.3: Drawn above is the hypothesized process of pointer guided recombination. The repeat sequence X occurs twice and the segment between the two copies of X is eliminated as an IES.

address the manner in which matching real pointers are found. The first was of Kari and Landweber [11, 15]. The other one is the model by Ehrenfeucht, Prescott, and Rozenberg [8].

It was noted in [4] that this model on its own can only uniquely descramble if pointer sequences are unique, or if there are not many repeats. However this is not the case biologically. There are often repeats that are not real pointers. These are called *fake pointers*.

We will further address the feasibility of this hypothesis in Chapter 5.

2.2.2 Template-guided recombination

Template-Guided DNA Recombination was a biological model that was created to address some of the limitations with pointer guided recombination [7, 24, 5]. They hypothesized that the old macronucleus is involved in the genetic recombination. We know that after mating, the old macronulceus gets destroyed. However, in this model, the authors propose that genetic material from the old macronucleus colocates to the developing macronucleus, and aides in the alignment of consecutive pointers. They also hypothesize a particular enzymatic mechanism which could accomplish this task. As support of this hypothesis, in [23] the authors inject DNA into the old macronucleus, which would then affect the resulting genes. This gives evidence of the importance of the old macronucleus in descrambling.

2.2.3 RNAi

In 2008, Meng-Chao Yao presented a new model, which states that during sexual reproduction, although the macronucleus is not directly transferred to the progeny, an RNA copy of it could serve as a template for the unscrambling of the rearranged micronuclear DNA to give rise to the new macronucleus [26]. Indeed, small RNAs of length 21-26 bp are detected in ciliates during descrambling [18]. Whether the entire macronuclear gene segments are important in descrambling, or just the small RNA segments generated from the macronucleus is still unknown. However, experiments show that changing these segments will result in the failure to descramble.

2.2.4 DNA structure

The DNA structure hypothesis attempts to address the manner in which real pointers are aligned together [6]. It examines how the three-dimensional structure of the micronuclear DNA could impact pointer matching. The authors propose that the micronucleus folds in such a way that real pointers are brought together in proximity. Then, recombination between real pointers can occur to produce an unscrambled gene [6].

The authors of [6] build this model using *knot theory*. In their study, the micronuclear sequence is represented as a knot [6] such as that of Figure 2.4. When a recombination event takes place,

Figure 2.4: Pictured above is a knot that represents a DNA sequence [6].

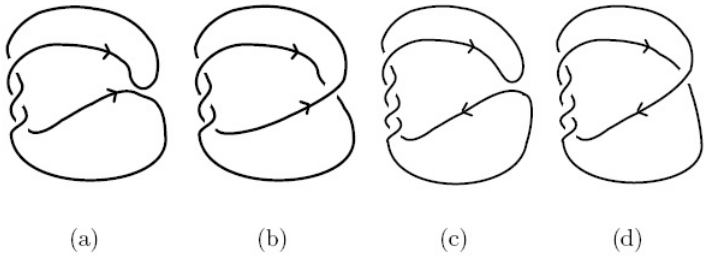

Figure 2.5: Pictured above is the result of recombination with a direct repeat, (a) to (b), and with an inverted repeat, (c) to (d) [6].

the recombination will alter the topology of the DNA molecule as a result of *supercoiling*. This is a process whereby a DNA molecule twists around itself. This model proposes that after a recombination event, the DNA will supercoil creating a new topology, which will bring together the MDSs that are to be descrambled next. Indeed, recombination is known to either positively or negatively supercoil the DNA, which could then modify the topology to juxtapose the next new MDS to be descrambled (as is known to occur biologically [10, 17]). This process is repeated until the gene is descrambled.

Interpreting a DNA molecule as a knot, when recombination occurs, the strands with a direct repeat produce a link (two intertwined knots) with an inverted repeat. For example, Figure 2.5(a) has a direct repeat which recombines to form a link (b) with an inverted repeat. On the other hand, a knot with an inverted repeat produces a knot with a direct repeat after recombination as with (c) to (d). Such DNA topology changes have been characterized biologically in [14]. Furthermore, the pointer regions would not need to be brought together completely, but instead both sites could be shifted by the same amount. Indeed, within a fixed structure, DNA is known to perform a one dimensional movement known as "slithering". So essentially, pointers would only need to match after slithering (Figure 2.6) [10, 17].

Based on this hypothesis, the topological structure could be used to help align pointers. To do this, first, the static structure could be formed where matching pointers are a certain distance apart.

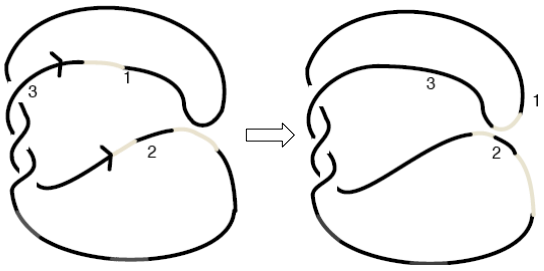

Figure 2.6: We have a picture of the hypothetical structure of a micronucleus before (left) and after (right) slithering. The numbers represent MDSs, while the IESs are in grey. Positions 1 and 2 and 3, along with the given distances between them, move together while the structure does not change. The arrows show the direction of slithering.

The DNA could then slither within the fixed structure which could bring together the potential pointer sequences (see Figure 2.6). If this hypothesis is true, then repeats would only be identified as being pointers if they both are separated by the same distance, or distance range, as the distance between the two positions of the recombination site. That is, if some topological structure formed followed by one-dimensional "slithering", then repeats would only be identified as pointers if it could slither so that the repeats are aligned. We will further describe this hypothesis in Chapter 5.

2.3 Bioinformatics preliminaries

2.3.1 Sequence alignment

In bioinformatics, a sequence alignment is a way of arranging the primary sequences of DNA, RNA, or proteins in order to identify regions of similarity. This may have consequences to the functional, structural, or evolutionary relationships between the sequences. Normally, the aligned sequences of nucleotides or amino acid residues are represented as rows within a matrix, as in Figure 2.7. There are three main types of changes that can occur to DNA over time, which are mutations, insertions and deletions. Insertions and deletions are far less common than mutations. In Figure 2.7, we use the gap symbol "–" to represent a gap, which could be the result of an insertion from the first sequence or a deletion from the second sequence. A mismatch could be the result of a mutation on one of the sequences. Given a set of sequences, many alignments can be constructed, and a score can be given for each, which depends on the quantity of gaps and mismatches. The higher the score, the more similar the sequences will be [13]. Typically, this is done by defining a match score, mismatch score and gap penalty. The scores for each position of the alignment are summed to obtain a score for the entire alignment.

A	T	C	G	A	A	T	C	A
A	T	—	G	A	A	T	A	A

Figure 2.7: The figure above is an example of a sequence alignment between sequences ATCGAATCA and ATGAATAA. If the match score is 1, the mismatch score is -1 and the gap penalty is -2, then the score for the alignment is 4. There is a gap in the third column of the alignment and a mismatch in the eighth column.

		C	G	T	A
	(0,0) 0	(0,1) 0	(0,2) 0	(0,3) 0	(0,4) 0
C	(1,0) 0	(1,1) 1	(1,2) 0	(1,3) 0	(1,4) 0
C	(2,0) 0	(2,1) 1	(2,2) 0	(2,3) 0	(2,4) 0
G	(3,0) 0	(3,1) 0	(3,2) 2	(3,3) 0	(3,4) 0
T	(4,0) 0	(4,1) 0	(4,2) 0	(4,3) 3	(4,4) 0

Table 2.1: This is an example of Smith-Waterman algorithm performed on strings $s = CCGT$ and $r = CGTA$. At position (i, j) of the matrix (where indexing starts at 0), the score of the best local alignment of the first i characters of s with first j characters of r is stored.

There are generally two main ways used to computationally align sequences: *global alignments* and *local alignments*. Global alignments align and detect the similarity along the entire sequences. By contrast, local alignments identify regions of similarity within long sequences.

2.3.2 Smith-Waterman algorithm

The Smith-Waterman Algorithm is a *local sequence alignment* method commonly used in bioinformatics [13]. As the algorithm calculates local alignments, the method does not focus on the similarity of both entire sequences, but does find similar regions between sequences. As such, it will be an important part of Chapter 4, in the detection of similar regions between the micronucleus and the macronucleus. An example of a computation used within the Smith-Waterman algorithm is shown in Table 2.1.

When aligning two sequences s and r[1], this algorithm creates a matrix of size $|s| + 1$ by $|r| + 1$, whereby at position (i, j) of the matrix, the score of the best local alignment of the first i characters

[1]The length of a string s is denoted by $|s|$.

of s with the first j characters of r is stored. As Table 2.1 shows, the two sequences s and r are assigned to the first column and row respectively. The algorithm calculates the value of each position in the matrix according to the algorithm below. In this example, the gap penalty is -2, mismatch penalty is -1, and the match score is 1, which are indicated by g, mm and m respectively, and the matrix is named M. Finally, the score of the best local alignment will be the biggest value of the table.

The algorithm to calculate the optimal local alignment for each prefix of s and r operates according to the following recurrence relation: $x_{(i,0)} = 0, \forall i \geqslant 0; x_{(0,j)} = 0, \forall j \geqslant 0$; and for every $i, j \geqslant 1$,

$$x_{(i,j)} = max \begin{cases} x_{(i-1,j-1)} + m, & \text{if } s_i = r_i, \\ x_{(i-1,j-1)} + mm, & \text{if } s_i \neq r_i, \\ x_{(i-1,j)} + g, \\ x_{(i,j-1)} + g, \\ 0. \end{cases}$$

The algorithm which calculates all values of this recurrence is given in Algorithm 1.

For example, at $M_{(1,1)}$, the match score is $0+1=1$, the two gap scores are both $0-2=-2$, of which the maximum value is 1, which is then stored in position $M_{(1,1)}$. At $M_{(2,1)}$, the match score is $0+1=1$, the two gap scores are $0-2=-2$ and $1-2=-1$, the value with the maximum being 1. At $M_{(1,2)}$, the match score is $0-1=-1$, the two gap scores are $0-2=-2$ and $1-2=-1$, with the maximum value being -1, which is lower than 0. Thus the value of $M_{(1,2)}$ is 0. We can then find the maximum score of the entire matrix to find the optimal local sequence alignment of s and r. In this example, this value is stored at $M_{(4,3)}$, which is 3. From there we can determine the actual alignment by backtracking within the matrix from the highest value until we hit 0. We can determine the alignment from right to left by examining whether a match, gap, or a mismatch led to the maximum, thus giving us the last position of the alignment. In this example, the highest value 3 which was maximal when the match score m was added from the score at $M_{(3,2)}$, and the value 2 in $M_{(3,2)}$ was maximal from the match score m added to $M_{(2,1)}$, the value 1 in $M_{(2,1)}$ was maximal from the match score m added from $M_{(1,0)}$. Then 0 is reached, which gives the end of the local alignment. So $M_{(2,1)}$, $M_{(3,2)}$, and $M_{(4,3)}$ are marked as a match, which represents CGT of $R_{(1...3)}$ and CGT of $C_{(2...4)}$. This is the best local alignment of the two given sequences.

The *Needleman-Wunsch* algorithm is similar except that it performs a global sequence alignment. The only changes that need to be made to the algorithm above is to initialize the first row and column to be multiples of the gap penalty, and to remove "0" from the recurrence relation [13], as well as backtracking from the bottom right hand corner to the top left hand corner, so as to find a global alignment instead of a local alignment.

Algorithm 1: Smith-Waterman algorithm [13]

Input: two input sequences s, r with m the match score, mm the mismatch penalty and g, the gap penalty.

Output: An $(|s|+1) \times (|r|+1)$ matrix M with entry (i, j) containing the best local alignment score of $s(1 \ldots i-1)$ with $r(1 \ldots j-1)$.

```
/* Initialize the first line and first column to be zero.  */
```

while $0 \leq i \leq |s|$ **do**
  $M_{(i,0)} = 0$;
end

while $0 \leq j \leq |r|$ **do**
  $M_{(0,j)} = 0$;
end

```
/* Calculating other scores in matrix.  */
```

while $1 \leq i \leq |s|$ **do**
  while $1 \leq j \leq |r|$ **do**

```
/* Calculating the match and mismatch score of position (i,j).  */
```

    if $s_i = r_j$ **then**
      $X = M_{(i-1,j-1)} + m$;
    else
      $X = M_{(i-1,j-1)} + mm$;
    end

```
/* Calculating the gap scores of position (i,j).    */
```

    $gp1 = M_{(i-1,j)} + g$;
    $gp2 = M_{(i,j-1)} + g$;

```
/* Compare and assign the highest score to M(i,j).  */
```

    $M_{(i,j)} = Max(X, gp1, gp2)$;

```
/* If the score less than zero, then set the score to be zero.  */
```

    if $M_{(i,j)} \leq 0$ **then**
      $M_{(i,j)} = 0$;
    end
  end
end

2.3.3 BLAST

BLAST, which stands for Basic Local Alignment Search Tool, is an algorithm for comparing primary biological sequence information [13]. Using BLAST, a researcher can compare a sequence with a library or database of sequences. It can also be used, as with regular local sequence alignment, to align two sequences to each other. For example, given a group of newly discovered genetic sequences, a BLAST search could be applied to the human genome to check for the existence of similar genes between the human genome and the newly discovered sequences. Essentially, it calculates local alignments, however they are not guaranteed to be optimal.

BLAST is based on a combination of short exact matches and threshold techniques, which enables it to be far faster than an optimal sequence alignment algorithm like the Smith-Waterman algorithm discussed above. However, BLAST obtains the speed at the expense of sensitivity [13, 1].

BLAST uses the concept of an *HSP*, short for *High Scoring Sequence Pair*, which is the fundamental unit of BLAST algorithm output. An HSP consists of two sequence fragments whose alignment is locally maximal and for which the alignment score meets or exceeds a threshold or cutoff score.

Associated with a given score, the *E-value* can be calculated. The *E-value* is the expected number of random sequences obtaining the same score or better by chance.

2.3.4 Ciliate-based bioinformatics micronuclear and macronuclear segment partitioning

In [3], the authors wrote a PERL script which took a micronuclear and macronuclear gene as input and attempted to identify IESs, MDSs, pointers and the relationship between the segments. Then it provides a visual diagram of the location of MDSs, with lines between those in the MIC and the MAC. The authors first find the longest repeat between the MIC and the MAC, then the next consecutive MDS with the same pointer is detected.

The algorithm uses *high scoring segment pairs (HSPs)* as follows:

1. The HSP with the lowest e-value is kept and assumed to be an MDS.

2. It then finds HSPs in which there is an overlap to the extremities of the previous macronuclear MDS.

3. It repeats step 2 until no overlapping HSPs can be found, and searches for the HSP that is closest to the end of the previous MDS. This is considered a new MDS.

4. It repeats step 2 and step 3 until no more HSPs are found in the remaining part of the macronuclear gene.

They also tried the algorithm on non-scrambled genes. Their algorithm only attempts to identify the different segments, but does not align the matching segments. In order to discuss and compare the algorithms, we refer to this algorithm as the A-L algorithm in this thesis.

Chapter 3

Formal modelling of Ciliate scrambling systems

3.1 Motivation

As discussed in Chapter 2, different methods can be used to align macronuclear genes with micronuclear genes. In order to formally compare the algorithms and extend the traditional definition of alignments, a formal model is created which can be used to capture the idea behind a micronuclear gene aligning with a macronuclear gene.

3.2 Scrambling system

Typically, in the literature, MDSs are defined to include the adjacent pointer regions. This is depicted in Figure 3.1, where the darker regions are pointer areas and the lighter regions represent the MDS areas. We would like to identify these pointer regions in order to categorize each section of the micronucleus and macronucleus into exactly one type of region. Hence, we define a reduced MDS denoted $\overline{MDS}$, which contains the MDS, but excludes the pointer regions. This is pictured in Figure 3.2. Then, each position of a micronuclear gene is either part of an IES, an $\overline{MDS}$, or a pointer, and is only a part of one such region (Figure 3.3). Similarly, each position of a macronuclear gene is a part of exactly one of an $\overline{MDS}$, or a pointer, as shown in Figure 3.4. In general, any macronuclear or micronuclear segment can have their positions partitioned into $\overline{MDS}$, IES, and pointer sets.

The following definition of a *micronuclear classification* is any way of partitioning the intervals of a micronuclear segment into the three categories. There are many possible classifications, with some of better quality than others. This is akin to the definition of an alignment, which does not

Figure 3.1: Traditionally, the literature on ciliates includes pointers as being part of the MDS. In the image above, the pointers are in a darker shade of grey and are included in the MDS.

Figure 3.2: We provide a new definition called $\overline{MDS}$ which has all the nucleotides of an individual MDS but excludes the pointers.

Figure 3.3: On top, we have a hypothetical micronuclear gene. On the bottom, we provide a classification of that gene. Each position of the micronuclear gene is either part of an IES, an $\overline{MDS}$, or a pointer.

necessarily need to be the best possibility.

Definition 3.2.1 *A micronuclear classification of a string s over $\{A, T, C, G\}$ is a tuple, $\gamma = (\overline{MDS}, IES, P)$ where*

$$\overline{MDS} \subseteq \mathbb{N} \times \mathbb{N}, IES \subseteq \mathbb{N} \times \mathbb{N}, P \subseteq \mathbb{N} \times \mathbb{N},$$

and $(x, y) \in \overline{MDS} \cup IES \cup P$ implies $x \leqslant y$, and for each i, $1 \leqslant i \leqslant |s|$, there exists exactly one (x, y) in one of $\overline{MDS}$, IES, or P such that $x \leqslant i \leqslant y$. We call $\overline{MDS}$ the set of $\overline{MDS}$s, IES the set of IESs, and P the set of pointers.

Usually we will consider strings s which are micronuclear genes, or gene segments. According to the definition above, each position i in a micronuclear classification γ belongs to either part of an IES, a $\overline{MDS}$, or a pointer segment. The micronuclear classification γ is the union of segments whereby each position belongs to exactly one of $\overline{MDS}$, or IES, or pointer. Essentially, each pair in one of $\overline{MDS}$, IES, or P gives the position of that particular $\overline{MDS}$, IES, or pointer respectively.

The definition below provides a different way of partitioning micronuclear genes into segments of MDSs which include both flanking pointers (or one pointer if it is the first or last MDS). This definition will also be useful for the alignment model in matching micronuclear and macronuclear pairs.

Definition 3.2.2 *Let $\gamma = (\overline{MDS}, IES, P)$ be a micronuclear classification of a string s. We define the set seg_γ to be the set of all intervals following the pattern $(P \quad \overline{MDS} \quad P)$. That is,*

$$seg_\gamma = \{(i, j) \mid (i, k) \in P, (k + 1, l) \in \overline{MDS}, (l + 1, j) \in P, \text{ for some } i, k, l, j \}.$$

We also define the set

$$start_\gamma = \{(i, j) \mid (i, k) \in \overline{MDS}, (k + 1, j) \in P, \text{ and either } i = 1 \text{ or position } i - 1 \text{ is not in } P \},$$

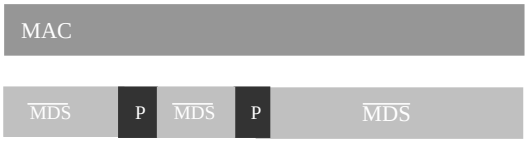

Figure 3.4: We provide a classification of a hypothetical macronuclear gene. Each position of a macronuclear gene is either part of an $\overline{MDS}$, or a pointer.

and the set

$$end_\gamma = \{(i,j) \mid (i,k) \in P, (k+1,j) \in \overline{MDS},\ \textit{and either } j = |s| \ \textit{ or position } j+1 \textit{ is not in } P\ \}.$$

Roughly, the segments in seg_γ follow the traditional notion of MDSs which include adjacent pointers, while $start_\gamma$ will contain the first MDS which does not have a pointer before it, and end_γ will contain the last MDS which does not have a pointer after it.

Some micronuclear classifications do not make sense biologically, so next we provide a type of "filter" to exclude some.

Definition 3.2.3 *Let $\gamma = (\overline{MDS}, IES, P)$ be a micronuclear classification of a string s. Then the micronuclear classification γ is* valid *if and only if γ consists of alternating pairs of elements between $seg_\gamma \cup start_\gamma \cup end_\gamma$, and IES, and also $start_\gamma$ and end_γ both contain one element.*

We are only interested in valid classifications, as real micronuclear genes alternate between IESs and MDSs, and contain one start MDS and one end MDS.

Example 1 *According to Figure 3.3, we provide a micronuclear classification $\gamma = (\overline{MDS}, IES, P)$, where*

$$\begin{aligned}
\overline{MDS} &= \{(42,72),(140,180),(272,307)\},\\
IES &= \{(0,30),(86,126),(181,271),(319,352)\},\\
P &= \{(31,41),(73,85),(127,139),(308,318)\},\\
start_\gamma &= \{(272,318)\},\\
seg_\gamma &= \{(31,85)\},\\
end_\gamma &= \{(127,180)\}.
\end{aligned}$$

Also, γ is a valid micronuclear classification since the segments alternate between IESs and elements of $seg_\gamma \cup start_\gamma \cup end_\gamma$, and $start_\gamma$ and end_γ each contain only one element.

Similarly, we define the classification of a macronuclear segment.

Definition 3.2.4 *The macronuclear classification of a string r over $\{A, T, C, G\}$ is a tuple, $\beta = (\overline{MDS}, P)$ where*

$$\overline{MDS} \subseteq \mathbb{N} \times \mathbb{N}, P \subseteq \mathbb{N} \times \mathbb{N},$$

and $(x, y) \in \overline{MDS} \cup P$ implies $x \leqslant y$, and for each i, $1 \leqslant i \leqslant |r|$, there exists exactly one (x, y) in one of $\overline{MDS}$ or P such that $x \leqslant i \leqslant y$.

According to the definition above, each position i in a macronuclear classification β belongs to either part of an $\overline{MDS}$, or a pointer segment. The macronuclear classification β is the partition of the positions into either $\overline{MDS}$, or P. Each pair in one of $\overline{MDS}$, or P gives the position of that particular $\overline{MDS}$, or pointer respectively.

We can exclude certain macronuclear classifications which do not occur biologically.

Definition 3.2.5 *We say that β is* valid *if the order of $\overline{MDS}$s, and pointers is in the following regular expression* [1]*:*

$$\overline{MDS}(P \quad \overline{MDS})^+$$

Furthermore, we define the first $\overline{MDS}$ with the adjacent pointer after it to be $start_\beta$, the last $\overline{MDS}$ with the adjacent pointer before it to be end_β, and every other $\overline{MDS}$ with the adjacent pointer before and after it to be seg_β. Formally,

$$\begin{aligned} start_\beta &= \{(1, j) \mid (1, k) \in \overline{MDS}, (k+1, j) \in P\}, \\ end_\beta &= \{(i, j) \mid (i, k) \in P, (k+1, j) \in \overline{MDS}, \text{ and position } j{+}1 \text{ does not exist } \}, \\ seg_\beta &= \{(i, j) \mid (i, k) \in P, (k+1, l) \in \overline{MDS}, (l+1, j) \in P, \text{ for some } i, k, l, j \}. \end{aligned}$$

Notice that a position can be in two intervals of seg_β simultaneously if they are in the pointer region.

Now that we have defined classifications for both macronuclear and micronuclear segments, we need to define how they align with each other. We define a scrambling system, which is composed of both a micronuclear and macronuclear gene.

Definition 3.2.6 *A scrambling system of strings s and r is a pair, $\delta = (\gamma, \beta)$, where we have $\gamma = (\overline{MDS}_\gamma, IES_\gamma, P_\gamma)$ which is a micronuclear classification of s and $\beta = (\overline{MDS}_\beta, P_\beta)$, which is a macronuclear classification of r. Intuitively, a scrambling system groups together a micronuclear and a macronuclear classification.*

Further, δ is valid *if both γ and β are valid, $\left|\overline{MDS}_\gamma\right| = \left|\overline{MDS}_\beta\right|$, and $|P_\gamma| = 2\,|P_\beta|$.*

A valid scrambling system partitions both the micronuclear and macronuclear segments into the same number of $\overline{MDS}$s. It does not align the various sections between the micronuclear gene

[1] *Here, "+" means the regular expression in brackets occurs one or more times.*

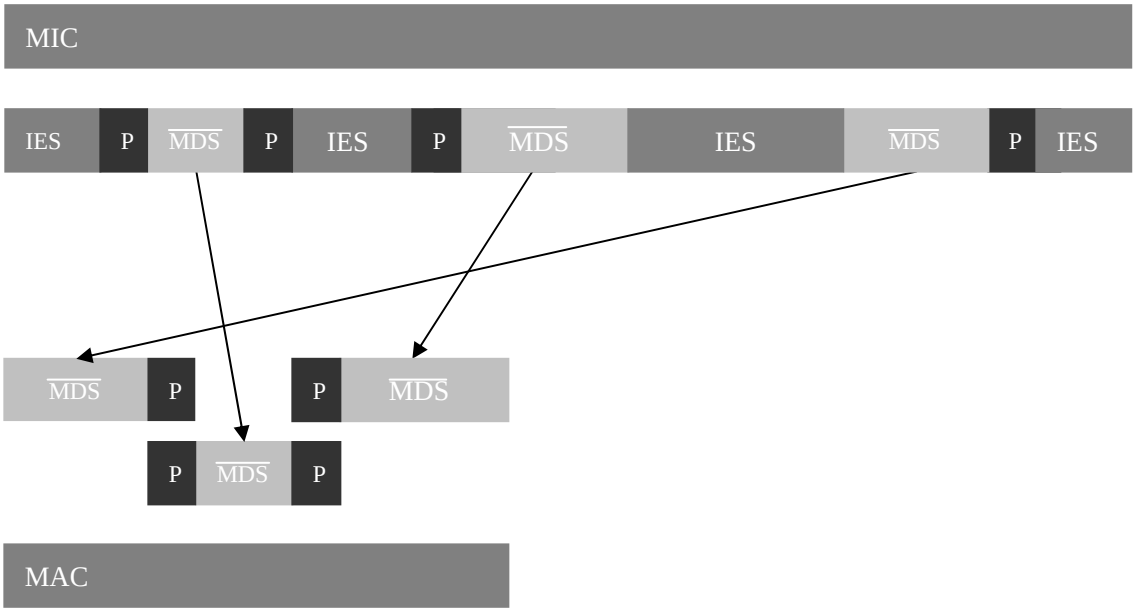

Figure 3.5: A labelling is a method to match the MDSs from the MIC onto the MAC, including the pointers.

and macronuclear gene. Indeed, we know that such an alignment exists but do not know how the segments align as yet.

3.3 Labelling a scrambling system

Biologically, MDSs from a MIC gene map onto those from a MAC gene. Thus, we need to describe how the various segments from a micronuclear and macronuclear gene relate to each other.

Given a pair of micronuclear and macronuclear gene segments, a *labelling* matches the various MDSs from the MIC onto the MAC. The arrows in Figure 3.5 represent matching MDS pairs, in which the two MDSs are presumably similar (gaps and mismatches are allowed).

The formal definition of a labelling is given below. In it, we use the concept of a bijective function which is any function where each element gets mapped to a distinct element, and every element in the codomain has an element mapped to it.

Definition 3.3.1 *Let $\delta = (\gamma, \beta)$ be a valid scrambling system of strings s and r, where $\gamma = (\overline{MDS}_\gamma, IES_\gamma, P_\gamma)$ and $\beta = (\overline{MDS}_\beta, P_\beta)$. Then a labelling of δ consists of a function f_δ from $seg_\gamma \cup start_\gamma \cup end_\gamma$ to $seg_\beta \cup start_\beta \cup end_\beta$, where f_δ is bijective, the element of $start_\gamma$ maps to the element of $start_\beta$, and the element of end_γ maps to the element of end_β.*

The labelled segment pairs over δ is expressed as

$$match(\delta) = \{(x, y, p, q) \mid f_\delta(x, y) = (p, q)\}.$$

Essentially, the function explains how segments of a micronuclear gene map onto segments of a macronuclear gene.

Now that we have formalized a labelling of a valid scrambling system, we can associate a score with each such labelling. This will allow to discuss the "best" labelling. Essentially it will be the sum of the global sequence alignment scores for each matching segment, minus a penalty for every $\overline{MDS}$. This penalty serves to give preference to longer $\overline{MDS}s$ over short $\overline{MDS}s$. For example, in a labelling, one could take any match between an $\overline{MDS}$ on the MIC with the $\overline{MDS}$ on the MAC, divide the segments into two, and the total sequence alignment score will be the same. As a more extreme example, we could only match segments of length 3 from the MIC onto the MAC. If the MIC is long enough, these are likely to occur exactly, and this situation would get the best score. However, we would prefer to use a smaller number of longer $\overline{MDS}s$ in this case, so the MDS penalty will serve this goal.

Moreover, the gap penalty and mismatch penalty serves to give the preference to multiple close matches with IESs over long matches with many gaps and mismatches. Those two penalties together with the MDS penalty enable to establish a balance between allowance of gaps and mismatches and the integrity of $\overline{MDS}s$.

The definition below provides a method of calculating a score of a labelling over a valid scrambling system.

Definition 3.3.2 *Let f_δ be a labelling of a valid scrambling system $\delta = (\gamma, \beta)$, where we have micronuclear and macronuclear classifications $\gamma = (\overline{MDS}_\gamma, IES_\gamma, P_\gamma)$, $\beta = (\overline{MDS}_\beta, P_\beta)$ of strings s and r. Let m be the match score, mm be the mismatch score, g be the gap penalty, and e be the MDS penalty. The score of a labelling f_δ, named Θ_{f_δ} is:*

$$\Theta_{f_\delta} = \Big(\sum_{(x,y,p,q)\in match(\delta)} GSA_{m,mm,g}(x,y,p,q) \Big) - e \cdot \left|\overline{MDS}_\beta\right|.$$

where $GSA_{m,mm,g}(x,y,p,q)$ is the maximum score of the global sequence alignment of $s[x \dots y]$ with $r[p \dots q]$, and $s[x \dots y]$ with $r[q \dots p]$, with match score m, mismatch score mm, and gap penalty g.

We took the maximum of these two scenarios as MDSs can be inverted (see Chapter 2). That is, one MDS can be similar to another in reverse.

Thus, each labelling of a valid scrambling system gives a global sequence alignment score between each pair of segments in $match_\delta$, and a total score of the entire alignment. Lastly, with the total score, we can attempt to find the best labelling of a valid scrambling system and its associated score.

Definition 3.3.3 *Let s, r be strings, m the match score, mm the mismatch score, g the gap penalty and e the MDS penalty. Then the optimal labelling score is*

$$max\{\Theta_{f_\delta} \mid \delta = (\gamma, \beta) \text{ is a valid scrambling system }, f_\delta \text{ is a labelling of } \delta\}.$$

An optimal labelling is a labelling f_δ and a valid scrambling system which gives this maximum.

Therefore, we are looking to find a labelling and a valid scrambling system which gives a score that is either optimal, or close to optimal.

Chapter 4

New algorithm to partition micronuclear genes and macronuclear genes

4.1 Motivation

As discussed in Chapter 2, the A-L Algorithm uses a small pipeline based on BLAST to detect MDSs, IESs, and pointers. But it is possible that the accuracy could be improved by using a modified version of the Smith-Waterman Algorithm to gain better control of gaps and mismatches. As the authors state, in some cases, if there is a region of low similarity between the MAC and MIC forms of a gene – probably due to varying alleles – the program breaks a proposed MDS into two smaller ones, and sometimes connects two MDSs that were annotated as separate. Considering the existence of mutations, every method should accept gaps and mismatches. As individual genes are usually quite small, we would rather use more sensitive and accurate alignment procedures at the expense of speed. Our new algorithm is based on this idea. However, its aim is not only to partition micronuclear genes into the MDSs, IESs, and pointers as the A-L algorithm does, but also to provide an alignment and a labelling of a scrambling system.

Furthermore, the A-L algorithm uses the traditional definition of MDS, which does not separate pointers from MDSs, while in this algorithm, the definition of $\overline{MDS}$ and the entire scrambling system and labelling is used. Three different criteria are used to evaluate the labelling, including the score of a labelling, Θ_{f_δ}.

4.2 Problem description

First, we would like to formalize the problem that we wish the algorithm to solve. Ideally, we would like to have a labelling of a valid scrambling system which achieves the highest possible score. Unfortunately, it was shown in our paper [12] (with Keil and McQuillan), that this problem is NP-Complete. Hence, the only algorithm that we know of which does this takes exponential time. Thus, we need to create an algorithm using heuristics in order to operate in polynomial time, although it may not always give the best possible answer.

As mentioned above, we could have part of a MAC sequence which does not match anywhere in the MIC, likely due to missing data. As a result, the scrambling system might not always be valid as the macronuclear segments might not match the micronuclear sequences. So we will not always provide a labelling of a valid scrambling system as output. When some part of the macronuclear segments does not match anywhere in the micronuclear segments, we do not consider these segments in the score. We call this a *partial labelling.* We describe the problem as follows:

- Problem: Create a algorithm to detect the $\overline{MDS}s$, IESs, and pointer segments within micronuclear and macronuclear genes, and provide the relationship between MDSs in the MIC and the MAC, as well as the alignment of matching MDSs.
- Description of the problem: Align a micronuclear gene s and a macronuclear gene r with the given match score, mismatch score and gap penalty.
- Inputs: Micronuclear gene, macronuclear gene, match score, mismatch score, gap penalty.
- Outputs: A partial labelling of a scrambling system.

The description above gives a definition of the problem. It provides something similar to a labelling as output.

4.3 Criteria to analyze the accuracy of the algorithms

In order to compare and analyze different algorithms, we will analyze different labellings based on three criteria, which reflect different perspectives of the problem and are not designed for any specific algorithms. Those criteria are total coverage, labelling score, and computational efficiency.

The coverage will give the percentage of the MAC that is labelled as $\overline{MDS}$ or pointer. When using real data, we could have MAC sequences which do not match anywhere in the MIC, likely due to partial sequence in the database. Coverage attempts to find the amount that this does not occur. By taking the percentage, we normalize the result into the range between 0 and 1. Formally the coverage is:

$$coverage = \frac{\sum_{(x,y)\in \overline{MDS} \bigcup P} |y-x|+1}{|r|}.$$

where r is the macronuclear string.

The algorithms are based on sequence alignment, so we will also consider the scores of labellings according to the given match score, mismatch penalty, gap penalty, and an additional MDS penalty. That is, we will calculate the score of the partial labelling of the determined scrambling system and use this as a measure of biological accuracy.

Also, we will briefly consider the program running time in order to discuss computational efficiency.

The algorithms should achieve acceptable values with all three criteria to be considered an acceptable algorithm.

4.4 Algorithm description

The new algorithm we create is based on the Smith-Waterman local sequence alignment algorithm, using dynamic programming as discussed in Chapter 2. To start, we set up a dynamic programming matrix aligning the micronuclear sequence s with the macronuclear sequence r. The algorithm includes a function which calculates the scores within this matrix. The values in the matrix indicate the local similarity of the MIC and MAC segments. The higher the value, the more similarity the segments share. According to the Smith-Waterman algorithm (Chapter 2), the aim of the function is to find the values that are the highest in the matrix in order to eventually trace back and detect the best alignments between the micronuclear and macronuclear gene.

Intuitively, we find the best local sequence between the macronuclear and micronuclear gene, and we call that section within both sequences a matching $\overline{MDS}$ pair. We then continue recursively with the remaining segments, which we store separately in a micronuclear segments array and a macronuclear segments array. In this way, the algorithm is greedy as we always take the best alignment, calling it an MDS, and continuing with the remaining sections.

We first define the subfunction which calculates local sequence alignment scores using dynamic programming, as in Algorithm 1 (Chapter 2). We will not only use it to calculate scores of the entire micronuclear and macronuclear sequences, but also subsequences of the micronuclear and macronuclear genes as well.

In the main function (Algorithm 2), the algorithm finds a local alignment with the highest value above a certain threshold. Inverted matches are also included and detected as possibilities. Although only the pseudocode of the algorithm is provided, the PERL code appears in Appendix A.

In the algorithm, match score, mismatch penalty, gap penalty, and cutoff score are all parameters which have influence on the results. The cutoff score needs to be big enough to avoid finding MDSs which are too small such as MDSs with less than 6 nucleotides. Similarly, the mismatch penalty and gap penalty needs to be big enough to avoid connecting two MDSs, and small enough to allow certain gaps and mismatches within MDSs.

As opposed to traditional sequence alignments, we are interested in both long and short MDSs. As a result, if the penalty scores are too big, it will not allow any gaps and mismatches within the small MDSs. On the other hand, if the penalty scores are too small, it will connect different

Algorithm 2: This is an algorithm to partition micronuclear genes into $\overline{MDSs}$, IESs, pointers, and macronuclear genes into $\overline{MDSs}$ and pointers. It provides a partial labelling of the micronuclear and macronuclear pair.

Input: match score m, mismatch score mm, and gap penalty g, percentage penalty pp, MIC sequence s, MAC sequence r, cut-off threshold X.

Output: a partial labelling of the micronuclear gene and the macronuclear gene

⋆ Put the micronuclear sequence into the micronuclear array and the macronuclear gene into the macronuclear array.

while *no more local alignments above score* X *can be found* **do**

⋆ Calculate the local alignment matrix M using Algorithm 1 and the modified recurrence below with each pair of micronuclear and macronuclear segments in the arrays to find the best alignment.

⋆ Calculate the local alignment matrix M with each pair of micronuclear and macronuclear segments in arrays with the inverted macronuclear sequence to find the best alignment.

⋆ Take the best score, remove both segments from the arrays, call each section a matching MDS, and put the two remaining sections (to the left and right of the MDS) of the micronuclear segments into the micronuclear array and similarly with the macronuclear array.

end

MDSs as a single MDS when the IESs between them are comparatively small. In order to solve this problem, a new penalty method is provided. Instead of giving the penalty score a permanent value, we define the allowance of gaps or mismatches within MDSs, as a *"percentage penalty"*. When calculating the mismatch score and gap scores, penalties will increase as the local alignment gets longer. For example, if the current alignment score is 100, the percentage penalty is 50%, and the mismatch penalty is -2, then the mismatch value is

$$\min\{100 - 100 \times 50\% = 50, 100 - 2 = 98\} = 50.$$

With this method, given a big enough percentage penalty, for example 50%, both the long and short MDSs would accept a similar number of mismatches or gaps.

So given the match score m, the mismatch penalty mm, the gap penalty g, and the percentage penalty pp, the formal expression of the algorithm to calculate the optimal local alignment for each prefix of s and r operates according to the following recurrence relation: $x_{(i,0)} = 0, \forall i \geqslant 0, x_{(0,j)} = 0, \forall j \geqslant 0$ and

$$x_{(i,j)} = \max \begin{cases} x_{(i-1,j-1)} + m, & \text{if } s_i = r_i, \\ min\left\{x_{(i-1,j-1)} + mm, x_{(i-1,j-1)} - x_{(i-1,j-1)} \times pp\right\}, & \text{if } s_i \neq r_i, \\ min\left\{x_{(i-1,j)} + g, x_{(i-1,j)} - x_{(i-1,j)} \times pp\right\}, & \\ min\left\{x_{(i,j-1)} + g, x_{(i,j-1)} - x_{(i,j-1)} \times pp\right\}, & \\ 0. & \end{cases}$$

Notice that the old recurrence coincides with this one with 0 as the percentage penalty.

4.5 Algorithm analysis

The time complexity for the subfunction (Algorithm 1) is $O(k \cdot l)$, where k and l are the lengths of the two sequences. In the main function, if in iteration i, there are x micronuclear segments in the micronuclear array, and y macronuclear segments in the macronuclear array, then in each iteration we calculate the local alignment of each micronuclear segment with each macronuclear segment. The entire computation of all these alignments can fit within the original $|s| \times |r|$ matrix. Moreover, there are at most $\max\{|s|, |r|\}$ such interactions since we add one MDS at each round. Thus, the entire time complexity is $O(|s| \cdot |r| \cdot \max\{|s|, |r|\})$.

In this case, the algorithm above is based on the intuition that the longer the matches which occur, the better chance it would be an MDS pair. Instead of using the pointers to guide the alignment, we would be more interested in whether the MDSs themselves are enough to create a scrambling system. Also, the modified recurrence can potentially achieve a better balance between gaps, mismatches, and splitting up MDSs.

4.6 Result analysis

Using the algorithm provided above, all 13 micronuclear gene and macronuclear gene pairs in IES_MDS_Database [2] have been tested. The three comparison criteria, which are coverage, labelling score, and program running time have been calculated. The labelling scores are calculated by the global sequence alignment algorithm with the parameters $m = 1, mm = -2, gap = -1$. The A-L algorithm does not give an alignment, but just annotates the MDSs and gives the correspondence between the micronuclear segments and macronuclear segments. The new algorithm developed outputs a formal scrambling system, which provides a partial labelling and also calculates the quality of the labelling. In order to compare the new algorithm with the A-L algorithm, we developed a PERL script to calculate the coverage and labelling score of the A-L algorithm. This script is attached in Appendix B.

Four cases are tested to analyze and compare the new algorithm and the A-L algorithm. We use cutoff scores of 5 and 10 respectively, together with percentage penalties of 50% and 0%. Most of the MDSs in the database are longer than 10 nucleotides, as our match score is 1, so we choose 10 as the cutoff score. However, we noticed that some of the MDSs are only 2 nucleotides when using the A-L algorithm. In order to compare, we also choose 5 as another cutoff score to test. It would be interesting to test a large set of possible percentage penalties, but due to time constraints, we only tested 50% and 0% percentage penalty for simplicity. When testing the algorithm in the case that no percentage penalty is used, the parameters used within local sequence alignment are $m = 1, mm = -25, g = -25$. Such large mismatch and gap scores are used because otherwise two large MDSs could be easily be connected even when separated by an IES. In the case that the percentage penalty is used (50%), standard sequence alignment parameters of $m = 1, mm = -2, g = -1$ are used instead.

In order to better compare the results, we give each micronuclear and macronuclear pair a numerical identifier, which is identical to the identifier used in the online ciliate database [2]. The relationship between the numerical identifier and the micronuclear–macronuclear gene pairs is shown in Table 4.1. Henceforth in the thesis, each MIC and MAC gene pair is represented by the number shown in the table.

The results will be discussed from four different perspectives: the number of MDSs detected, the coverage, the labelling score and the program running time. In the end, a summary and comparison between the results using the different parameters of the new algorithm and the A-L algorithm will be discussed.

	Organism	Gene Product
38	*Sterkiella nova*	Actin I
35	*Oxytricha fallax*	Actin I
1	*Sterkiella histriomuscorum*	Actin I
28	*Stylonychia pustulata*	Actin I
32	*Sterkiella histriomuscorum*	Alpha Telomere Binding Protein
42	*Uroleptus sp*	Alpha Telomere Binding Protein
41	*Paraurostyla weissei*	Alpha Telomere Binding Protein
37	*Sterkiella nova*	Alpha Telomere Binding Protein
33	*Stylonychia mytilus*	Alpha Telomere Binding Protein
44	*Uroleptus sp*	DNA Polymerase Alpha
19	*Sterkiella histriomuscorum*	DNA Polymerase Alpha
26	*Stylonychia lemnae*	DNA Polymerase Alpha
36	*Paraurostyla weissei*	DNA Polymerase Alpha

Table 4.1: The table above shows the numerical identifier of each macronuclear gene and micronuclear gene pair. The first four gene pairs are all the gene Actin I from different species, the next five entries with grey background are all Alpha Telomere Binding Protein, and the final four entries are all DNA Polymerase Alpha.

	MAC Length	MIC Length	A-L	p50c5	p50c10	p0c5	p0c10
38	1604	2374	9	12	9	12	9
35	1553	989	2	9	5	24	16
1	1558	2115	10	14	11	25	16
28	1528	1600	8	21	8	22	13
32	2166	2756	17	27	20	66	40
42	1961	3205	16	20	16	27	16
41	1859	1834	14	33	17	47	25
37	2217	2700	14	19	17	32	23
33	2141	2686	14	18	18	14	14
44	4916	5656	34	76	30	112	36
19	5007	6500	40	98	45	273	107
26	4986	6588	48	124	52	200	93
36	4746	6930	48	308	62	431	53

Table 4.2: The table shows the different number of MDSs detected, using both algorithms with the different parameters.

4.6.1 Analysis of the number of MDSs detected

The number of MDSs found with different parameters are shown in Table 4.2. We use $p50c5$ for the percentage penalty of 50%, and a cutoff score of 5. Similarly, $p50c10$ stands for the percentage penalty of 50%, and a cutoff score of 10; $p0c5$ stands for the cutoff score of 5 with no percentage penalty, and $p0c10$ stands for the cutoff score of 10 with no percentage penalty. Finally, A-L stands for the A-L algorithm (introduced in Chapter 2).

The data in Table 4.2 is graphed in Figure 4.1 to visualize the differences in the number of MDSs detected.

As Figure 4.1 shows, when using the parameters $p50c10$, the number of MDSs found are nearly identical to that for the A-L algorithm. When the cutoff score is 5, the new algorithm tends to find more MDSs. However, when using parameters $p0c10$ on gene pairs numbered 19 and 26, it tends to find a lot more MDSs than the A-L algorithm and new algorithm with parameter $p50c10$. This is likely because without using a percentage penalty, large gaps and mismatch penalties are needed, so some MDSs are considered to be as a result of mismatches or gaps rather than being a single MDS.

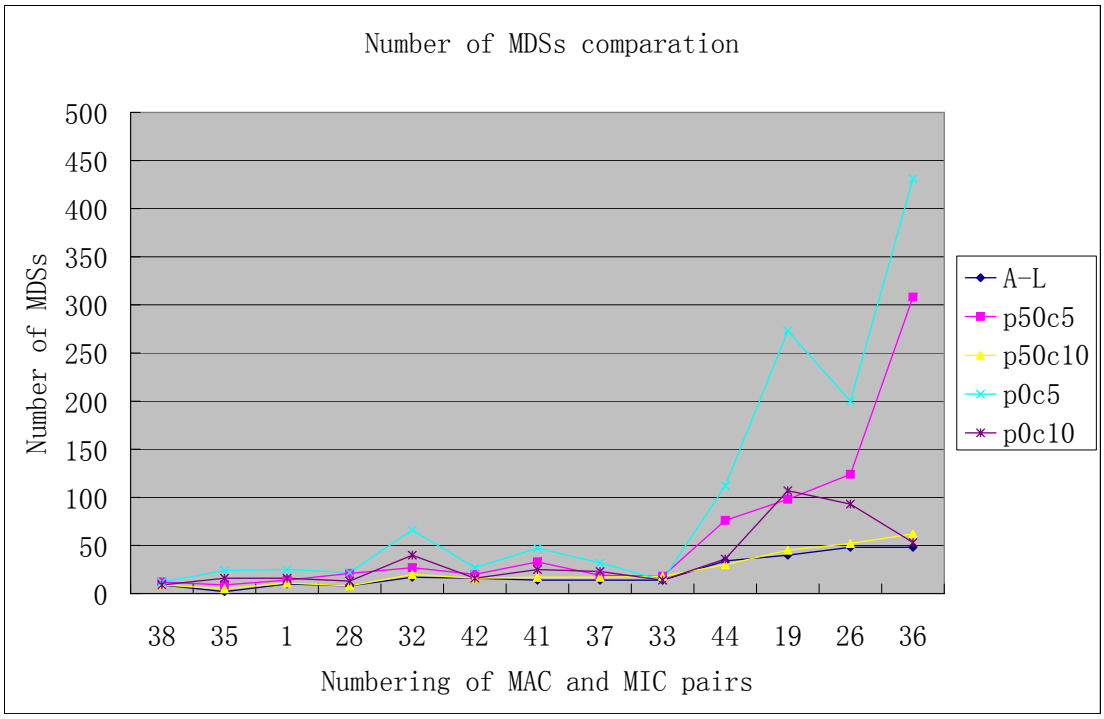

Figure 4.1: The figure shows the different number of MDSs detected by the various algorithms and parameters used. The numbers on the x-axis are the different numerical identifiers. The lines here are only used to show the trends, and are not defined between any two points on the x-axis.

	MAC Length	MIC Length	A-L	p50c5	p50c10	p0c5	p0c10
38	1604	2374	94.8	99.5	97.7	99.2	96.9
35	1553	989	59.4	63.9	60.9	61.8	57.4
1	1558	2115	95.8	98.9	96.3	97.5	91.3
28	1528	1600	79.8	88.4	77.9	84.4	76.7
32	2166	2756	94.1	99.1	95.2	94.2	83.1
42	1961	3205	93.4	98.9	96.5	98.9	93.4
41	1859	1834	78.9	91.4	81.6	88.7	79.2
37	2217	2700	96.2	99.6	98.7	99.1	95.2
33	2141	2686	96.7	99.1	99.1	99.2	99.2
44	4916	5656	94.8	98.1	85.7	95.4	79.1
19	5007	6500	91.4	98.3	83.8	93.0	60.3
26	4986	6588	98.0	97.8	79.8	94.6	71.0
36	4746	6930	96.8	91.4	32.2	87.1	15.1
average			90.0	94.2	83.5	91.8	76.8

Table 4.3: The table shows the coverage (%) of the macronuclear genes with different algorithms and parameters, as well as the average in the last row.

4.6.2 Coverage analysis

The different coverages are shown in Table 4.3, which we use to construct the graph in Figure 4.2.

As Figure 4.2 shows, the A-L algorithm, and the algorithm with the cutoff score of 5 reaches coverage above 80% in all the cases except for MIC and MAC pair number 35. However, the pair 35 has a longer macronuclear gene than the micronuclear gene which is the obvious result of missing data (the gene was only partially sequenced). Using a cutoff score of 10, some small MDSs might be excluded, when in reality, MDSs seem to be comparatively small in DNA Polymerase Alpha. Further, using a cutoff score of 5, and 50% percentage penalty, we achieve better coverage in most of the data pairs than the A-L algorithm. The average of the coverage percentages of $p50c5$ and $p0c5$ are both higher than the A-L algorithm, as seen in Table 4.3.

4.6.3 Labelling score analysis

The labelling scores are shown in Table 4.4. As one of the criteria used to analyze the accuracy of the algorithm, the labelling score calculates the global sequence alignment score of MDS pairs. As mentioned above, the A-L algorithm only identifies MDSs, and does not align them. Thus, in order to give the labelling score, we use global sequence alignment on the MDSs they identify. This

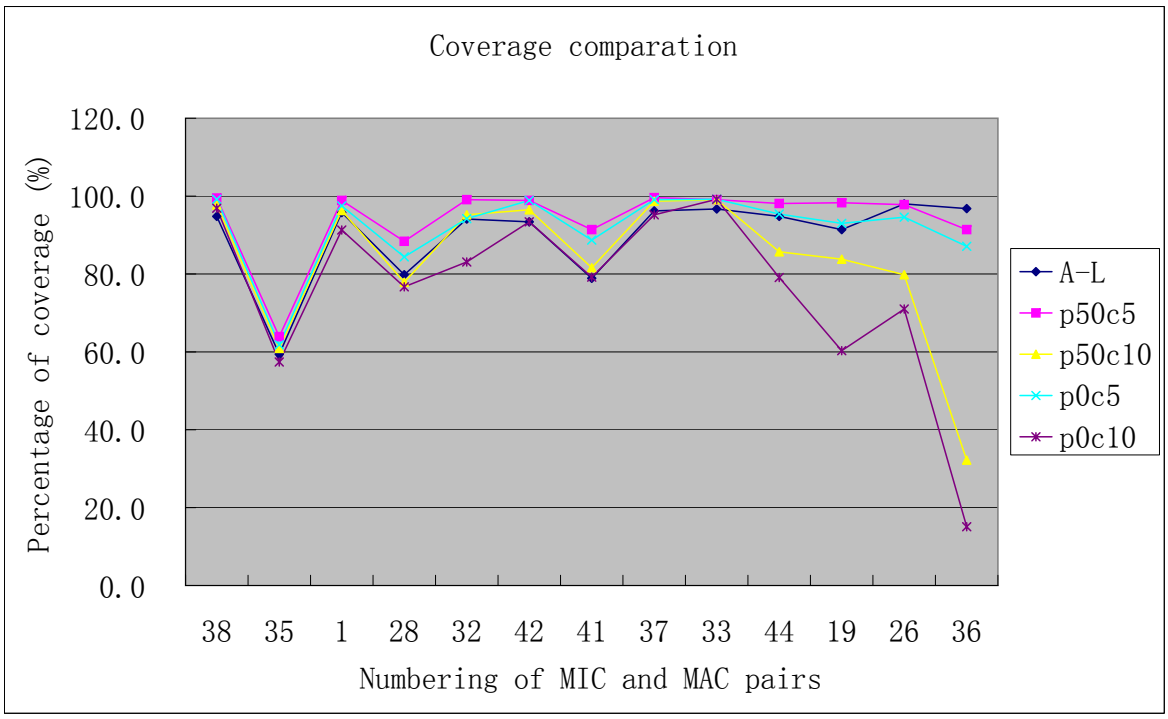

Figure 4.2: The figure above shows coverage using the different algorithms and parameters. The lines here are only used to show the trends, and are not defined between any two points on the x-axis.

	MAC Length	MIC Length	A-L	p50c5	p50c10	p0c5	p0c10
38	1604	2374	1424	1430	1424	1428	1422
35	1553	989	825	832	818	782	760
1	1558	2115	1311	1325	1305	1288	1252
28	1528	1600	1136	1168	1098	1135	1071
32	2166	2756	1708	1692	1671	1593	1521
42	1961	3205	1724	1734	1721	1730	1700
41	1859	1834	1321	1347	1306	1321	1279
37	2217	2700	1937	1931	1923	1900	1874
33	2141	2686	1941	1925	1925	1952	1952
44	4916	5656	3932	3963	3773	3827	3536
19	5007	6500	3347	3527	3233	2858	2365
26	4986	6588	3101	3544	3236	3330	2932
36	4746	6930	3918	1643	815	1467	471
average			2125	2005	1865	1893	1703
average without 36			1976	2035	1953	1929	1805

Table 4.4: The table shows labelling scores with different algorithms and parameters.

should give a better score than if we had used BLAST to construct the alignments.

The resulting graph is provided in Figure 4.3.

As Figure 4.3 shows, with the groups of Actin I and Alpha Telomere Binding Protein, providing 9 pairs out of 13 gene pairs, both the A-L algorithm and our new algorithm with any chosen parameters reach similar labelling scores. As the match score is 1, in the best case without mismatches and gaps, the ideal maximum score would be near to the length of the macronuclear gene but will not reach the value, because the MDS penalty is also calculated. Considering the last group of gene pairs (DNA Polymerase Alpha), when using the percentage penalty, the labelling score tends to be even better than the A-L algorithm. In fact, the new algorithm using the percentage penalty reaches higher labelling scores than the A-L algorithm in more cases except the last gene pair number 36. The average labelling score is 59 points higher using $p50c5$ than the A-L algorithm or any other parameters when not calculating the gene pair number 36. However, if we include gene 36, the A-L algorithm achieves the highest average.

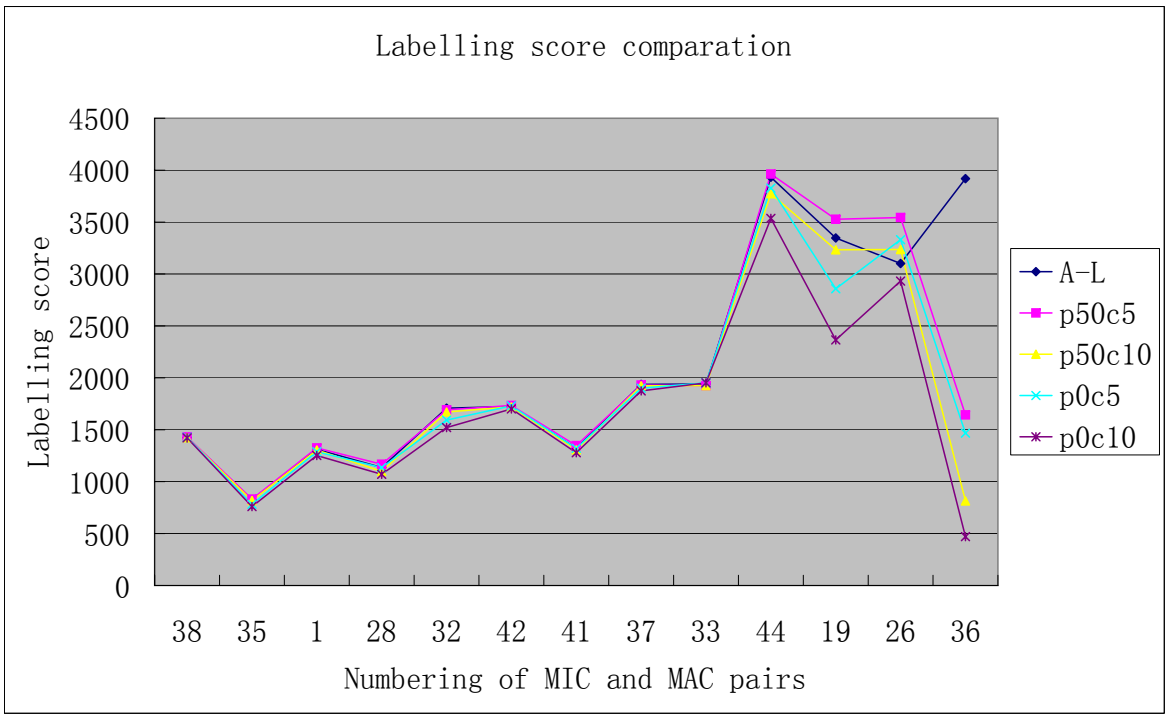

Figure 4.3: The figure above shows the labelling scores achieved by the different algorithms and parameters. The lines here are only used to show the trends, and are not defined between any two points on the x-axis.

	MAC Length	MIC Length	p50c5	p50c10	p0c5	p0c10
38	1604	2374	67	68	45	35
35	1553	989	29	29	49	25
1	1558	2115	80	80	63	47
28	1528	1600	58	50	49	36
32	2166	2756	253	253	347	261
42	1961	3205	166	165	122	95
41	1859	1834	127	121	123	92
37	2217	2700	133	134	113	90
33	2141	2686	176	177	73	62
44	4916	5656	2526	2377	2059	1402
19	5007	6500	4847	4520	12823	7782
26	4986	6588	4506	3808	6992	5656
36	4746	6930	28127	13037	36555	7824

Table 4.5: The table shows different program running times (seconds) with different parameters used.

4.6.4 Program running time analysis

The running time of the A-L algorithm which is executed in a web server is not comparable to our new algorithm on a Desktop PC. It can be noted that the running time for the A-L algorithm was nearly instantaneous, although we do not know the infrastructure of their server. However, we do expect that our new method, which is based on local sequence alignment, should be slower than BLAST. As the database is small and the length of micronuclear gene and macronuclear gene pairs are small, sensitivity rather than time is our primary goal with the new algorithm. The running times are shown in Table 4.5.

The data in Table 4.5 is graphed in Figure 4.4.

The program running times are calculated for Table 4.5 and Figure 4.4 on a single dual core 2.4 GHZ iMac computer with 2 GB of RAM.

4.6.5 Comparison between the algorithms and parameters

With only 13 gene pairs, it is impossible to make any general conclusions as to the best algorithm and parameters. However, on average, our algorithm with $p50c5$ achieved the highest average coverage, followed by $p0c5$, followed by A-L, with both versions using a cutoff score of 10 being significantly lower. The parameters $p50c5$ gave the highest coverage for 10 of 13 gene pairs, with

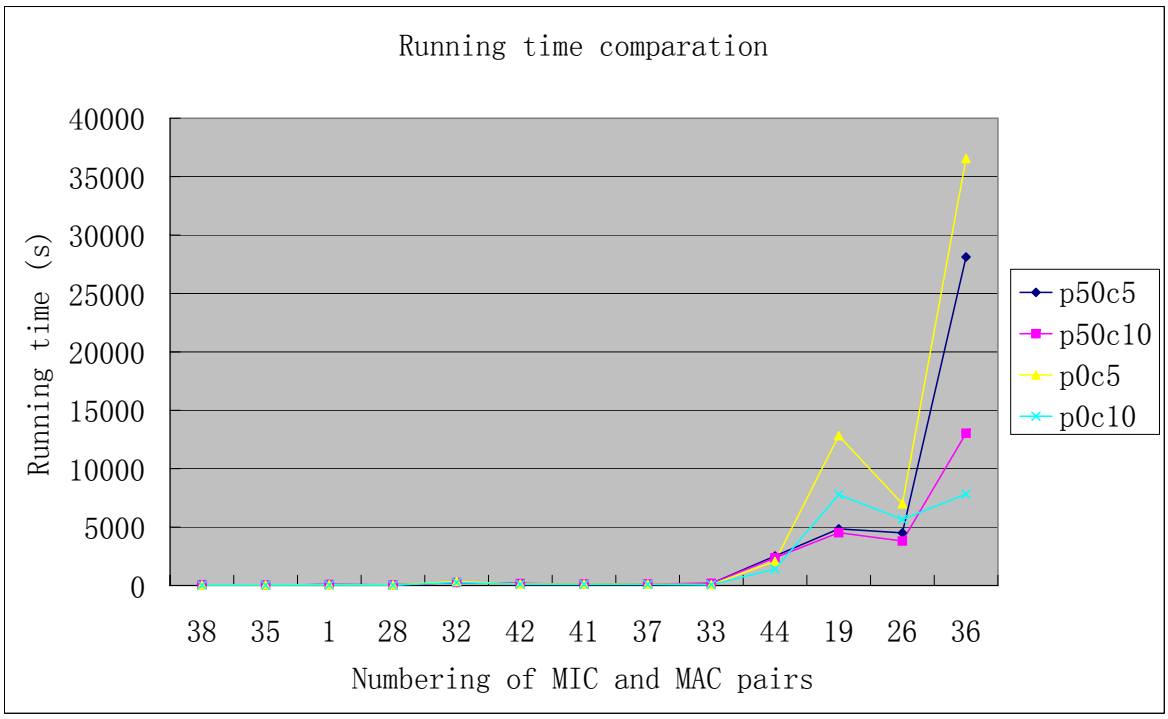

Figure 4.4: The figure shows the trend of different program running times. The lines here are only used to show the trends, and are not defined between any two points on the x-axis.

the A-L algorithm slightly higher score for two pairs, and both $p0c5$ and $p0c10$ being slightly higher on a single gene pair. On the other hand, with respect to the labelling score, $p50c5$ achieves highest average score when not considering data pair 36, and is the highest for 9 out of 13 gene pairs except for gene pair number $32, 37, 33, 36$. But the score is significantly lower than A-L algorithm when including that last gene pair, which could also be the result of the large MDS penalty together with the fact that it founds 260 more MDSs than the A-L algorithm on pair 36. When using $p50c10$, $p0c10$, and $p0c5$, the labelling score is lower, however $p0c5$ does achieve a higher labelling score for pairs $38, 42, 33, 26$.

Generally speaking, both the coverage and labelling score are better when using a percentage penalty. Also, when the cut off score is lower, more MDSs are detected and better scores could be achieved. However, when the cutoff score is too small, many MDSs are found of which many could be non-MDSs. In this particular case, we suspect that 10 is a better score than 5 when using a percentage penalty.

The last gene pair (data numbering 36) is the only case with a low labelling score and when using a percentage penalty. This could be because the real MDS size of this pair is too small. The labelling result shows that many mismatches and gaps occur. This could indicate that this could be improved by choosing a better set of parameters which is left for future research.

4.6.6 Special cases

The micronuclear gene copy of gene pair 26 contains both a major locus and a minor locus (two parts of the gene are located in different areas of the chromosome). In order to consider both parts of the micronuclear gene but not simply connect them, the minor locus is pushed into the micronuclear array after calculating the best match between the major locus of micronuclear gene and the macronuclear gene.

Gene pair 35 has a longer macronuclear gene than micronuclear gene, which is a case of missing data. As a result, the coverage on this pair cannot be expected to reach a high score for any algorithm.

Chapter 5

Genetic scrambling analysis based on biological hypotheses and a bioinformatics algorithm

5.1 Motivation

Instead of simply categorizing micronuclear gene components, the other goal of our study is to help understand scrambling biologically. In this chapter, bioinformatics algorithms are created and used to test hypotheses and help reveal some patterns of scrambling.

A model of the genetic behaviour of descrambling has been introduced in Chapter 2, attempting to address the structure of the micronuclear molecules. The authors hypothesize that the topological structure could be used to help align pointers. As discussed in Section 2.2.4, after a structure has been formed, there is a one dimensional process known as slithering. After recombination (and perhaps multiple recombinations in parallel) the molecule will either negatively or positively supercoil, potentially allowing the next structure to form. Thus, before this slithering occurs, the real pointers to be aligned will have the same distance between themselves as the distance between the two positions of the recombination site. If this hypothesis is true, then repeats would only be identified as being pointers if they are some fixed distance or distance range away from each other. That is, if some topological structure is formed, followed by slithering (Figure 5.1), then repeats would only be identified as pointers if they could slither so that the repeats were aligned. Hence, we can examine this hypothesis by evaluating repeats separated by various distances.

In Chapter 2, the model of template guided recombination was also discussed. According to this hypothesis, the pointer repeats along with the surrounding regions could be used to identify pointers within the micronucleus. Since we do not know what factors contribute, we would also like to try to determine what kinds of information is enough to perform descrambling.

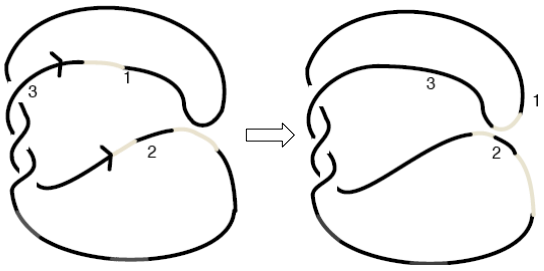

Figure 5.1: On the left is a picture of a hypothetical structure of a micronucleus before slithering, and on the right, after slithering. The numbers represent MDSs, while the IESs are in grey. Positions 1 and 2 and 3 along with the given distance between them, move together while the structure does not change. The arrows on the left part show the direction of slithering.

5.2 An algorithm suite based on the structure model

5.2.1 Definitions of basic terms used in the algorithm

In order to test the potential ability to descramble based on structure, we will build an algorithm that detects repeats (which may or may not represent real pointers) in a micronucleus, which are separated by the same distance as real pointers. Given different constraints, the less repeats that are present satisfying this criteria, other than real pointers, the more feasible it is for that given information to be important for descrambling.

The algorithm is based on pointers adjacent to MDSs. The distances between each pointer downstream of a given MDS and the pointer upstream of the next consecutive MDS (consecutive in terms of the MAC ordering) are collected. We calculate the potential real pointers using the predictions from the A-L algorithm. Here, we use the results from the A-L algorithm based on two reasons. First, the results of the A-L algorithm is currently the common method to annotate MDS, IES, and pointer regions. Second, our algorithm could not locate pointer regions in some of the cases and better parameter sets still need to be studied and tested. To better understand the algorithm suite, we need the following definition:

Definition 5.2.1 *A* window *is two sequences of nucleotides of the same length, starting at two positions of a string. A window is a* repeat *if both sequences are the same. A window is a* real pointer *if it is a matching pointer as determined by the A-L algorithm. To simplify the following discussion, the front segment window is called the* front window*, and the back segment window is called the* back window*. Here, the front is in the terms of the lower index of the micronuclear string. The* window distance *is the difference between the starting position of the two sequences of the window. The* window size *is the number of nucleotides of each sequence in the window. The*

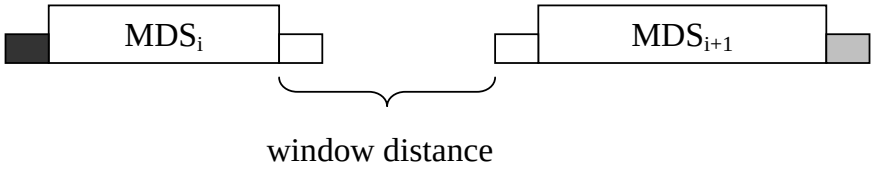

Figure 5.2: A window is any two subsequences of the same length on a string. For example, the two white pointers together form a window. The difference between the start of the second sequence and the start of the first sequence is the window distance (one more than the number of nucleotides between the two start positions). The number of nucleotides in both sequences of the window is the window size, and the nucleotides in both segments is the window sequence.

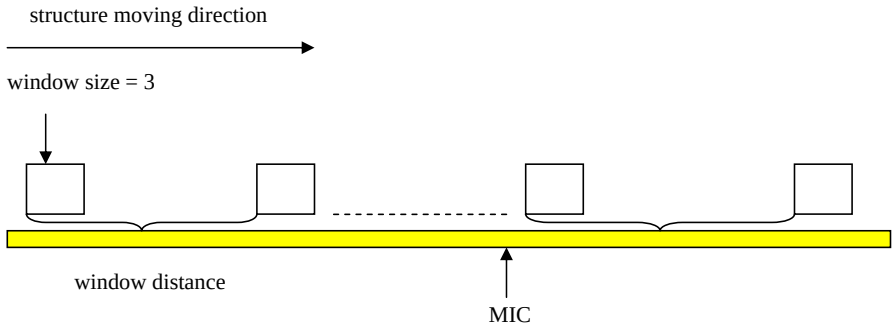

Figure 5.3: The figure above is the diagram describing the "slithering movement" of the window. The window, together with the window distance, moves from the beginning to the end of a micronuclear sequence, one nucleotide at a time.

window sequence *is the sequence within both segments of the window, where they are both the same.*

The definitions above give the necessary concepts used in the algorithms. We will try to test the hypothesis, which is based on the structure of the micronucleus. A *"sliding window"* refers to a window moving one position to the right iteratively (both sequences in parallel as Figure 5.3 shows).

5.2.2 Definition of the algorithm suite

We would like to test the feasibility of whether structure together with slithering is sufficient to descramble. As the biological machinery is unknown, we will instead test multiple possibilities at once by varying parameters. As such, we will consider six variations of the same algorithm which have slightly different assumptions.

Definition 5.2.2 *We will consider the following algorithm suite, the components of which are all derived from the same script by changing parameters.*

KnotD *is the algorithm that only uses the* window distance *factor. That is, it will attempt to find the locations of repeat windows of the same distance as real pointer windows, and determine the proximity of these windows to the real pointers.*

KnotS *is the algorithm that only uses the* window size *factor. That is, it attempts to find repeat windows of the same size as real pointer windows regardless of the distance.*

KnotSe *is the algorithm that only uses the* window sequence *factor. That is, it attempts to find windows of the same sequence as real pointer windows regardless of the distance.*

KnotDS *is the algorithm that uses both* window distance *and* window size *factors.*

KnotDSe *is the algorithm that uses both* window distance *and* window sequence *factors.*

KnotDSe3 *is the algorithm that uses both* window distance *and* window sequence *factors but only attempts to match the first three nucleotides of the front window and back window.*

The KnotD, KnotDS, KnotDSe, and KnotDSe3 all use the window distance factor. That is, each slides a window of the same window distance as the real pointers. Then, it calculates the proximity of each repeat to the real pointers, where the four variants will register a match if the repeat is any repeat at all (KnotD), if the repeat is the same size as the real pointer (KnotDS), if the repeat is the same sequence of the real pointer (KnotDSe), and if the repeat is the same sequence only considering the first three nucleotides of the real pointer (KnotDSe3).

In contrast, the variants KnotS and KnotSe do not need to have a window of the same distance as the real pointers, but indeed only need to be of the same size, and sequence respectively as the real pointers.

The algorithm suite will test each real pointer sequence pair for each algorithm variant. It will calculate the distance between the nearest qualified repeat and the real pointer position both from the left side upstream and the right side downstream. The former will give a repeat with lower index and the latter will give one with higher index. This gives the "range of slithering" that the sequence could move around each recombination site without encountering repeats other than the real pointers that match on the recombination site. This range will be called the *slithering range*. The range between the nearest repeat with lower, or higher index from the real pointer will be called the *left slithering range*, and the *right slithering range*, respectively.

5.3 Algorithm description

Here, we will give the description of the algorithm suite discussed above. Algorithm 3 describes the algorithm with variants KnotD, KnotDS, KnotDSe, and KnotDSe3. Algorithm 4 describes the algorithm with variants KnotS and KnotSe.

Algorithm 3: Algorithm with variants which consider the window distance.

Input: MIC sequence, variant V, list of real pointers and their positions and distances.

Output: repeat windows in the MIC with the same distance as the real pointer distance, and satisfying variant V.

while *0≤i≤ length of MIC − window size ×2− window distance* **do**
  if *the window, or the window with the reverse complement of the back window, satisfies variant V* **then**
    print the window information;
  end
  i=i+1;
  //slide the window
end

In Algorithm 3, the variant V for one of KnotD, KnotDS, KnotDSe, and KnotDSe3 are any repeats with the minimum window size of 3, any repeats with the same window size, any repeats with the same window sequence, any repeats with the same first three nucleotides of window sequence compared with real pointer sequence, respectively.

Figure 5.4 is a sample result of the KnotD and KnotDSe3 variants, which both use 3 as the minimum default window size. Here, on each line, we show the front window and its position of every repeat with the same distance as the real pointer. The start index of the front window is also the start position of the entire window. According to the result of the A-L algorithm, the first pointer of the micronuclear gene number 19 is "AGATA", the distance between real pointers is 50 nucleotides. As such, the nearest distances without other constraints are $2252 - 2212 = 40$, and $2282 - 2252 = 30$, on the left and right respectively for the constraint KnotD. Here, the repeat $2252, 2253, 2254$ are consecutive repeats representing a single pointer, so the distance will count from the first non-consecutive repeat. Similarly, for the constraint KnotDSe3, the first three nucleotides of the repeat is "AGA". As a result, the left slithering range is $2252 - 1608 = 644$ and the right slithering range is $3482 - 2252 = 1230$, which are the closest positions on the left and right where that exact sequence occurs.

Figure 5.5 represents the result with the KnotDS and KnotDSe constraints. Those 8 repeats are the only repeats that have the same window distance and window size of 5 as the real pointer in the entire micronuclear gene. So the left and right slithering ranges are $2252 - 945 = 1307$ and $3497 - 2252 = 1245$, respectively, for the KnotDS constraint. With this pointer, there is no other repeat in the micronuclear gene number 19 with the distance of 50 and a sequence of "AGATA". As a result, the length of the MIC, which is 6500, is given as the slithering range. When the slithering

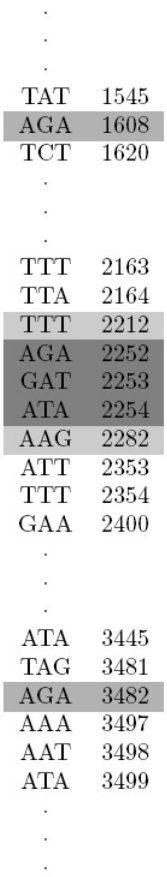

Figure 5.4: This is the result of the first pointer of micronuclear gene number 19 with Algorithm 3 using the minimum window size of 3. The repeat in dark grey is the real pointer, and repeats in lighter grey near the real pointer are the two nearest repeats according to the variant KnotD, left and right respectively. The two lighter grey repeats with the sequence of "AGA" farther away from the real pointer are the two nearest repeats according to the variant KnotDSe3.

range of a pointer is equal to the length of the micronuclear gene that the pointer belongs to under the constraint of either KnotD, KnotDS, KnotDSe, KnotDSe3, KnotS, or KnotSe, it means the pointer sequence is unique under that constraint.

In order to discuss whether the window distance could have influence on descrambling, we are also interested in finding repeats only under the constraint of window size or window sequence without considering the window distance. As a result, the algorithm under variant KnotS, and KnotSe is described in Algorithm 4.

AATAA	945
AGATA	2252
AAATA	3497
ATTTT	3876
AAAAA	4375
ATTAT	4862
TTATA	4863
AACAA	5924

Figure 5.5: This is the result of the first pointer of micronuclear gene number 19 with Algorithm 3 using the window size of 5 as is the real pointer size. The repeat in dark grey is the real pointer, and repeats in lighter grey near the real pointer are the two nearest repeats according to the variant KnotDS, left and right respectively. There is no repeat which satisfies the variant KnotDSe with the distance of 50.

Algorithm 4: Algorithm with variants which do not consider the window distance.

Input: window size, MIC sequence, variant V, list of real pointers, their sizes, and positions.

Output: repeats in the MIC with the same size or sequence as the real pointer size or sequence.

while *0≤i≤ length of MIC − size of window × 2* **do**
    j=0 ;
    while *0≤j≤ length of MIC − size of window* **do**
        if *the window, or the window with the reverse complement of the back window, satisfies variant V* **then**
            print out window information;
        end
        j=j+1;
    end
    i=i+1;
end

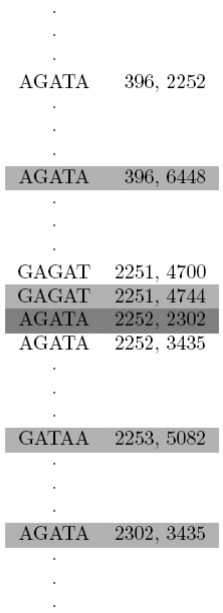

Figure 5.6: This is the result of the first pointer of micronuclear gene 19 with Algorithm 4, using the window size of 5 as is the real pointer size. The repeat in dark grey is the real pointer, and repeats in lighter grey near the real pointer are the two nearest repeats according to the variant KnotS, on the left and right respectively. The two lighter grey repeats with the sequence of "AGATA" further away from the real pointer are the two nearest repeats according to the variant KnotDSe.

As Algorithm 4 describes, the variant V for KnotS would check that the window size is the same as the real pointer size, and the variant V for KnotSe would check that the window sequence is the same as the real pointer sequence. Figure 5.6 shows the result with the variant KnotS and KnotSe. Here, the first element of each line stands for the matching repeat sequence. The second element is the start index of the front window, and the third element is the start index of the back window. As a result, the left and right slithering range for constraint KnotS are both 1, the left slithering range for variant KnotSe is $2252 - 396 = 1856$, and the right slithering range is $2302 - 2252 = 50$ according to the index of the front window. However, we also need to calculate the slithering range of the back window (which does not move in tandem with the front window). The minimum slithering range is the minimum range of both the back window and the front window. As a result, we could sort the resulting list using the order of the back window. In that case, we could find the minimum left slithering range for the back window is $2302 - 2252 = 50$, with the nearest repeat window index of "396, 2252". The right slithering range of the back window is 1133, corresponding to the repeat index of "396, 3435". So the left slithering range for this pointer under constraint KnotSe is $\min\{1856, 50\} = 50$, and the right slithering range is $\min\{50, 1133\} = 50$.

	Number of pointers	number of reverse pointers	minimum pointer size	median pointer size
38	8	2	4	11
35	1	0	4	4
1	9	2	4	10
28	4	0	4	9
32	16	0	3	7
42	14	0	5	8
41	15	0	2	5
37	13	0	3	9
33	13	0	3	7
44	32	30	2	8
19	30	25	2	8
26	45	26	2	8
36	47	40	2	7
group 1	22	4	4	10
group 2	71	0	2	7
group 3	154	121	2	8
total	247	125	2	8

Table 5.1: The table shows a statistical summary of the data collected. In the first column, we have the identifier from Table 4.1. Group 1, 2, and 3 refers to *Actin I*, *Alpha Telomere Binding Protein*, and *DNA Polymerase Alpha* respectively, and the total represents pointers in all 13 genes. The second column represents the number of pointers in each gene or group. The third column indicates the number of reversed pointers. The last two columns signifies the minimum and median pointer size of each gene or group respectively.

The PERL code of the algorithms provided above is attached in Appendix C, and the left and right slithering range for all 247 available pointers of 13 micronuclear gene are calculated and listed in Appendix D with each variant of KnotD, KnotDS, KnotDSe, KnotDSe3, KnotS, and KnotSe.

5.4 Result analysis

Using the algorithm provided above, all 13 micronuclear gene and macronuclear gene pairs have been tested. The slithering range both from the left side and right side of all 247 pointer sequences have been calculated with the algorithm suite of KnotDS, KnotDSe, KnotD, KnotDSe3, KnotS, and KnotSe.

Table 5.1 shows the quantity of data, and certain statistical attributes of the pointer sequences. It includes the number of reverse pointers, the minimum pointer size, and the median pointer size. In the table, the first column of the table indicates the numbering of the MIC sequence according to Table 4.1. Henceforth, we will use this convention.

From the table we can see that 50.6% of the pointer sequences are reversed and 121 of them come from group 3, and 4 comes from group 1. Although the minimum size of the pointer could be 2, the median size of all the pointer sequences is 8.

Table 5.2, 5.3, and 5.4 shows the result of the algorithm suite on micronuclear gene number 19. The area with grey background represents that the real pointer sequence is unique within the entire micronuclear gene based on the specific variant. The rows with a blue background signifies that the real pointer sequence according to the A-L algorithm includes mismatches or gaps. In this case, we will calculate and use the longest exact match portion of the pointer sequence. The area with a red background signifies that the pointer sequence pair of this row is a reverse complement match. The one cell with a pink background indicates that the two MDS sequences that the pointers belong to are all inverted, so the two pointer sequences are matched in the normal order but the first element of the result will show the complementary sequence.

We collect the same information for all 13 genes in Appendix D. The same layout and colour are used in the tables of Appendix D.

In order to better compare and analyze the results, we will discuss it from the following three perspectives: the percentage of unique sequences, the minimum slithering range and the median slithering range.

Table 5.5 shows the percentage of unique sequences in each micronuclear gene, each group, and the total, for each variant. Here, "unique" means that the real pointer does not have any other repeats which satisfy the appropriate condition, to the left or right within that gene. This means that the only repeat satisfying the condition is the real pointer given the specific micronuclear gene and the parameter (KnotD, KnotDS, KnotDSe, KnotDSe3, KnotS, or KnotSe.). With the strongest constraint, KnotDSe always gets the highest value, 92.7% in total, and 100.0%, 90.1%, and 92.9% in each group on the left sides, 94.3% in total, and 100.0%, 90.1%, and 95.5% in each group on the right sides. The parameters with the next highest total scores are from KnotDS, KnotSe, KnotDSe3, KnotS to KnotD from the highest to lowest. The last two parameters are significantly lower compared with the others. Although the slithering range of the pointer sequences which are not unique is not listed in this table, the higher the percentage, the more chance there is for the parameters to have influence on descrambling. Note from the data above, the percentage from the left side and right side are slightly different but none of them have an obvious advantage. Generally speaking, the values are higher with the first group than the other two groups with all the tested micronuclear genes, and using both sides of slithering.

Table 5.6 shows the minimum slithering range of each micronuclear gene, each group, and the total with each parameter, on both sides. From the table we can see the minimum slithering range of the entire data. Groups 2 and 3 are smaller, with the result of group 1 with KnotDSe, KnotDS, and KnotSe being significantly better. This is because the minimum pointer size of group 2 and group 3 are both 2 (Table 5.1). Almost all the small slithering ranges come from the pointer size of 2 or 3. Notice that the median pointer size is 8, so the median slithering range is also of interest.

Table 5.7 shows the median slithering range of each micronuclear gene, each group, and the total across all genes with each parameter on both sides. The cells with a darker grey background represent the fact that the range is the length of the entire micronuclear gene, which means the median slithering range for that micronuclear gene is infinite. Here, when calculating the median slithering range for groups, or the total, even if the number has a grey background, it may not mean the median slithering range for the entire group is infinite because the length of different micronuclear genes is different, varying from 989 to 6930. However, it could still give us a sense of the influence of each variant on descrambling.

In the table, we can see that all the cells for the constraint KnotDSe have a darker grey background which means the median slithering range for variant KnotDSe from both left and right sides in each gene and the entire group are the lengths of the corresponding MACs. Then, if we were to rank the constraints from strongest to weakest in terms of that factor being enough to guide descrambling, we get KnotDSe, KnotDS, KnotSe, KnotDSe3, KnotD and KnotS. Comparing KnotS and KnotDS, and also KnotSe and KnotDSe, the window distance constraint significantly enlarges the slithering range of the pointers around the recombination site. From the result of the KnotD and KnotDS constraints, we can see that the window size constraint also gives a significant increase. Comparing KnotS and KnotSe, knowing the window sequence, and not only the size has a significant affect to eliminate fake repeats. However, comparing KnotDS, and KnotDSe, we do have a larger slithering range for KnotDSe, but the values for KnotDS are already quite large. It is unknown what values are sufficient biologically. From the last variant KnotDSe3 listed in the table, we can see that only using the distance between the real pointers together with the first 3 nucleotides of the real pointer sequence would also be a strong constraint with the minimum of the median slithering ranges to be 583 nucleotides.

		position				KnotDS		KnotDSe	
size	pointer	p1	p2	distance	reverse	left	right	left	right
5	agata	2252	2302	50	n	1307	1245	6500	6500
4	atta	2903	3113	210	n	595	437	6500	6500
13	atgagtggaatta	1859	3407	1548	y	6500	6500	6500	6500
4	aaca	1847	3453	1606	y	127	109	6500	6500
9	agaaatatg	1813	3482	1669	y	6500	6500	6500	6500
8	ttatcatt	1745	3515	1770	y	6500	2970	6500	6500
8	aaaataat	1718	3546	1828	y	6500	6500	6500	6500
8	gtttcttg	1656	3569	1913	y	6500	6500	6500	6500
7	atgcaaa	1624	3596	1972	y	6500	568	6500	6500
8	taaaatga	1597	3624	2027	y	6500	6500	6500	6500
7	agaggag	1573	3643	2070	y	6500	6500	6500	6500
10	taatgatgga	1524	3677	2153	y	6500	6500	6500	6500
8	atggtgag	1489	3783	2294	y	905	6500	6500	6500
8	aaaatcaa	1469	3810	2341	y	6500	6500	6500	6500
12	aaagcatgcttg	1440	3892	2452	y	6500	6500	6500	6500
7	aagaaaa	1356	3931	2575	y	6500	6500	6500	6500
10	gttactcttg	1316	3985	2669	y	6500	6500	6500	6500
12	agctcaataaaa	1240	4030	2790	y	6500	6500	6500	6500
3	atc	1196	4063	2867	y	8	6	6500	1134
7	aaaactt	1111	4089	2978	y	6500	6500	6500	6500
10	gagagataga	1077	4173	3096	y	6500	6500	6500	6500
9	tagttgctc	1020	4220	3200	y	6500	6500	6500	6500
8	aagctaga	980	4339	3359	y	6500	6500	6500	6500
8	ggaggatc	953	4393	3440	y	6500	6500	6500	6500
8	caagataa	935	4412	3477	y	6500	6500	6500	6500
15	ataagactttgatga	803	4463	3660	y	6500	6500	6500	6500
8	ctaatgaa	191	250	59	n	6500	459	6500	6500
2	ag	0	4541	4541	y	6500	13	6500	82
6	ataaaa	6119	6194	75	n	113	6500	6500	6500
5	tatat	6352	6364	12	n	327	6500	6500	6500

Table 5.2: The table above shows an example of the algorithm suite with parameter KnotDS and KnotDSe on micronuclear gene number 19. The first column indicates the size of the pointer, the second column represents the pointer sequence. The third and fourth columns are the start positions, in terms of the index of the micronuclear gene string, of the front and back windows of the real pointers respectively. The fifth column is the distance between the two start positions. The sixth column indicates whether the pointers are reverse matched. We use "y" for a reverse complement match and "n" for the match with the ordinary order. The next two columns show the minimum number of nucleotides to the left (upstream) of the real pointers, and the right (downstream) of the real pointers. That is, it is the minimum number of nucleotides of the real pointer in which there are no other matching repeats satisfying the variant KnotDS. Similarly, the last two columns show the minimum number of nucleotides with the condition KnotDSe.

		position				knotD		knotDSe3	
size	pointer	p1	p2	distance	reverse	left	right	left	right
5	agata	2252	2302	50	n	40	30	644	1230
4	atta	2903	3113	210	n	231	11	760	598
13	atgagtggaatta	1859	3407	1548	y	27	27	1148	98
4	aaca	1847	3453	1606	y	3	92	674	6500
9	agaaatatg	1813	3482	1669	y	40	13	1472	1325
8	ttatcatt	1745	3515	1770	y	9	4	518	496
8	aaaataat	1718	3546	1828	y	34	40	202	425
8	gtttcttg	1656	3569	1913	y	12	29	6500	2746
7	atgcaaa	1624	3596	1972	y	34	47	633	568
8	taaaatga	1597	3624	2027	y	10	40	733	6500
7	agaggag	1573	3643	2070	y	15	20	6500	6500
10	taatgatgga	1524	3677	2153	y	100	19	6500	6500
8	atggtgag	1489	3783	2294	y	264	34	1377	606
8	aaaatcaa	1469	3810	2341	y	15	112	77	156
12	aaagcatgcttg	1440	3892	2452	y	36	33	6500	360
7	aagaaaa	1356	3931	2575	y	22	36	770	438
10	gttactcttg	1316	3985	2669	y	37	64	6500	6500
12	agctcaataaaa	1240	4030	2790	y	13	86	763	1021
3	atc	1196	4063	2867	y	8	6	6500	1134
7	aaaactt	1111	4089	2978	y	10	37	6500	1190
10	gagagataga	1077	4173	3096	y	61	20	199	775
9	tagttgctc	1020	4220	3200	y	18	5	6500	6500
8	aagctaga	980	4339	3359	y	48	8	168	442
8	ggaggatc	953	4393	3440	y	46	135	6500	6500
8	caagataa	935	4412	3477	y	55	62	55	521
15	ataagactttgatga	803	4463	3660	y	56	58	6500	794
8	ctaatgaa	191	250	59	n	6	69	6500	365
2	ag	0	4541	4541	y	6500	13	6500	82
6	ataaaa	6119	6194	75	n	75	13	75	159
5	tatat	6352	6364	12	n	28	14	1258	14

Table 5.3: The table above shows an example of the algorithm suite with parameter KnotD and KnotDSe3 on micronuclear gene number 19. The first column indicates the size of the pointer, the second column represents the pointer sequence. The third and fourth columns are the start positions, in terms of the index of the micronuclear gene string, of the front and back windows of the real pointers respectively. The fifth column is the distance between the two start positions. The sixth column indicates whether the pointers are reverse matched. We use "y" for a reverse complement match and "n" for the match with the ordinary order. The next two columns show the minimum number of nucleotides to the left (upstream) of the real pointers, and the right (downstream) of the real pointers. That is, it is the minimum number of nucleotides of the real pointer in which there are no other matching repeats satisfying the variant KnotD. Similarly, the last two columns show the minimum number of nucleotides with the condition KnotDSe3.

		position				KnotS		KnotSe	
size	pointer	p1	p2	distance	reverse	left	right	left	right
5	agata	2252	2302	50	n	1	1	50	50
4	atta	2903	3113	210	n	1	1	42	6
13	atgagtggaatta	1859	3407	1548	y	120	967	6500	6500
4	aaca	1847	3453	1606	y	1	1	5	92
9	agaaatatg	1813	3482	1669	y	3	8	6500	6500
8	ttatcatt	1745	3515	1770	y	1	7	6500	6500
8	aaaataat	1718	3546	1828	y	1	1	1494	648
8	gtttcttg	1656	3569	1913	y	4	6	6500	6500
7	atgcaaa	1624	3596	1972	y	1	1	6500	6500
8	taaaatga	1597	3624	2027	y	1	1	6500	1489
7	agaggag	1573	3643	2070	y	2	1	6500	6500
10	taatgatgga	1524	3677	2153	y	9	16	6500	6500
8	atggtgag	1489	3783	2294	y	2	5	6500	6500
8	aaaatcaa	1469	3810	2341	y	3	1	414	6500
12	aaagcatgcttg	1440	3892	2452	y	200	138	6500	6500
7	aagaaaa	1356	3931	2575	y	1	1	242	854
10	gttactcttg	1316	3985	2669	y	56	18	6500	6500
12	agctcaataaaa	1240	4030	2790	y	138	87	6500	6500
3	atc	1196	4063	2867	y	1	1	3	28
7	aaaactt	1111	4089	2978	y	2	2	633	6500
10	gagagataga	1077	4173	3096	y	3	17	6500	6500
9	tagttgctc	1020	4220	3200	y	12	20	6500	6500
8	aagctaga	980	4339	3359	y	1	1	6500	6500
8	ggaggatc	953	4393	3440	y	9	6	6500	6500
8	caagataa	935	4412	3477	y	13	3	6500	6500
15	ataagactttgatga	803	4463	3660	y	6500	445	6500	6500
8	ctaatgaa	191	250	59	n	9	6	6500	6500
2	ag	0	4541	4541	y	1	1	31	14
6	ataaaa	6119	6194	75	n	1	1	75	6500
5	tatat	6352	6364	12	n	1	1	12	12

Table 5.4: The table above shows an example of the algorithm suite with parameter KnotS and KnotSe on micronuclear gene number 19. The first column indicates the size of the pointer, the second column represents the pointer sequence. The third and fourth columns are the start positions, in terms of the index of the micronuclear gene string, of the front and back windows of the real pointers respectively. The fifth column is the distance between the two start positions. The sixth column indicates whether the pointers are reverse matched. We use "y" for a reverse complement match and "n" for the match with the ordinary order. The next two columns show the minimum number of nucleotides to the left (upstream) of the real pointers, and the right (downstream) of the real pointers. That is, it is the minimum number of nucleotides from the real pointer in which there are no other matching repeats satisfying the variant KnotS. Similarly, the last two columns show the minimum number of nucleotides with the condition KnotSe.

	KnotS		KnotSe		KnotDS		KnotDSe		KnotD		KnotDSe3	
	left	right	left	right	left	right	left	right	left	right	left	right
38	37.5	50.0	62.5	62.5	62.5	87.5	100.0	100.0	0.0	12.5	37.5	87.5
35	0.0	0.0	0.0	0.0	100.0	0.0	100.0	100.0	0.0	0.0	100.0	100.0
1	22.2	11.1	88.9	88.9	100.0	88.9	100.0	100.0	11.1	22.2	66.7	55.6
28	25.0	25.0	75.0	75.0	75.0	75.0	100.0	100.0	0.0	0.0	100.0	50.0
32	0.0	0.0	68.8	50.0	81.3	68.8	93.8	93.8	0.0	0.0	68.8	43.8
42	0.0	0.0	92.9	85.7	92.9	78.6	100.0	100.0	0.0	0.0	64.3	42.9
41	6.7	6.7	46.7	40.0	60.0	46.7	86.7	86.7	6.7	0.0	66.7	66.7
37	23.1	7.7	69.2	61.5	69.2	61.5	92.3	92.3	0.0	0.0	38.5	61.5
33	7.7	7.7	76.9	76.9	84.6	76.9	76.9	76.9	0.0	0.0	76.9	46.2
44	3.1	3.1	71.9	56.3	68.8	65.6	90.6	96.9	3.1	0.0	46.9	40.6
19	3.3	0.0	63.3	70.0	76.7	73.3	100.0	93.3	3.3	0.0	40.0	23.3
26	2.2	2.2	55.6	57.8	68.9	66.7	93.3	97.8	0.0	2.2	46.7	48.9
36	0	0	48.9	38.3	68.1	53.2	89.4	93.6	6.4	0.0	42.6	27.7
group 1	27.3	27.3	72.7	72.7	81.8	81.8	100.0	100.0	4.5	13.6	63.6	68.2
group 2	7.0	4.2	70.4	62.0	77.5	66.2	90.1	90.1	1.4	0.0	63.4	52.1
group 3	1.9	1.3	58.4	53.9	70.1	63.6	92.9	95.5	3.2	0.6	44.2	35.7
total	5.7	4.5	63.2	57.9	73.3	66.0	92.7	94.3	2.8	1.6	51.4	43.3

Table 5.5: The table above shows the percentage of unique sequences in each micronuclear gene, each group, and the total 13 genes with each parameter. The first column of the table indicates the numbering of the MIC sequence according to Table 4.1, in addition to group 1, group 2, group 3 and the total. We show the percentage to the left (upstream) of the real pointers, and to the right (downstream) of the real pointers. That is, it is the percentage of real pointers in which there are no other matching repeats satisfying the varying conditions.

	KnotS		KnotSe		KnotDS		KnotDSe		KnotD		KnotDSe3	
	left	right	left	right	left	right	left	right	left	right	left	right
38	1	1	32	11	116	75	2374	2374	4	14	25	452
35	1	1	65	28	989	293	989	989	70	26	989	989
1	1	1	68	28	2115	454	2115	2115	7	9	77	59
28	1	1	21	92	56	216	1600	1600	4	13	1600	175
32	1	1	12	12	11	23	507	83	4	13	399	40
42	1	1	394	484	379	1	3205	3205	4	10	168	23
41	1	1	1	2	9	4	65	4	4	4	58	4
37	1	1	37	4	117	5	667	213	4	5	112	115
33	1	1	3	13	57	34	223	117	4	14	223	46
44	1	1	2	9	6	24	8	24	3	8	8	24
19	1	1	3	6	8	6	6500	82	3	4	55	14
26	1	1	9	14	4	34	131	223	4	9	91	29
36	1	1	2	3	5	33	14	38	4	4	14	14
group 1	1	1	21	11	56	75	989	989	4	9	25	59
group 2	1	1	1	2	9	1	65	4	4	4	58	4
group 3	1	1	2	3	4	6	8	24	3	4	8	14
total minimum	1	1	1	2	4	1	8	4	3	4	8	4

Table 5.6: The table above shows the minimum slithering range of each micronuclear gene, each group, and the total 13 genes with each parameter. The first column of the table indicates the numbering of the MIC sequence according to Table 4.1, as well as group 1, group 2, group 3, and the total. We show the minimum number of nucleotides to the left (upstream) of the real pointers, and the right (downstream) of the real pointers. That is, it is minimum number of nucleotides from real pointers within which there are no other matching repeats satisfying the varying conditions.

	KnotS		KnotSe		KnotDS		KnotDSe		KnotD		KnotDSe3	
	left	right	left	right	left	right	left	right	left	right	left	right
38	195	1261	2374	2374	2374	2374	2374	2374	30	34	1089	2374
35	1	1	65	28	989	293	989	989	70	26	989	989
1	48	26	2115	2115	2115	2115	2115	2115	66	37	2115	2115
28	26	32	1600	1600	1600	1600	1600	1600	93	78	1600	1027
32	2	3	2756	1712	2756	2756	2756	2756	38	50	2756	1026
42	8	6	3205	3205	3205	3205	3205	3205	13	63	3205	1700
41	1	1	522	230	1834	750	1834	1834	60	47	1834	1834
37	19	2	2700	2700	2700	2700	2700	2700	50	46	583	2700
33	4	14	2686	2686	2686	2686	2686	2686	64	61	2686	594
44	2	3	5656	5656	5656	5656	5656	5656	17	27	1127	769
19	2	3	6500	6500	6500	6500	6500	6500	34	32	1203	691
26	2	1	6588	6588	6588	6588	6588	6588	25	34	3233	2965
36	1	1	5480	1359	6930	6930	6930	6930	39	26	964	575
group 1	29	32	2115	2115	2115	2115	2115	2115	61	38	1600	2115
group 2	4	4	2686	2686	2686	2686	2700	2700	48	52	2686	1834
group 3	1	2	5656	5656	6500	6078	6588	6588	27	29	1451	874
total	3	2	2700	2402	2756	2700	5656	5656	34	34	1834	1134

Table 5.7: The table above shows the median slithering range of each micronuclear gene, each group, and the total 13 genes with each parameter. The first column of the table indicates the numbering of the MIC sequence according to Table 4.1, as well as group 1, group 2, group 3, and the total. We show the median number of nucleotides to the left (upstream) of the real pointers, and the right (downstream) of the real pointers. That is, it is median number of nucleotides from real pointers within which there are no other matching repeats satisfying the varying conditions.

Chapter 6

Conclusion

In this thesis, we presented a background of ciliate scrambling and preliminary works introduced from both a biological and bioinformatics perspectives in Chapter 2. We discussed the only existing algorithm to detect MDS, IES and pointer segments. We briefly reviewed various biological hypotheses with particular interest to the DNA structure hypothesis.

A formal definition of ciliate scrambling systems was given in Chapter 3, and the labelling of a scrambling system was defined. This formalization of a scrambling system presents a way of describing algorithms in future research. Furthermore, it provides a basis of comparison between different algorithms, much like alignments are traditionally defined for sequence alignment. Labelling a scrambling system is quite a bit more complicated than traditional sequence alignment as we cannot simply align the micronuclear genes with macronuclear genes from left to right, but instead have to make sure that we partition each segment and then map them onto each other. Much like traditional sequence alignment seeks to find the best alignment according to a scoring scheme, we can attempt to find the best labelling of a valid scrambling system. As a result, the different algorithms could be represented with different labelling functions, and the labelling functions could be compared within criteria defined in Chapter 4.

In Chapter 4, we devised criteria to compare different labelling functions, which include coverage, labelling score, and efficiency. This gave a way to compare algorithms with both a biological and bioinformatics perspectives.

The A-L algorithm does not give a labelling of the scrambling system, or attempt to align the segments, but only detects the MDSs, IESs, and pointers. In order to fill this gap, and in order to be more precise with partitioning micronuclear genes and macronuclear genes, a new algorithm was created. Parameters are used for a match score, a mismatch score, a gap penalty, a cut-off score and a percentage penalty. Furthermore, study of the best values of these parameters is to be considered in future research. With some parameters considered, our new method did achieve better coverage and labelling score, and indeed, the average was higher. However, for some gene pairs, this was not the case. The time complexity of the new algorithm is $O(|s| \cdot |r| \cdot \max\{|s|, |r|\})$.

Further, a new percentage penalty scoring scheme was introduced in Chapter 4, which gave a way to align both long and short matching segments. Generally speaking, both the coverage and

labelling scores are better when using a percentage penalty. This method could also be used for other problems which is left for future research. Also, when the cutoff score is lower, more MDSs are detected and better scores could be achieved. However, when the cutoff score is too small, many MDSs are found which may not all be MDSs. In this particular case, we suspected that 10 is a better score than 5 when using a percentage penalty, or perhaps something in between 5 and 10.

Considering the last gene pair (data numbering 36), this is the only case seen with a low labelling score when using a percentage penalty. This could be because the real MDS sizes of this gene are too small. As we looked at this labelling result, many mismatches and gaps occur. This could indicate that this could be improved by choosing a better set of parameters, which is left for future research.

In Chapter 5, an algorithm suite was introduced to attempt to either refine or reject the feasibility of some scrambling hypotheses, and to try to reveal the influence of different factors like window distance, window size, and window sequence on descrambling. The left and right slithering ranges of all 247 pointer sequences in 13 micronuclear genes has been collected, with the variants of KnotD, KnotDS, KnotDSe, KnotS, KnotSe, and KnotDSe3. Certain statistical attributes of the pointer sequences was calculated. Moreover, the percentage of unique repeats, the minimum slithering range, the median slithering range of each micronuclear gene, gene group and total have been calculated and presented with each variant.

From the data summary, we could tell that with the strongest constraint, KnotDSe, obtains the best results most of the time. It has the total percentage of unique repeats, 92.7% and 94.3% of the time from the left and right respectively. It has a minimum slithering range of 8 and 4, but this stems from the fact that there are 2.8% of the pointer sequences with the size of 2. Finally, the total median slithering range is 5656, which is the length of micronuclear gene 44. Indeed, the median slithering range for each gene or gene group is the length of a corresponding micronuclear gene with this constraint. Then, if we were to rank the constraints from strongest to weakest in terms of that factor being enough to guide descrambling, we get KnotDSe, KnotDS, KnotSe, KnotDSe3, KnotD and KnotS. Comparing KnotS and KnotDS, and also KnotSe and KnotDSe, the window distance constraint significantly enlarges the slithering range of the pointers around the recombination site. From the result of the KnotD and KnotDS constraints, we can see that the window size constraint is also important. Comparing KnotS and KnotSe, if we know the window sequence, and not just the size, then this helps significantly in eliminating fake repeats. However, comparing KnotDS, and KnotDSe, we do have a larger slithering range for KnotDSe, but the values for KnotDS are already quite large. It is unknown what values are sufficient from a biological perspective. The variant KnotSe3, which uses only the distance between the real pointers together with the first 3 nucleotides of the real pointer sequence, is also a strong constraint. Indeed, the total minimum of the median slithering ranges is 583 nucleotides.

In the future, a range of window distances instead of a fixed window distance could be considered. This would enable the structure to have some variation. Further, the elimination order of MDSs is unknown and not considered in this thesis, and the distance could be altered after certain eliminations. This would require more biological experiments and is left for future research.

Although the biological mechanisms involved with descrambling are still not completely understood and need to be tested with biological experiments, the results in this thesis seem to support the structural hypothesis, in that the structure along with other factors can almost always provide enough information to precisely match real pointers.

References

[1] Stephen F. Altschul, Thomas L. Madden, Alejandro A. Schaffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25(17):3389–3402, June 1997.

[2] Andre R. O. Cavalcanti, Thomas H. Clarke, and Laura F. Landweber. MDS_IES_DB: a database of macronuclear and micronuclear genes in spirotrichous ciliates. *Nucleic Acids Research*, 33:396–398, 2005.

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

[4] M. Daley. *Computational Modeling of Genetic Processes in Stichotrichous Ciliates*. PhD thesis, University of Western Ontario, 2003.

[5] M. Daley and I. McQuillan. Template-guided DNA recombination. *Theoretical Computer Science*, 330(2):237–250, 2005.

[6] Mark Daley, Ian McQuillan, Nicholas A. Stover, and Laura F. Landweber. A simple topological mechanism for gene descrambling in stichotrichous ciliates, 2008. Unpublished manuscript.

[7] A. Ehrenfeucht, T. Harju, I. Petre, D.M. Prescott, and G. Rozenberg. *Computation in Living Cells, Gene Assembly in Ciliates*. Springer-Verlag, Berlin, 2004.

[8] A. Ehrenfeucht, D.M. Prescott, and G. Rozenberg. Computational aspects of gene (un)scrambling in ciliates. In L.F. Landweber and E. Winfree, editors, *Evolution as Computation*, pages 45–86. Springer-Verlag, Berlin, Heidelberg, 2001.

[9] Anne Fleury, Pilar Delgado, Francine Iftode, and André Adoutte. Molecular phylogeny of ciliates: What does it tell us about the evolution of the cytoskeleton and of developmental strategies? *Developmental Genetics*, 13:247–254, 2005.

[10] Jing Huang, Tamar Schlick, and Alexander Vologodskii. Dynamics of site juxtaposition in supercoiled DNA. *PNAS*, 98(3):968–973, January 2001.

[11] L. Kari and L.F. Landweber. Computational power of gene rearrangement. In E. Winfree and D. Gifford, editors, *DNA5, DIMACS series in Discrete Mathematics and Theoretical Computer Science*, volume 54, pages 207–216. American Mathematical Society, 2000.

[12] J. Mark Keil, Jing Liu, and Ian McQuillan. Algorithmic properties of ciliate sequence alignment, 2009. Submitted.

[13] Dan E. Krane and Michael L. Raymer. *Fundamental Concepts of Bioinformatics*. Benjamin-Cummings Publishing Co., San Francisco, 2002.

[14] Mark A. Krasnow, Andrzej Stasiak, Sylvia J. Spengler, Frank Dean, Theo Koller, and Nicholas R. Cozzarelli. Determination of the absolute handedness of knots and catenanes of DNA. *Nature*, 304(5926):559–560, 1983.

[15] L.F. Landweber and L. Kari. The evolution of cellular computing: Nature's solution to a computational problem. In L. Kari, H. Rubin, and D.H. Wood, editors, *DNA4, BioSystems*, volume 52, pages 3–13. Elsevier, 1999.

[16] L.F. Landweber, T. Kuo, and E.A. Curtis. Evolution and assembly of an extremely scrambled gene. *PNAS*, 97:3298–3303, 2000.

[17] John F. Marko. The internal 'slithering' dynamics of supercoiled DNA. *Physica A*, 244:263–277, 1997.

[18] Adrian G. Paschka, Franziska Jönsson, Verena Maier, Matthias Möllenbeck, Katrin Paeschke, Jan Postberg, Sina Rupprecht, and Hans J. Lipps. The use of RNAi to analyze gene function in spirotrichous ciliates. *European Journal of Protistology*, 39:449–454, 2003.

[19] I. Q. H. Phan, S. F. Pilbout, W. Fleischmann, and A. Bairoch. NEWT, a new taxonomy portal. *Nucleic Acids Research*, 31(13):3822–3823, 2003.

[20] David M. Prescott. The DNA of ciliated protozoa. *Microbiological Reviews*, 58(2):233–267, 1994.

[21] D.M. Prescott. Cutting, splicing, reordering, and elimination of DNA sequences in hypotrichous ciliates. *BioEssays*, 14(5):317–324, 1992.

[22] D.M. Prescott. The unusual organization and processing of genomic DNA in hypotrichous ciliates. *Trends in Genet.*, 8:439–445, 1992.

[23] D.M. Prescott. Genome gymnastics: Unique modes of DNA evolution and processing in ciliates. *Nature Reviews Genetics*, 1:191–198, 2000.

[24] D.M. Prescott, A. Ehrenfeucht, and G. Rozenberg. Template-guided recombination for IES elimination and unscrambling of genes in stichotrichous ciliates. *Journal of Theoretical Biology*, 222(3):323–330, 2003.

[25] Benjamin Raphael, Degui Zhi, Haixu Tang, and Pavel Pevzner. A novel method for multiple alignment of sequences with repeated and shuffled elements. *Genome Res*, 14:2336–2346, 2004.

[26] Meng-Chao Yao. Studies of an old genetic puzzle in a little-known protozoan reveal a new frontier in the expanding world of RNAs: an RNA template guides genome-wide DNA rearrangements during sexual reproduction. *Nature*, 451(10):131–132, 2008.

Appendix A

PERL code of the new algorithm to partition micronuclear genes and macronuclear genes

```
#!/usr/bin/perl

@infileMACg=("38stnMAC.fasta");
@infileMICg=("38stnMIC.fasta");

open(MYOUT, ">result.txt");
*STDOUT = *MYOUT;
print "The result:\n";

$fileNum=0;

foreach $infileMAC (@infileMACg)
{

$time1=time();

$infileMIC=$infileMICg[$fileNum];

################################
# read input                   #
################################

#print "Enter the name of a file with MAC sequence: ";
#$infileMAC = <>;
print "the MAC file is $infileMAC \n";
open MACFILE, $infileMAC;

#print "Enter the name of a file with MIC sequence: ";
#$infileMIC = <>;
print "the MIC file is $infileMIC \n";
open MICFILE, $infileMIC;


$inputMAC="";
$inputMIC="";

##############################################
#    input MAC                               #
##############################################
$line=<MACFILE>; #get rid of the first line   according to fasta format
while(defined($line=<MACFILE>)){
 chomp $line;
 $inputMAC=$inputMAC.$line;
# combine the input to a single string
}
##############################################
#    input MIC                               #
##############################################
$line=<MICFILE>;
while(defined($line=<MICFILE>)){
chomp $line;
$inputMIC=$inputMIC.$line;
# combine the input to a single string
}

#print "\n the MAC is\n";
#print "$inputMAC"; #test
```

```
59 print "\n the MIC is\n";
   print "$inputMIC"; #test
61
   $begin="CCCCAAAA";
63 $end="GGGGTTTT";
   $inputMAC=~ /$begin+(.*)$end+/;
65 $inputMAC=$1;
   #print "\n$inputMAC\n";
67 while($inputMAC=~ /$begin+(.*)$end+/){
   $inputMAC=$1;
69 #print "\n$theString\n";
   }
71 print "\n the MAC is\n";
   print "$inputMAC"; #test
73
   $originalMIC=$inputMIC; # save for different solutions
75 $originalMAC=$imputMAC;

77 close MICFILE;
   close MACFILE;
79 ################### the array to save the final result
   @MACalign=(); @MICalign=();
81 @MACposition=(); @MICposition=();
   ############################################################
83
   %bestMatch = findBest($inputMAC,$inputMIC);
85 $currentMAC = reverse $inputMAC;
   $currentMAC=~ tr/ATCG/TAGC/;
87 %bestRMatch = findBest($currentMAC,$inputMIC);

89 $bigNum = $bestMatch{"bigNum"};
   $bigNum2 = $bestRMatch{"bigNum"};
91
   if($bigNum>$bigNum2){
93 $MACseg = $bestMatch{"macSeg"};
   $MICseg = $bestMatch{"micSeg"};
95 $currentMACI=$bestMatch{"MACindex"};
   $currentMICI=$bestMatch{"MICindex"};
97 #printf "the biggest number is %d\n", $bigNum;
   #print "the MAC INDEX is $currentMACI\n";
99 #print "the MIC INDEX is $currentMICI\n";

101 #print "\n###################################\n";
   #print "the MAC segment is \n$MACseg\n";
103 #print "the MIC segment is \n$MICseg\n";
   }else{
105 $MACseg = $bestRMatch{"macSeg"};
   $MACseg =~ tr/ATCG/TAGC/;
107 $MACseg = reverse $MACseg;
   $MICseg = $bestRMatch{"micSeg"};
109 $currentMACI=$bestRMatch{"MACindex"};
   $currentMICI=$bestRMatch{"MICindex"};
111 #printf "the biggest number is %d\n", $bigNum2;
   $currentMACI=length($inputMAC)-$currentMACI-length($MACseg);
113 #print "the MAC INDEX is $currentMACI\n";
   #print "the MIC INDEX is $currentMICI\n";
115
   #print "\n###### Reverse Match ###############\n";
117 #print "the MAC segment is \n$MACseg\n";
   #print "the MIC segment is \n$MICseg\n";
119
   }
121 push(@MACalign,$MACseg); push(@MICalign,$MICseg);
   push(@MACposition,$currentMACI); push(@MICposition,$currentMICI);
123
   ##########################################
125 @MACgroup=(); @MICgroup=();
```

```
@MACI=(); @MICI=();
127 ##########################################

129 $finish=0;  $foundAll=0;
$myLength=length($MACseg);
131 #print "the length is $myLength\n";
#print "\n###################################\n";
133

135 $formerMAC=substr($inputMAC,0,$currentMACI);
$laterMAC=substr($inputMAC,$currentMACI+length($MACseg),length($inputMAC)-
    $currentMACI-length($MACseg));
137
#print "\n\n\n";
139 #print "the former MAC is \n$formerMAC \n";
#print "the latter MAC is \n$laterMAC \n";
141

143 $formerMIC=substr($inputMIC,0,$currentMICI);
$laterMIC=substr($inputMIC,$currentMICI+length($MICseg),length($inputMIC)-
    $currentMICI-length($MICseg));
145
#print "the former MIC is \n$formerMIC \n";
147 #print "the latter MIC is \n$laterMIC \n";

149
if(length($formerMAC)>5){ push(@MACgroup,$formerMAC); push(@MACI,0);}
151 if(length($laterMAC)>5){push(@MACgroup,$laterMAC); push(@MACI,length($formerMAC)+
    length($MACseg));}
if(length($formerMIC)>5){push(@MICgroup,$formerMIC); push(@MICI,0);}
153 if(length($laterMIC)>5){push(@MICgroup,$laterMIC); push(@MICI,length($formerMIC)+
    length($MICseg));}

155 ########################################
#test
157 #$currentMAC =  pop(@MACgroup);
#print "#########  the current MAC is $currentMAC\n";
159 ##############################################

161 #use code below to eliminate solution
#$foundAll=1;
163
$round=0;
165
while($foundAll==0){  # find all the acceptable pairs
167
%nextMatch = findNext();          #solution 1
169
if($nextMatch{"found"} == 1){
171
$MACseg = $nextMatch{"MACseg"};
173 $MICseg = $nextMatch{"MICseg"};
$lengthTemp=length($MACseg);
175 $MACnum=@MACgroup;
$MICnum=@MICgroup;
177 $MACindex=$nextMatch{"MACindex"};
$MICindex=$nextMatch{"MICindex"};
179 #print "\n#################################################################\n";
#print "find best finished in round $round\n";
181 #print "the length of the best match is $lengthTemp\n";
#print "the best MAC segment in this round is \n $MACseg\n";
183 #print "the best MIC segment in this round is \n $MICseg\n";
#print "the MAC index is $MACindex, and the MIC index is $MICindex\n";
185 #print "there are $MACnum MAC and $MICnum MIC\n";
#print "\n#################################################################\n";
187 push(@MACalign,$MACseg); push(@MICalign,$MICseg);
push(@MACposition,$MACindex); push(@MICposition,$MICindex);
189  $round++;
```

```
}else{
print"\n#############################################################\n";
        print"\n#############################################################\n";
  print"final result\n";
 $foundAll=1; print "find all the pairs in right order in round $round\n";
  $leftLength=0;
  foreach $tempMAC (@MACgroup){
   $leftLength+=length($tempMAC);
  }
   print "the total length of non-matched MAC is $leftLength\n";
   $coverage=(length($inputMAC)-$leftLength)/length($inputMAC);
   print "the percentage of coverage is $coverage\n";
print "\n#############################################################\n";
    ############
    ############
    $alignCount=0;
   foreach $tempMACal (@MACalign){
      print"\n#############################################################\n";
      $myLength=length($tempMACal);
        print "the length is $myLength\n";
        print "the best MAC segment in round $alignCount is \n $tempMACal\n";
        print "the best MIC segment in round $alignCount is \n $MICalign[$alignCount
            ]\n";
        print "the MAC index is $MACposition[$alignCount], and the MIC index is
            $MICposition[$alignCount]\n";
        print "\n#############################################################\n";
      $alignCount++;
   }
 }#endelse
}#endwhile

$time2 = time();
$runningtime=$time2-$time1;
print "\n Running time : $runningtime seconds\n";

##########################################################
## calculate global sequence alignment score.
##########################################################

@MDSscore=();
$MDSpenalty=-5;
$gsacount=0;

foreach $MACI (@MACposition){
  $MICI=$MICposition[$gsacount];
  $MACS=length($MACalign[$gsacount]);
  $MICS=length($MICalign[$gsacount]);
  $currentMIC=substr($inputMIC,$MICI,$MICS);
  $currentMAC=substr($inputMAC,$MACI,$MACS);
  $reverseMAC=reverse $currentMAC;
  $reverseMAC =~ tr/ATCG/TAGC/;
  $score=calcugsa($currentMAC,$currentMIC);
  $score2=calcugsa($reverseMAC,$currentMIC);

  if($score>$score2)
   {
     push(@MDSscore, $score);
  } else {push(@MDSscore,$score2);}
   $gsacount++;
 }#end foreach data set

$myscore=0;
$roundCount=0;
$MDSnumber=@MACposition;
foreach $score (@MDSscore)
{
 $myscore = $myscore+$score;
 print "\n The current score in round $roundCount is $score \n";
```

```
    $roundCount++;
257 }
    $myscore = $myscore+$MDSnumber*$MDSpenalty;
259  print "\n The total labelling score is $myscore\n";

261  print "finished $fileNum test\n";
    $fileNum++;
263 }#end foreach @macfile
    print "test finished!\n";
265 ##########################sub function###########
    ##############################################

267
    ################## find the next match
        ##########################################
269 sub findNext{
    #local($ref1,$ref2) = @_;
271  #local(@MACgroup)=@$ref1;
    #local(@MICgroup)=@$ref2;
273 $found = 0;  @testedMAC=(); @testedMIC=();  @testMACI=(); @testMICI=();
    $countMAC=0; $countMIC=0;
275 $bestMAC=""; $bestMIC="";  $bestNumber=0; $MACnum=0; $MICnum=0;
    $currentMACI=0; $currentMICI=0; $bestMACindex=0; $bestMICindex=0;
277 ######################################
    #test
279 #$currentMAC =  pop(@MACgroup);
    #print "the current MAC in sub function is $currentMAC\n";
281 #$currentMAC =  pop(@MACgroup);
    #print "the current MAC in sub function is $currentMAC\n";
283 #$currentMIC =  pop(@MICgroup);
    #print "the current MIC in sub function is $currentMIC\n";
285 #$currentMIC =  pop(@MICgroup);
    #print "the current MIC in sub function is $currentMIC\n";
287 ##########################################

289 foreach $currentMAC (@MACgroup){
        $countMIC=0;
291     foreach $currentMIC (@MICgroup){
    #  print "\ntest part \n\n the new MAC is $currentMAC\n the new MIC is
        $currentMIC\n" ;
293      %bestMatch = findBest($currentMAC,$currentMIC);
         $revMAC = reverse $currentMAC;
295      $revMAC=~ tr/ATCG/TAGC/;
         %bestRMatch = findBest($revMAC,$currentMIC);
297           $bigNum = $bestMatch{"bigNum"};
              $bigNum2 = $bestRMatch{"bigNum"};
299        if($bigNum>$bigNum2){
              $MACseg = $bestMatch{"macSeg"};
301           $MICseg = $bestMatch{"micSeg"};
              $currentMACI=$bestMatch{"MACindex"};
303                $currentMICI=$bestMatch{"MICindex"};
             # print "the currentMACI is $currentMACI and the currentMICI is
                $currentMICI \n";
305          }else{ # the reverse case
             $MACseg = $bestRMatch{"macSeg"};
307          $MACseg=~ tr/ATCG/TAGC/;
             $MACseg = reverse $MACseg;
309          $MICseg = $bestRMatch{"micSeg"};
             $currentMACI=$bestRMatch{"MACindex"};
311               $currentMICI=$bestRMatch{"MICindex"};
             # print "the currentMACI is $currentMACI and the currentMICI is
                $currentMICI \n";
313          $currentMACI=length($currentMAC)-$currentMACI-length($MACseg);
             #$mytemp=length($inputMAC);
315          #print"\n the currentMICI is $currentMICI, the length is $mytemp\n";
             # print "\n the currentMACI is $currentMACI and the currentMICI is
                $currentMICI \n";
317               }#end else
           if($bigNum > 10 && $bigNum > $bestNumber){
```

```
319                 $bestNumber=$bigNum; $bestMAC=$MACseg; $bestMIC=$MICseg;
                     $found=1; $MACnum=$countMAC; $MICnum=$countMIC;
321                  $bestMACindex=$currentMACI; $bestMICindex=$currentMICI;
                     #print "the MACindex changed to $bestMACindex, the MICindex changed to
                          $bestMICindex \n";
323                  #$mytemps=substr($currentMAC,$currentMACI,22);
                     #print" the sequence is $mytemps \n";
325
            }#end if
327       $countMIC++;
     }#end while
329    $countMAC++;
   }#end while
331 $countMIC--;
   $countMAC--;
333 #$countMAC--; #caused by the last add operation.
   #print "the count MIC here is $MICnum and count MAC here is $MACnum\n";
335 #print "the count MIC here is $countMIC and count MAC here is $countMAC\n";
   ##################################################
337 #### segments after the match ######
   if($found==1){
339 while($countMAC!=$MACnum){
     $currentMAC=pop(@MACgroup);   push(@testedMAC,$currentMAC);
341  $currentMACI=pop(@MACI);      push(@testedMACI,$currentMACI);
     $countMAC--;
343 }

345 while($countMIC!=$MICnum){
     $currentMIC=pop(@MICgroup);   push(@testedMIC,$currentMIC);
347  $currentMICI=pop(@MICI);      push(@testedMICI,$currentMICI);
     $countMIC--;
349 }
   ######################################
351  $currentMAC=pop(@MACgroup);
     $currentMIC=pop(@MICgroup);
353  $currentMACI=pop(@MACI);
     $currentMICI=pop(@MICI);
355 #print "\n##############\n";
   #print "the original MAC with best match is \n $currentMAC\n";
357 #print "the original MIC with best match is \n $currentMIC\n";
   #print "the mac index is $currentMACI and the mic index is $currentMIC\n";
359 #print "the countMAC is $MACnum, the countMIC is $MICnum\n";
   #print "\n##############\n";
361
   #print "the position that find mac is $bestMACindex and the mic is $bestMICindex \n
        ";
363
   #$currentMAC=~ /(.*)$bestMAC(.*)/;
365 #$formerMAC=$1;
   #$laterMAC=$2;
367 #print "the former MAC is $formerMAC\n";
   #print "the later MAC is $laterMAC\n";
369
   $formerMAC=substr($currentMAC,0,$bestMACindex);
371 $laterMAC=substr($currentMAC,$bestMACindex+length($bestMAC),length($currentMAC)-
        $bestMACindex-length($bestMAC));

373 $bestMACindex=$bestMACindex+$currentMACI;

375 if(length($formerMAC)>5){ push(@MACgroup,$formerMAC); push(@MACI,$currentMACI);}
   if(length($laterMAC)>5){ push(@MACgroup,$laterMAC); push(@MACI, $bestMACindex+length
        ($bestMAC));}
377
   $formerMIC=substr($currentMIC,0,$bestMICindex);
379 $laterMIC=substr($currentMIC,$bestMICindex+length($bestMIC),length($currentMIC)-
        $bestMICindex-length($bestMAC));

381 $bestMICindex=$bestMICindex+$currentMICI;
```

```
383 if(length($formerMIC)>5){push(@MICgroup,$formerMIC);push(@MICI,$currentMICI);}
    if(length($laterMIC)>5){push(@MICgroup,$laterMIC);push(@MICI, $bestMICindex+length(
        $bestMIC));}
385

387 ########segments before the match##########
    while(defined($currentMAC = pop(@testedMAC))){
389  push(@MACgroup,$currentMAC);
     $currentMACI=pop(@testedMACI); push(@MACI,$currentMACI);
391 }
    while(defined($currentMIC = pop(@testedMIC))){
393  push(@MICgroup,$currentMIC);
    $currentMICI=pop(@testedMICI); push(@MICI,$currentMICI);
395 }
    }#end if
397
     %nextMatch = ("MACseg",$bestMAC,"MICseg",$bestMIC,"found",$found,"MACindex",
          $bestMACindex,"MICindex",$bestMICindex);
399  return %nextMatch;
    }
401
    ###########################################################################
403 ################## find the best match
    ###########################################################################
405
    sub findBest{
407
    local($myinputMAC,$myinputMIC) = @_;
409 $gap=-1; $match=1; $mismatch=-2;
    @scoreArray=(); @currentRow=(); @previousRow=(); $biggestNum=0; @index=(); #index
        save the index of the biggest value
411
    $valueC; $valueR; $valueM; # the value of current state calculated by column, row,
        and match/mismatch
413 $value;   $biggestNum=0;
    $MACseg; $MICseg;
415
    @matchArray=(); # matchArray save the indexes of the matched sequences.
417
    $lengthMAC=length($myinputMAC);
419 $lengthMIC=length($myinputMIC);

421 $numMAC=0; $numMIC=0; # i+1 row, j+1 column
    $numZero = 0;
423
    ###initial the first row #######
425 while($numMIC < $lengthMIC+1){
      push(@currentRow,$numZero);
427   $numMIC = $numMIC + 1;
    }
429 @previousRow = @currentRow;
    push(@scoreArray,[@currentRow]);
431 @currentRow = ();
    $numMIC = 0;
433 ################################
    while($numMAC < $lengthMAC){
435  push(@currentRow,$numZero);  # initial the first column.
     while($numMIC< $lengthMIC){
437      $tempMAC = substr($myinputMAC,$numMAC,1);
         $tempMIC = substr($myinputMIC,$numMIC,1);
439
         if($tempMAC eq $tempMIC){
441       $valueM=$previousRow[$numMIC]+$match;
          # print "the current mac is $tempMAC, the current mic is $tempMIC  EQUAL!!!!\n
              ";
443       # print "the indices is $numMAC,$numMIC\n";
        }
```

```
445         else{
              if($previousRow[$numMIC]/2>(0-$mismatch))
447             { $valueM=int($previousRow[$numMIC]/2);}
                else{
449               $valueM=$previousRow[$numMIC]+$mismatch;
                }
451         # print "the current mac is $tempMAC, the current mic is $tempMIC";
            # print "the index is $numMAC,$numMIC\n";
453       }

455       if($previousRow[$numMIC+1]/2>(0-$gap))
          { $valueC=int($previousRow[$numMIC+1]/2); }
457        else{
             $valueC = $previousRow[$numMIC+1]+$gap;
459        }
          if($currentRow[$numMIC]/2>(0-$gap))
461        { $valueR = int($currentRow[$numMIC]/2); }
           else{
463          $valueR = $currentRow[$numMIC]+$gap;
           }
465           ################ compare and decide value ##########
          $value = $valueM;
467       if($valueC > $value){
            $value = $valueC;
469       }
          if($valueR > $value){
471         $value = $valueR;
          }
473
          if($value<0){
475          $value = 0;
          }
477
          if($value>$biggestNum){
479         $biggestNum=$value;
            @index=();
481         push(@index,$numMAC);
            push(@index,$numMIC);
483       }
           ################ compare and decide value ##########
485        push(@currentRow,$value);
           $numMIC = $numMIC+1;
487
         #  print "$tempMAC, tempMIC, $valueC, $valueM, $valueR";
489
       }#end whileMIC
491     push(@scoreArray,[@currentRow]);
        @previousRow = @currentRow;
493     @currentRow = ();
        $numMAC = $numMAC+1;
495     $numMIC = 0;
    }#end whileMAC
497   $numMAC = 0;
    ################################
499 ## output test              ##
    ################################
501 #print "\n\n";
    #$rowCount=0;
503 #foreach $currentRow (@scoreArray)
    #{
505 #  print "this are the results at the row: $rowCount. \n";
    #  foreach $outValue (@$currentRow)      # currentRow here is a pointer #in perl.
507 #  {
    #    print "$outValue  ";
509 #  }#end foreach
    #     $rowCount++;
511 #    print "\n";
    #}#end foreach
```

```
513
    #printf "\nthe biggest number is %d, in row: %d, column: %d\n", $biggestNum,$index
        [0],$index[1];
515 #printf "the biggest number is %d\n",$scoreArray[$index[0]+1][$index[1]+1];
    #printf "there are %d rows (MAC), and %d columns (MIC)\n",$lengthMAC,$lengthMIC;
517 ###################
    $MACindex=$index[0];
519 $MICindex=$index[1];

521 push(@matchArray,[@index]);

523 #$tempi=substr($myinputMIC,$index[1],1); $tempm=substr($myinputMAC,$index[0],1);
    #  print "the equal ones are $tempi and $tempm\n";
525 #  print "the index are $index[0],$index[1] \n";

527 while($scoreArray[$index[0]+1][$index[1]+1]!=0){
    $tempi=substr($myinputMIC,$index[1],1); $tempm=substr($myinputMAC,$index[0],1);
529   $cS=$scoreArray[$index[0]+1][$index[1]+1]; #current score
      $pS=$scoreArray[$index[0]][$index[1]];     #previous score
531 if(($cS==$pS+$match)&&($tempi eq $tempm)){
      $index[0]=$index[0]-1;
533   $index[1]=$index[1]-1;
    #test#########
535 #  print "the equal ones are $tempi and $tempm\n";
    #  print "the index are $index[0],$index[1] \n";
537  }else{
     if (((($cS==$pS+$mismatch)||($cS==int($pS/2)))&&($tempi ne $tempm)){
539   $index[0]=$index[0]-1;
      $index[1]=$index[1]-1;
541 #test##########
    #$tempi=substr($myinputMIC,$index[1],1); $tempm=substr($myinputMAC,$index[0],1);
543 #  print "the not equal ones are $tempi and $tempm\n";
    #  print "the index are $index[0],$index[1] \n";
545  }else{
      $pS= $scoreArray[$index[0]][$index[1]+1];
547   if (($cS==$pS+$gap)||($cS==int($pS/2))){
       $index[0]=$index[0]-1;
549   }else{
       $pS=$scoreArray[$index[0]+1][$index[1]];
551    if (($cS==$pS+$gap)||($cS==int($pS/2))){
        $index[1]=$index[1]-1;
553    }else{ print "error at $index[0], $index[1] \n";}
      }
555  }
    }
557  push(@matchArray,[@index]);
    }#end while
559  pop(@matchArray); #because the last one is not the match.

561  ########################
    #test                  #
563  ########################
    #foreach $currentIndex (@matchArray)
565 #{
    #  foreach $indexValue (@$currentIndex)      # currentRow here is a #pointer in perl
        .
567 #  {
    #    print "$indexValue ";
569 #  }#end foreach
    #    print "; \t";
571 #}#end foreach
    #print "\n \n";
573
    $tempx=0; $tempy=0; $shortLine="-"; $MICseg=""; $MACseg=""; #there shouldn't be " ",
        otherwise might have problem
575  foreach $currentMatch (@matchArray)
    {
577   @currentIndex = @$currentMatch;
```

```
    if($currentIndex[0]==$tempx){
579 #  $MACseg=$MACseg.$shortLine;    #just for printing test
    }else{
581   $MACseg=$MACseg.substr($myinputMAC,$currentIndex[0],1);}

583  if($currentIndex[1]==$tempy){
    #     $MICseg=$MICseg.$shortLine;     #just for printing test
585 }else{
      $MICseg=$MICseg.substr($myinputMIC,$currentIndex[1],1);}
587
     $tempx=$currentIndex[0];   $tempy=$currentIndex[1];
589 }#end foreach

591 #print "\n#################################\n";
    $myLength=length($MACseg);
593 #print "the length is $myLength\n";

595 # change the result to the right order
    $MICseg = reverse $MICseg;
597 $MACseg = reverse $MACseg;

599
    $MACindex=$MACindex-length($MACseg)+1;
601 $MICindex=$MICindex-length($MICseg)+1;

603 #print "\n#################################\n";
    #print "the MAC segment in best match function is \n $MACseg\n";
605 #print "the MIC segment in best match function is \n $MICseg\n";
    #print "the MAC index in best match function is \n $MACindex\n";
607 #print "the MIC index in best match function is \n $MICindex\n";
    #print "\n#################################\n";
609

611 %returnValue = ("bigNum",$biggestNum,"macSeg",$MACseg,"micSeg",$MICseg,"MACindex",
        $MACindex,"MICindex",$MICindex);
     return %returnValue;
613 }

615 ######################################################################
    ############################calculate gsa#############################
617 ######################################################################

619 sub calcugsa{

621 local($currentMAC,$currentMIC) = @_;

623 $gap=-1; $match=1; $mismatch=-2;

625 @scoreArray=(); @currentRow=(); @previousRow=();

627 $valueC; $valueR; $valueM; # the value of current state calculated by column, row,
        and match/mismatch
    $value=0;
629 $MACseg; $MICseg;

631 $lengthMAC=length($currentMAC);
    $lengthMIC=length($currentMIC);
633
    $numMAC=0; $numMIC=0; # i+1 row, j+1 column
635 $numZero = 0;

637 ###initial the first row #######
    while($numMIC < $lengthMIC+1){
639   push(@currentRow,$numZero-$numMIC );
      $numMIC = $numMIC + 1;
641 }
    @previousRow = @currentRow;
643 push(@scoreArray,[@currentRow]);
```

```
    @currentRow = ();
645 $numMIC = 0;
    ################################
647 while($numMAC < $lengthMAC){
     push(@currentRow,$numZero-$numMAC);  # initial the first column.
649  while($numMIC< $lengthMIC){
        $tempMAC = substr($currentMAC,$numMAC,1);
651     $tempMIC = substr($currentMIC,$numMIC,1);

653      if($tempMAC eq $tempMIC){
          $valueM=$previousRow[$numMIC]+$match;
655       # print "the current mac is $tempMAC, the current mic is $tempMIC  EQUAL!!!!\n
              ";
          # print "the index is $numMAC,$numMIC\n";
657     }
          else{
659        $valueM=$previousRow[$numMIC]+$mismatch;
          # print "the current mac is $tempMAC, the current mic is $tempMIC";
661       # print "the index is $numMAC,$numMIC\n";
        }
663
        $valueC = $previousRow[$numMIC+1]+$gap;
665     $valueR = $currentRow[$numMIC]+$gap;

667         ################ compare and decide value ##########
        $value = $valueM;
669     if($valueC > $value){
          $value = $valueC;
671     }
        if($valueR > $value){
673       $value = $valueR;
        }
675
         ################ compare and decide value ##########
677
         push(@currentRow,$value);
679      $numMIC = $numMIC+1;

681     #  print "$tempMAC, tempMIC, $valueC, $valueM, $valueR";

683  }#end whileMIC
      push(@scoreArray,[@currentRow]);
685   @previousRow = @currentRow;
      @currentRow = ();
687   $numMAC = $numMAC+1;
      $numMIC = 0;
689 }#end whileMAC
      $numMAC = 0;
691

693 # the last value is the GSA value of two sequences.
    return $value;
695 }
```

Appendix B

PERL code of evaluating A-L algorithm

```
#!/usr/bin/perl

@MICIndex=(); @MACIndex=(); @MICIndex2 =(); @MACIndex2=();
@MICsgsize=(); @MACsgsize=(); @MDSscore=(); @pointerS=();

### command line version of input
#while($MICI >= 0){

#print "Enter the MIC start index: \n";
#$MICI = <>;
#print "Enter the MIC start index is $MICI \n";

#push(@MICIndex,$MICI);

#print "Enter the MIC end index: \n";
#$MICI2 = <>;
#print "Enter the MIC end index is $MICI2 \n";

#push(@MICIndex2,$MICI2);

#print "Enter the MAC start index: \n";
#$MACI = <>;
#print "Enter the MAC start index is $MACI \n";

#push(@MACIndex,$MACI);

#print "Enter the MAC end index: \n";
#$MACI2 = <>;
#print "Enter the MAC end index is $MACI2 \n";

#push(@MACIndex2,$MACI2);


#print "Enter the MIC segment size: \n";
#$MICss = <>;
#print "Enter the MIC segment size is $MICss \n";

#push(@MICsgsize,$MICss);

#print "Enter the MAC segment size: \n";
#$MACss = <>;
#print "Enter the MAC segment size is $MACss \n";

#push(@MACsgsize,$MACss);


#print "Enter the pointer size: \n";
#$MACss = <>;
#print "Enter the MAC segment size is $pss \n";

#push(@pointerS,$pss);
#}

#pop(@MICIndex); pop(@MACIndex); pop(@MICIndex2); pop(@MACIndex2); pop(@pointerS);
######### command line version of input finish ###############

open(MYOUT, ">result.txt");
*STDOUT = *MYOUT;
print "The result:\n";
```

```
61 ##########################################################################
   ##########      38stn     ############
63 @MICIndex=(2034,1959,428,949,1669,1411,1726,2240,1851);
   @MICIndex2=(2225,2019,920,1395,1706,1558,1839,2266,1942);
65 @MACIndex=(39,218,266,755,1189,1221,1359,1461,1477);
   @MACIndex2=(229,278,758,1201,1226,1368,1472,1487,1568);
67 @pointerS=(12,13,4,13,6,10,12,11);
   ##########################################################################
69 $infileMIC="38stnMIC.fasta";
   $infileMAC="38stnMAC.fasta";
71
   mygsa();
73
   ##########################################################################
75 ##########      35oxf     ############
   @MICIndex=(1,407);
77 @MICIndex2=(345,989);
   @MACIndex=(252,593);
79 @MACIndex2=(596,1175);
   @pointerS=(4,0);
81 ##########################################################################
   $infileMIC="35oxfMIC.fasta";
83 $infileMAC="35oxfMAC.fasta";

85 mygsa();

87 ##########################################################################
   ##########      1sth      ############
89 @MICIndex=(1851,1775,1,410,1219,1016,1273,2079,1386,1729);
   @MICIndex2=(2050,1829,345,992,1253,1167,1365,2115,1452,1752);
91 @MACIndex=(21,210,253,594,1164,1191,1330,1413,1439,1499);
   @MACIndex2=(220,264,597,1176,1198,1342,1422,1449,1505,1522);
93 @pointerS=(11,12,4,13,8,13,10,11,7,0);
   ##########################################################################
95 $infileMIC="1sthMIC.fasta";
   $infileMAC="1sthMAC.fasta";
97
   mygsa();
99
   ##########################################################################
101 ##########      28stp     ############
   @MICIndex=(1441,1381,1,409,1080,872,1142,1270);
103 @MICIndex2=(1600,1428,221,849,1119,1025,1253,1355);
   @MACIndex=(14,174,478,695,1124,1155,1300,1422);
105 @MACIndex2=(173,221,698,1135,1163,1308,1411,1507);
   @pointerS=(0,0,4,12,9,9,0,0);
107 ##########################################################################
   $infileMIC="28stpMIC.fasta";
109 $infileMAC="28stpMAC.fasta";

111 mygsa();

113 ##########################################################################
   ##########      32sth     ############
115 @MICIndex
       =(1,683,252,990,313,1047,507,1086,1166,559,1332,599,1410,1486,1723,2015,2359);
   @MICIndex2
       =(208,973,305,1024,478,1080,535,1118,1311,588,1394,626,1444,1657,1962,2250,2756)
       ;
117 @MACIndex=(49,244,527,564,589,749,773,791,821,959,978,1034,1051,1079,1244,1478,1707)
       ;
   @MACIndex2
       =(256,534,580,598,754,782,801,823,966,988,1040,1061,1085,1250,1483,1713,2104);
119 @pointerS=(13,8,17,10,6,10,11,3,8,11,7,11,7,7,6,7,0);
   ##########################################################################
121 $infileMIC="32sthMIC.fasta";
   $infileMAC="32sthMAC.fasta";
123
```

```
mygsa();

#########################################################################
##########      42urs     ###########
@MICIndex=(1,886,431,1058,474,1190,531,1272,684,1319,726,1365,768,1511,2171,3052);
@MICIndex2
    =(159,1049,452,1171,514,1243,662,1311,713,1357,753,1500,791,1613,2884,3205);
@MACIndex=(83,233,392,407,513,547,593,718,746,768,797,818,944,973,1061,1764);
@MACIndex2=(241,396,413,520,553,600,724,757,775,806,824,953,967,1073,1774,1917);
@pointerS=(9,5,7,8,7,8,7,12,8,10,0,13,11,0);
#########################################################################
$infileMIC="42ursMIC.fasta";
$infileMAC="42ursMAC.fasta";

mygsa();

#########################################################################
##########      41paw     ###########
@MICIndex=(1,477,66,771,117,825,268,865,313,919,354,1050,403,1111,1160,1721);
@MICIndex2=(47,753,103,813,260,852,301,898,345,1040,393,1094,443,1148,1676,1834);
@MACIndex=(207,244,518,554,586,717,740,767,791,819,927,958,998,1027,1061,1576);
@MACIndex2=(253,520,555,597,729,744,773,800,823,940,966,1002,1038,1064,1577,1689);
@pointerS=(10,3,2,12,13,5,7,10,5,14,9,5,12,4,2,0);
#########################################################################
$infileMIC="41pawMIC.fasta";
$infileMAC="41pawMAC.fasta";

mygsa();

#########################################################################
##########      37stn     ###########
@MICIndex=(50,760,328,1055,412,1128,585,1183,641,1369,674,1444,1514,2283);
@MICIndex2=(288,1042,379,1101,573,1159,620,1355,665,1418,711,1481,2227,2660);
@MACIndex=(37,265,539,582,617,770,789,816,983,999,1043,1062,1095,1806);
@MACIndex2=(275,547,590,628,778,801,824,988,1007,1048,1080,1099,1808,2182);
@pointerS=(11,9,9,12,9,13,9,6,9,6,19,5,3,0);
#########################################################################
$infileMIC="37stnMIC.fasta";
$infileMAC="37stnMAC.fasta";

mygsa();

#########################################################################
##########      33stm     ###########
@MICIndex=(1,793,342,1149,413,1209,624,1258,669,1464,720,1532,1605,2163);
@MICIndex2=(230,1081,396,1187,561,1252,651,1434,702,1520,752,1568,2117,2686);
@MACIndex=(21,232,512,557,582,724,755,774,940,965,1015,1035,1069,1579);
@MACIndex2=(250,520,566,595,730,767,782,950,973,1021,1047,1071,1581,2105);
@pointerS=(19,9,10,14,7,13,9,11,9,7,13,3,3,0);
#########################################################################
$infileMIC="33stmMIC.fasta";
$infileMAC="33stmMAC.fasta";

mygsa();

#########################################################################
##########      44urs     ###########
@MICIndex=(2273,2664,2102,3493,2025,3544,1958,3643,1889,3769,1755,3879,1535,3974,
1436,4047,1370,4174,497,4259,448,4365,312,4440,196,4547,139,4640,86,4764,
25,5011,5066,5540);
@MICIndex2=(2621,3477,2240,3530,2098,3626,1995,3759,1928,3833,1877,3953,1730,4038,
1512,4171,1420,4201,1367,4343,490,4414,442,4525,290,4622,184,4752,118,4998,
56,5063,5496,5656);
@MACIndex=(29,376,1177,1304,1331,1397,1467,1498,1607,1639,1696,1813,1875,2061,2120,
2189,2301,2340,2355,3210,3288,3323,3361,3481,3554,3640,3710,3748,3849,
3875,4103,4124,4186,4615);
@MACIndex2=(377,1189,1315,1341,1404,1479,1504,1614,1646,1703,1818,1887,2070,2125,
2196,2313,2351,2367,3225,3294,3330,3372,3491,3566,3648,3715,3755,3860,
```

```
3881,4109,4134,4176,4616,4731);
@pointerS
    =(2,13,12,11,8,13,7,8,8,8,6,13,10,6,8,13,12,13,16,7,8,12,11,13,9,6,8,12,7,7,11,0,2,0)
    ;
##########################################################################
$infileMIC="44ursMIC.fasta";
$infileMAC="44ursMAC.fasta";

mygsa();

#########################################################################
##########      19sth     ###########
@MICIndex=(1910,2303,3113,3359,1844,3454,1746,3516,1657,3570,1598,3625,1521,3678,
1470,3811,1357,3926,1241,4031,1112,4090,1021,4221,954,4394,846,4444,251
,1,4542,4661,4742,4947,5158,5305,5366,5787,6195,6365);
@MICIndex2=(2256,2907,3289,3420,1872,3491,1822,3554,1726,3603,1631,3654,1580,3791,
1497,3904,1452,3995,1328,4066,1196,4183,1087,4347,988,4420,943,4478,818
,199,4629,4713,4850,5067,5274,5338,5613,6125,6357,6501);
@MACIndex=(31,373,974,1175,1224,1245,1274,1341,1372,1434,1461,1487,1506,1552,1657
,1677,1759,1842,1900,1976,2009,2087,2172,2230,2349,2376,2395,2493,2513,
3073,3270,3369,3519,3670,3802,3928,3973,4341,4675,4833);
@MACIndex2
    =(377,978,1150,1236,1252,1282,1348,1379,1441,1467,1494,1516,1565,1664,1684,
1770,1854,1911,1987,2011,2093,2181,2238,2356,2383,2402,2492,2527,3080,
3271,3356,3421,3628,3790,3918,3962,4220,4680,4837,4968);
@pointerS=(5,5,0,13,8,9,8,8,8,7,8,11,14,8,8,12,13,12,12,3,7,10,9,8,8,8,0,15,8,2,0
,0,0,0,0,0,0,6,5,0);
#########################################################################
$infileMIC="19sthMIC.fasta";
$infileMAC="19sthMAC.fasta";

mygsa();

#########################################################################
##########      36paw     ###########
@MICIndex=(2494,2767,2833,3695,3929,2458,4284,2154,4317,2078,4368,2048,4417,2002,
4447,1957,4585,1937,4645,1829,4678,1718,4749,1593,4799,1489,4859,1435,
4979,581,5063,525,5174,262,5246,178,5344,138,5446,105,5575,37,5890,1,
5989,6394,6721,6798);
@MICIndex2=(2756,2818,3291,3707,4273,2492,4316,2236,4356,2143,4416,2071,4446,2035,
4574,1992,4638,1954,4668,1933,4738,1807,4781,1708,4852,1583,4961,1486,
5003,1428,5161,560,5235,381,5331,256,5439,165,5572,131,5869,69,5980,28,
6115,6510,6784,6930);
@MACIndex=(1,260,308,759,768,1107,1133,1158,1232,1264,1321,1361,1381,1404,1423,
1545,1570,1619,1630,1646,1742,1792,1873,1895,2003,2047,2133,2226,2266,
2285,3125,3218,3247,3296,3406,3484,3556,3643,3662,3778,3794,4082,4109,
4193,4214,4334,4449,4510);
@MACIndex2=(263,311,766,771,1112,1141,1165,1240,1271,1329,1369,1384,1410,1437,1550
,1580,1623,1636,1653,1750,1802,1881,1905,2010,2056,2141,2235,2277,2290,
3132,3223,3253,3308,3415,3491,3562,3651,3670,3788,3804,4088,4114,4199,
4220,4340,4450,4512,4642);
@pointerS=(4,4,8,4,6,9,8,9,8,9,9,4,7,15,6,11,5,7,8,9,11,9,11,8,10,9,10,12,6,8,6,
7,13,10,8,7,9,9,11,11,7,6,7,7,7,2,3,0);
#########################################################################
$infileMIC="36pawMIC.fasta";
$infileMAC="36pawMAC.fasta";

mygsa();

sub mygsa{

@MICsgsize=();  @MACsgsize=(); @MDSscore=();

print "\n#########################################\n";
print "\n#########################################\n";
#print "Enter the name of a file save MIC: ";
#$infileMIC = <>;
print "The MIC file is $infileMIC\n";
```

```
open MICFILE, $infileMIC;

#print "Enter the name of a file save MAC: ";
#$infileMAC = <>;
print "The MAC file is $infileMAC\n";
open MACFILE, $infileMAC;

$inputMAC="";
$inputMIC="";

##############################################
#    input MAC                               #
##############################################
$line=<MACFILE>; #get rid of the first line   according to fasta format
while(defined($line=<MACFILE>)){
 chomp $line;
 $inputMAC=$inputMAC.$line;
# combine the input to a single string
}
##############################################
#    input MIC                               #
##############################################
$line=<MICFILE>;
while(defined($line=<MICFILE>)){
chomp $line;
$inputMIC=$inputMIC.$line;
# combine the input to a single string
}

#print "\n the MAC is\n";
#print "$inputMAC"; #test

#print "\n the MIC is\n";
#print "$inputMIC"; #test

close MICFILE;
close MACFILE;
############################################################
############################################################
$mycount=0;

foreach $MICI (@MICIndex){
  $MACI=$MACIndex[$mycount];
  $MACI2=$MACIndex2[$mycount];
  $MICI2=$MICIndex2[$mycount];
  $pss=$pointerS[$mycount];
  $MICS=$MICI2-$MICI+1-$pss;
  $MACS=$MACI2-$MACI+1-$pss;
  push(@MACsgsize,$MACS);
  push(@MICsgsize,$MICS);

  $currentMIC=substr($inputMIC,$MICI,$MICS);
  $currentMAC=substr($inputMAC,$MACI,$MACS);
  $reverseMAC=reverse $currentMAC;
  $reverseMAC=~ tr/ATCG/TAGC/;
  #print "\n the current MIC is $currentMIC \n";
  #print "\n the current MAC is $currentMAC \n";
  #print "\n the current REVERSE MAC is $reverseMAC \n";

  $score=calcugsa($currentMAC,$currentMIC);
  $score2=calcugsa($reverseMAC,$currentMIC);

  #print "\n the score is $score \n";
  #print "\n the score2 is $score2 \n";

  if($score>$score2)
   {
     push(@MDSscore, $score);
```

```
    } else {push(@MDSscore,$score2);}
325   $mycount++;
    }#end foreach data set
327
  $myscore=0;
329 $MDSnumber=@MICIndex;
  $MDSpenalty=-5;
331
  foreach $score (@MDSscore)
333 {
   print "\n the current score is $score \n";
335  $myscore = $myscore+$score;
  }
337  $myscore = $myscore+$MDSnumber*$MDSpenalty;

339  $mysize=0;
   foreach $MACS (@MACsgsize){
341  $mysize=$mysize + $MACS-1;
  }
343  $mylength=length($inputMAC);
   $mycoverage=$mysize/$mylength*100;
345  print "\n $MDSnumber MDSs\n";
   print "\n The total labelling score is $myscore \n";
347  print "\n The coverage is $mycoverage percent \n";
   return;
349 }
  ##########################sub function##########
351 ################################################
  sub calcugsa{
353
  local($currentMAC,$currentMIC) = @_;
355
  $gap=-1; $match=1; $mismatch=-2;
357
  @scoreArray=(); @currentRow=(); @previousRow=();
359
  $valueC; $valueR; $valueM; # the value of current state calculated by column, row,
      and match/mismatch
361 $value=0;
  $MACseg; $MICseg;
363
  $lengthMAC=length($currentMAC);
365 $lengthMIC=length($currentMIC);

367 $numMAC=0; $numMIC=0; # i+1 row, j+1 column
  $numZero = 0;
369
  ###initial the first row #######
371 while($numMIC < $lengthMIC+1){
    push(@currentRow,$numZero-$numMIC );
373  $numMIC = $numMIC + 1;
  }
375 @previousRow = @currentRow;
  push(@scoreArray,[@currentRow]);
377 @currentRow = ();
  $numMIC = 0;
379 ################################
  while($numMAC < $lengthMAC){
381  push(@currentRow,$numZero-$numMAC);  # initial the first column.
   while($numMIC< $lengthMIC){
383     $tempMAC = substr($currentMAC,$numMAC,1);
      $tempMIC = substr($currentMIC,$numMIC,1);
385
      if($tempMAC eq $tempMIC){
387     $valueM=$previousRow[$numMIC]+$match;
      #  print "the current mac is $tempMAC, the current mic is $tempMIC  EQUAL!!!!\n
          ";
389   #  print "the index is $numMAC,$numMIC\n";
```

```
    }
391   else{
        $valueM=$previousRow[$numMIC]+$mismatch;
393   #  print "the current mac is $tempMAC, the current mic is $tempMIC";
      # print "the index is $numMAC,$numMIC\n";
395   }

397   $valueC = $previousRow[$numMIC+1]+$gap;
      $valueR = $currentRow[$numMIC]+$gap;
399
        ############### compare and decide value ##########
401   $value = $valueM;
      if($valueC > $value){
403     $value = $valueC;
      }
405   if($valueR > $value){
        $value = $valueR;
407   }

409    ############### compare and decide value ##########

411    push(@currentRow,$value);
       $numMIC = $numMIC+1;
413
  #     print "$tempMAC, tempMIC, $valueC, $valueM, $valueR";
415
   }#end whileMIC
417  push(@scoreArray,[@currentRow]);
     @previousRow = @currentRow;
419  @currentRow = ();
     $numMAC = $numMAC+1;
421  $numMIC = 0;
  }#end whileMAC
423  $numMAC = 0;

425 ##################################
  ## output test              ##
427 ##################################
  #print "\n\n";
429 #$rowCount=0;
  #foreach $currentRow (@scoreArray)
431 #{
  #  print "this are the results at the row: $rowCount. \n";
433 #  foreach $outValue (@$currentRow)      # currentRow here is a pointer #in perl.
  #  {
435 #    print "$outValue  ";
  #  }#end foreach
437 #    $rowCount++;
  #    print "\n";
439 #}#end foreach

441 # the last value is the GSA value of two sequences.
  #print "\n the value is $value \n";
443 return $value;
  }
```

Appendix C

PERL code of the new algorithm suite to test the hypotheses

C.1 Program that uses window to take control of distance

```
#!/usr/bin/perl

################################
# read input                   #
################################

##### command line version of input
#reverse order pointer?

#print "whether reversed? \n";
#chomp($pr = <>);
#print "whether reversed? $pr \n\n";

#$psize=0; @psizearray=(); @pdistancearray=();

#while($psize >= 0){

#print "Enter the pointer size: \n";
#$psize = <>;
#print "Enter the pointer size is $psize \n";

#push(@psizearray,$psize);

#print "Enter the pointer distance: \n";
#$pdistance = <>;
#print "Enter the pointer distance is $pdistance \n";

#push(@pdistancearray,$pdistance);
#}

#pop(@psizearray);
#pop(@pdistancearray);

#print "Enter the name of a file save MIC: ";
#$infileMIC = <>;
#open MICFILE, $infileMIC;
#############command line version of input finish

open(MYOUT, ">␣result.txt");
*STDOUT = *MYOUT;
print "The␣result:\n";

#print "whether reversed? $pr \n";

# example input
#########################################input#########
##########    36paw ########################

$pr='n';
@psizearray=(4,4,4,4,7,2,3);
@pdistancearray=(14,18,411,225,285,210,16);

$infileMIC="36pawMIC.fasta";
```

```
myscand();

$pr='y'; @rps=(6,9,8,9,8,9,9,4,7,11,6,6,5,7,8,9,11,7,6,8,10,4,10,12,6,8,6,7,13,10,8,
7,9,9,11,7,7,6,7,7);
@psizearray=(6,9,8,9,8,9,9,4,7,11,6,6,5,7,8,9,11,7,6,8,10,4,10,12,6,8,6,7,13,10,8,
7,9,9,11,7,7,6,7,7);
@pdistancearray
    =(1781,1826,2080,2163,2213,2290,2345,2369,2411,2449,2582,2633,2684,2708,
2735,2849,2931,3029,3068,3206,3269,3375,3475,3544,3575,4482,4601,4649,
4854,4984,5075,5166,5274,5308,5441,5474,5800,5853,5952,5988);

$infileMIC="36pawMIC.fasta";

myscand();

$pr='n';
@psizearray=(3,3,3,3,3);
@pdistancearray=(14,18,411,225,285);

$infileMIC="36pawMIC.fasta";

myscand();

$pr='y'; @rps=(6,9,8,9,8,9,9,4,7,11,6,6,5,7,8,9,11,7,6,8,10,4,10,12,6,8,6,7,13,10,8,
7,9,9,11,7,7,6,7,7);
@psizearray=(3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,3,
3,3,3,3,3,3);
@pdistancearray
    =(1781,1826,2080,2163,2213,2290,2345,2369,2411,2449,2582,2633,2684,2708,
2735,2849,2931,3029,3068,3206,3269,3375,3475,3544,3575,4482,4601,4649,
4854,4984,5075,5166,5274,5308,5441,5474,5800,5853,5952,5988 );

$infileMIC="36pawMIC.fasta";

myscand();

sub myscand{

$inputMIC;

##############################################
#    input MIC                               #
##############################################
print "The MIC file is $infileMIC\n";
open MICFILE, $infileMIC;

$inputMIC="";

$line=<MICFILE>;
while(defined($line=<MICFILE>)){
chomp $line;
$inputMIC=$inputMIC.$line;
# combine the input to a single string
}

#print "\n the MIC is\n";
#print "$inputMIC"; #test

close MICFILE;


#print "Enter the string: \n";
#$mystring = <>;

## test string #
 # $inputMIC="abcasdgaevasdfdasgsdfaseadabc";

```

```
    $miclength=length($inputMIC);
121 print "the length of MIC is $miclength \n";

123 #$miclength=length($mystring);
    #$inputMIC=$mystring;
125
    $distcount=0;
127
    foreach $psize (@psizearray){
129
    $pdistance=$pdistancearray[$distcount];
131 $pdistance=$pdistance-$psize; #calculating the real distance with the count of
        pointer size

133 if($pr eq 'y'){
     $myrps=@rps[$distcount]; #real pointer size
135  $pdistance=$pdistance+$myrps-$psize; #influence caused by reverse match
    }
137

139 $position=0;
    @sposition=();
141 #@sposition2=();
    @mystring=();
143 ##########################################################################
    ########################################given distance####################
145 ##########################################################################
    while($position <= $miclength-$psize-$pdistance-$psize){
147 $front=substr($inputMIC,$position,$psize);
    $back=substr($inputMIC,$position+$pdistance+$psize,$psize);
149
    if($pr eq 'y'){
151  $back = reverse $back;
     $back=~ tr/ATCG/TAGC/;
153 }

155 if($front eq $back){
     push(@sposition,$position);
157  push(@mystring,$front);
    }
159  $position++;
    }
161 $count=0;

163 print "\n Whether reverse? $pr \n";
    print "The pointer size is $psize \n";
165 print "The distance is $pdistance \n";

167 while($count < scalar(@sposition)){
     print " $mystring[$count] , $sposition[$count] \n";
169  $count++;
    }
171
    $distcount++;
173
    print "************************************************************";
175 print "************************************************************";
    print "************************************************************";
177 }
     return 0;
179 }
```

C.2 Program that ignores distance

```
1 #!/usr/bin/perl
  ##################################
3 # read input                     #
```

```
##############################
5
### command line version of input
7 #reverse order pointer?

9 #print "whether reversed? \n";
#chomp($pr = <>);
11 #print "whether reversed? $pr \n\n";

13 #$psize=0; @psizearray=();

15 #while($psize >= 0){
#print "Enter the pointer size: \n";
17 #$psize = <>;
#print "Enter the pointer size is $psize \n";
19 #push(@psizearray,$psize);
#}
21
#print "Enter the name of a file save MIC: ";
23 #$infileMIC = <>;
#open MICFILE, $infileMIC;
25
open(MYOUT, ">␣result.txt");
27 *STDOUT = *MYOUT;
print "The␣result:\n";
29 ######## command line version of input finish

31 #### example input
##########      36paw  ############
33 $pr='y';
@psizearray=(15,13,12,11,10,9,8,7,6,5,4);
35
$infileMIC="36pawMIC.fasta";
37
myscannd();
39
$pr='n';
41 @psizearray=(8,7,4,3,2);

43 $infileMIC="36pawMIC.fasta";

45 myscannd();

47 sub myscannd{

49 print "whether␣reversed?␣$pr␣\n";

51 $inputMIC;
##################################################
53 #    input MIC                                   #
##################################################
55 print "The␣MIC␣file␣is␣$infileMIC\n";
open MICFILE, $infileMIC;
57
$line=<MICFILE>;
59 $inputMIC="";
while(defined($line=<MICFILE>)){
61 chomp $line;
$inputMIC=$inputMIC.$line;
63 # combine the input to a single string
}
65
#print "\n the MIC is\n";
67 #print "$inputMIC"; #test

69 close MICFILE;

71
```

```
#print "Enter the string: \n";
#$mystring = <>;

$miclength=length($inputMIC);
print "the length of MIC is $miclength \n";

#$miclength=length($mystring);
#$inputMIC=$mystring;

#pop(@psizearray);  #the last item is -1 when use manual input#

foreach $psize (@psizearray){

$position=0;
@sposition=();
@sposition2=();
@mystring=();
#########################################################################
###################################without distance info
    ###############################################################################
$position1=0; $position2=1;
#$rplength=0; $repeats= "hello"; $rpp1=0; $rpp2=0;

while($position1 <= $miclength-$psize-$psize){
while($position2 <= $miclength-$psize){
$front=substr($inputMIC,$position1,$psize);
$back=substr($inputMIC,$position2,$psize);

if($pr eq 'y'){
 $back = reverse $back;
 $back=~ tr/ATCG/TAGC/;
}

if($front eq $back){
 push(@sposition,$position1);
 push(@sposition2,$position2);
 push(@mystring,$front);
# if(length($front)> $rplength)
# {
#    $repeats=$front;
#    $rpp1=$position1;
#    $rpp2=$position2;
#    $rplength=length($front);
# }
}

$position2++;
}
$position1++;
$position2=$position1+$psize;
}

$count=0;

print "\n Whether reverse? $pr \n";
print "The pointer size is $psize \n";

while($count < scalar(@sposition)){
 print " $mystring[$count] , $sposition[$count], $sposition2[$count] \n";
 $count++;
}

print "*********************************************************\n";
print "*********************************************************\n";
print "*********************************************************\n";
}
}
```

Appendix D

Result of the new algorithm suite

Table D.1: Result of KnotDS, and KnotDSe on all 13 micronuclear genes

			position				knotDS		knotDSe	
MIC	size	pointer	p1	p2	distance	reverse	left	right	left	right
38	12	cttactacacat	2007	2213	206	y	2374	2374	2374	2374
	13	cggagtcgtcaag	427	1958	1531	y	2374	2374	2374	2374
	4	aatc	916	948	32	n	139	75	2374	2374
	13	ctcccaagtccat	1382	1668	286	n	2374	2374	2374	2374
	6	agcccc	1410	1700	290	n	191	2374	2374	2374
	10	caaaactcta	1548	1725	177	n	2374	2374	2374	2374
	6	ggttga	1833	2245	412	n	116	2374	2374	2374
	11	aggttgaatga	1850	2255	405	n	2374	2374	2374	2374
35	4	tctc	341	406	65	n	989	293	989	989
1	11	agaccaacaaa	1818	2039	221	y	2115	2115	2115	2115
	12	aaggctggtttc	0	1774	1774	y	2115	2115	2115	2115
	4	tctc	341	409	68	n	2115	454	2115	2115
	13	agctcccaagtca	979	1218	239	n	2115	2115	2115	2115
	8	tattgcca	1015	1245	230	n	2115	2115	2115	2115
	8	tgaggaat	1154	1272	118	n	2115	2115	2115	2115
	10	tgaaacttaa	1355	2078	723	n	2115	2115	2115	2115
	11	gggttgaatga	1385	2104	719	n	2115	2115	2115	2115
	7	caaaaat	1445	1728	283	n	2115	2115	2115	2115
28	4	aatc	217	408	191	n	56	216	1600	1600
	12	cagagctcccaa	839	1079	240	n	1600	1600	1600	1600
	9	attgccagc	871	1110	239	n	1600	1600	1600	1600
	9	ctatcttta	1016	1141	125	n	1600	1600	1600	1600
32	8	caaagaaa	200	687	487	n	2756	2756	2756	2756
	8	ttgtcttg	251	965	714	n	2756	2756	2756	2756
	17	cattcacagacttggag	288	989	701	n	2756	2756	2756	2756
	5	agaat	312	1014	702	n	2756	511	2756	2756
	6	actcag	472	1046	574	n	2756	2756	2756	2756
	5	aatga	511	1075	564	n	2756	356	2756	2756
	11	ttgttcaaaac	524	1085	561	n	2756	2756	2756	2756
	3	aat	1115	1165	50	n	11	23	507	83
	8	agcttaag	558	1303	745	n	2756	2756	2756	2756
	11	tcaagttttct	577	1331	754	n	2756	2756	2756	2756
	7	aggttgt	598	1387	789	n	497	2756	2756	2756
	11	tcagctactta	615	1409	794	n	2756	2756	2756	2756
	7	aaaagaa	1437	1485	48	n	2756	1039	2756	2756
	7	acaagaa	1650	1722	72	n	2756	2756	2756	2756
	6	tcgtta	1956	2014	58	n	1814	198	2756	2756
	7	cagaatt	2243	2358	115	n	2756	2756	2756	2756
42	9	aacctcaaa	150	885	735	n	3205	3205	3205	3205
	5	ttccc	430	1044	614	n	379	1	3205	3205
	7	aaccaag	445	1057	612	n	3205	3205	3205	3205
	8	acgctaag	473	1163	690	n	3205	3205	3205	3205
	7	cagattg	507	1189	682	n	3205	3205	3205	3205
	8	tacaacca	530	1235	705	n	3205	3205	3205	3205
	7	cttctct	655	1271	616	n	3205	32	3205	3205
	8	aaaagaat	687	1303	616	n	3205	3205	3205	3205
	8	tagttcaa	705	1318	613	n	3205	3205	3205	3205
	10	caatacttta	725	1347	622	n	3205	3205	3205	3205
	7	gttattt	746	1364	618	n	3205	3205	3205	3205
	6	aatctg	771	1494	723	n	3205	185	3205	3205
	13	aaagttttaattt	1600	2170	570	n	3205	3205	3205	3205
	11	aattcaatata	2873	3051	178	n	3205	3205	3205	3205

Continued on next page

			position				knotDS		knotDSe	
MIC	size	pointer	p1	p2	distance	reverse	left	right	left	right
41	10	aaagagaggc	37	476	439	n	1834	1834	1834	1834
	3	aga	65	750	685	n	10	52	1834	1834
	2	ga	101	770	669	n	9	5	65	92
	4	gact	117	802	685	n	1834	427	1834	1834
	4	tttc	247	824	577	n	1834	9	1834	1834
	5	acacc	267	847	580	n	1834	358	1834	1834
	7	ttgttca	294	864	570	n	1834	1834	1834	1834
	10	ctcaacaata	312	888	576	n	1834	1834	1834	1834
	5	gattt	340	918	578	n	277	750	1834	1834
	10	ctaaagctta	357	1030	673	n	1834	1834	1834	1834
	9	actttcttc	384	1049	665	n	1834	1834	1834	1834
	5	taaga	402	1089	687	n	334	1834	1834	1834
	12	tctgctacttat	431	1110	679	n	1834	1834	1834	1834
	4	aagt	1144	1159	15	n	195	74	1834	1834
	2	aa	1674	1720	46	n	9	4	171	4
37	11	gaaggcgctgc	277	759	482	n	2700	2700	2700	2700
	5	gccac	327	1033	706	n	277	984	2700	2700
	4	tcat	370	1054	684	n	117	5	2700	2700
	12	agagctaccctc	411	1089	678	n	2700	2700	2700	2700
	9	tcaagcaag	564	1127	563	n	2700	2700	2700	2700
	10	ttgagaagaa	584	1146	562	n	2700	2700	2700	2700
	9	agaacctga	611	1182	571	n	2700	2700	2700	2700
	6	aaggac	640	1349	709	n	2700	787	2700	2700
	9	aagtgttct	656	1368	712	n	2700	2700	2700	2700
	6	agaact	673	1412	739	n	2700	2700	2700	2700
	14	agatcagccactta	697	1448	751	n	2700	2700	2700	2700
	5	cccaa	1476	1513	37	n	884	1012	2700	2700
	3	act	2224	2282	58	n	222	25	667	213
33	13	taaagacggcgca	211	792	581	n	2686	2686	2686	2686
	9	tagtcttat	341	1072	731	n	2686	2686	2686	2686
	5	ggaga	391	1153	762	n	2686	1516	2686	2686
	7	actctca	419	1180	761	n	2686	2686	2686	2686
	7	ccattcg	554	1208	654	n	2686	2686	2686	2686
	10	ttgagaagag	623	1239	616	n	2686	2686	2686	2686
	6	ccttct	642	1257	615	n	2686	2686	2686	2686
	11	ctcaagttgaa	668	1423	755	n	2686	2686	2686	2686
	9	aagttttct	693	1463	770	n	2686	2686	2686	2686
	7	ggagaag	719	1513	794	n	2686	2686	2686	2686
	13	atcagctacttat	739	1531	792	n	2686	2686	2686	2686
	3	aag	1565	1604	39	n	57	34	223	117
	3	aat	2114	2162	48	n	87	96	2008	366
44	2	ta	2619	2663	44	n	19	24	46	24
	9	aaaaatgat	2231	3464	1233	y	5656	5656	5656	5656
	12	tttccttctatt	2101	3492	1391	y	5656	5656	5656	5656
	6	aataat	2087	3524	1437	y	200	295	200	5656
	8	tatcttgt	2024	3543	1519	y	5656	5656	5656	5656
	4	ttta	1982	3622	1640	y	24	37	5656	5656
	7	aaataag	1957	3642	1685	y	215	578	5656	5656
	8	tgccaata	1920	3751	1831	y	1625	5656	5656	5656
	8	aaaattaa	1888	3768	1880	y	5656	5656	5656	5656
	8	tttgaatg	1869	3825	1956	y	5656	5656	5656	5656
	6	aagacc	1754	3878	2124	y	253	5656	5656	5656
	9	ttaaacaga	1717	3944	2227	y	5656	5656	5656	5656
	10	aatgccttta	1534	3973	2439	y	5656	5656	5656	5656
	6	tcttta	1506	4032	2526	y	1391	356	5656	5656
	8	ttattgct	1435	4046	2611	y	5656	5656	5656	5656
	13	aaataaaaagttt	1407	4158	2751	y	5656	5656	5656	5656
	6	ttcata	1375	4173	2798	y	5656	1156	5656	5656
	13	aagataagtaact	1354	4188	2834	y	5656	5656	5656	5656
	16	aaacagtatgtatggt	496	4258	3762	y	5656	5656	5656	5656
	7	aggcaga	483	4336	3853	y	5656	1192	5656	5656
	8	ctgagaat	447	4364	3917	y	5656	5656	5656	5656

Continued on next page

			position				knotDS		knotDSe	
MIC	size	pointer	p1	p2	distance	reverse	left	right	left	right
	7	actgact	435	4402	3967	y	5656	5656	5656	5656
	5	gtttt	317	4439	4122	y	5656	814	5656	5656
	13	atgttaaagttat	277	4512	4235	y	5656	5656	5656	5656
	4	ctag	200	4546	4346	y	8	24	5656	5656
	6	ttggtg	178	4616	4438	y	252	137	5656	5656
	8	caatatat	138	4639	4501	y	5656	5656	5656	5656
	12	aaaaagtgaagc	106	4740	4634	y	5656	5656	5656	5656
	7	gcttcat	85	4763	4678	y	5656	5656	5656	5656
	7	ctaagaa	49	4991	4942	y	5656	5656	5656	5656
	11	cagagaaaaag	24	5010	4986	y	5656	5656	5656	5656
	2	ta	5494	5539	45	n	6	64	8	5656
19	5	agata	2252	2302	50	n	1307	1245	6500	6500
	4	atta	2903	3113	210	n	595	437	6500	6500
	13	atgagtggaatta	1859	3407	1548	y	6500	6500	6500	6500
	4	aaca	1847	3453	1606	y	127	109	6500	6500
	9	agaaatatg	1813	3482	1669	y	6500	6500	6500	6500
	8	ttatcatt	1745	3515	1770	y	6500	2970	6500	6500
	8	aaaataat	1718	3546	1828	y	6500	6500	6500	6500
	8	gtttcttg	1656	3569	1913	y	6500	6500	6500	6500
	7	atgcaaa	1624	3596	1972	y	6500	568	6500	6500
	8	taaaatga	1597	3624	2027	y	6500	6500	6500	6500
	7	agaggag	1573	3643	2070	y	6500	6500	6500	6500
	10	taatgatgga	1524	3677	2153	y	6500	6500	6500	6500
	8	atggtgag	1489	3783	2294	y	905	6500	6500	6500
	8	aaaatcaa	1469	3810	2341	y	6500	6500	6500	6500
	12	aaagcatgcttg	1440	3892	2452	y	6500	6500	6500	6500
	7	aagaaaa	1356	3931	2575	y	6500	6500	6500	6500
	10	gttactcttg	1316	3985	2669	y	6500	6500	6500	6500
	12	agctcaataaaa	1240	4030	2790	y	6500	6500	6500	6500
	3	atc	1196	4063	2867	y	8	6	6500	1134
	7	aaaactt	1111	4089	2978	y	6500	6500	6500	6500
	10	gagagataga	1077	4173	3096	y	6500	6500	6500	6500
	9	tagttgctc	1020	4220	3200	y	6500	6500	6500	6500
	8	aagctaga	980	4339	3359	y	6500	6500	6500	6500
	8	ggaggatc	953	4393	3440	y	6500	6500	6500	6500
	8	caagataa	935	4412	3477	y	6500	6500	6500	6500
	15	ataagactttgatga	803	4463	3660	y	6500	6500	6500	6500
	8	ctaatgaa	191	250	59	n	6500	459	6500	6500
	2	ag	0	4541	4541	y	6500	13	6500	82
	6	ataaaa	6119	6194	75	n	113	6500	6500	6500
	5	tatat	6352	6364	12	n	327	6500	6500	6500
26	2	ta	2074	2116	42	n	123	44	131	223
	15	atctaagaatgatga	1697	2981	1284	y	6588	6588	6588	6588
	7	aataagc	1664	3020	1356	y	6588	1821	6588	6588
	8	aaaactat	1644	3040	1396	y	6588	6588	6588	6588
	7	attgatg	1573	3068	1495	y	6588	3300	6588	6588
	8	aaataatg	1556	3085	1529	y	6589	169	6588	6588
	5	taata	1505	3121	1616	y	49	86	6588	6588
	4	aatg	1489	3138	1649	y	175	76	6588	6588
	7	ctaaaat	1445	3172	1727	y	6588	1761	6588	6588
	4	agag	1425	3195	1770	y	139	135	6588	6588
	8	aaaatcaa	1396	3219	1823	y	6588	6588	6588	6588
	6	atggag	1354	3345	1991	y	387	1472	6588	6588
	12	cttgaaaatcaa	1332	3380	2048	y	6588	6588	6588	6588
	9	atgcttgaa	1300	3471	2171	y	6588	6588	6588	6588
	8	aagaaaac	1217	3506	2289	y	6588	6588	6588	6588
	8	atgatctt	1188	3570	2382	y	6588	6588	6588	6588
	7	aaatgag	1110	3593	2483	y	6588	2093	6588	6588
	9	tgtttggtt	1085	3615	2530	y	6588	6588	6588	6588
	8	gatggcaa	1005	3659	2654	y	6588	6588	6588	6588
	12	gaatatttaata	882	3712	2830	y	6588	6588	6588	6588
	8	gtggctca	805	3747	2942	y	6588	6588	6588	6588

Continued on next page

			position				knotDS		knotDSe	
MIC	size	pointer	p1	p2	distance	reverse	left	right	left	right
	7	agtttga	786	3865	3079	y	6588	6588	6588	6588
	7	gatccta	756	3897	3141	y	6588	6588	6588	6588
	7	aagataa	737	3913	3176	y	6588	6588	6588	6588
	10	aagtctagct	639	3946	3307	y	6588	6588	6588	6588
	13	ataagattttgat	561	3968	3407	y	6588	6588	6588	6588
	11	aagtatgctgg	199	5597	5398	y	6588	6588	6588	6588
	13	tgtatggttgctt	4019	5988	1969	n	6588	6588	6588	6588
	9	tagagaaac	4092	6034	1942	n	6588	6588	6588	6588
	4	ttgc	4116	6063	1947	n	205	422	6588	6588
	12	attatgatcaat	4166	6089	1923	n	6588	6588	6588	6588
	12	ttttcaagagtt	4209	6201	1992	n	6588	6588	6588	6588
	5	aatga	4277	6226	1949	n	1395	262	6588	6588
	7	gaaatta	4308	6305	1997	n	2338	6588	6588	6588
	10	atattggaga	4386	6323	1937	n	6588	6588	6588	6588
	7	atcaatt	4424	6362	1938	n	6588	6588	6588	6588
	6	gaaaga	4526	6394	1868	n	3977	6588	6588	6588
	8	aataattt	4546	6417	1871	n	6588	6588	6588	6588
	3	cat	4580	6453	1873	n	4	123	1503	6588
	3	aag	4592	6467	1875	n	114	34	1686	6588
	5	tggat	4834	6498	1664	n	570	6588	6588	6588
	9	aatcctatt	4849	6511	1662	n	6588	6588	6588	6588
	8	aattatga	4936	6553	1617	n	6588	6588	6588	6588
	7	aaggaaa	4964	6581	1617	n	27	6588	6588	6588
	5	tcatg	5253	5281	28	n	774	461	6588	6588
36	4	atgc	2752	2766	14	n	9	90	6930	6930
	4	aaat	2814	2832	18	n	92	33	288	6930
	4	ttaa	3283	3694	411	n	223	136	975	1557
	4	gaaa	3703	3928	225	n	5	80	6930	6930
	6	aagcag	2486	4267	1781	y	2482	986	6930	6930
	9	aaagcaaca	2457	4283	1826	y	6930	2290	6930	6930
	8	gattataa	2228	4308	2080	y	612	6930	6930	6930
	9	attatcttt	2153	4316	2163	y	6930	6930	6930	6930
	8	aaaataat	2135	4348	2213	y	6930	2221	6930	6930
	9	aatatgtct	2077	4367	2290	y	6930	6930	6930	6930
	9	agaaatata	2062	4407	2345	y	6930	6930	6930	6930
	4	aaat	2047	4416	2369	y	477	45	1765	91
	7	actctta	2028	4439	2411	y	6930	1854	6930	6930
	11	ataataagtta	2001	4450	2449	y	6930	6930	6930	6930
	6	tagcat	1986	4568	2582	y	6930	762	6930	6930
	6	aagtat	1956	4589	2633	y	6930	312	6930	6930
	5	aaaca	1949	4633	2684	y	313	425	6930	6930
	7	aatgctt	1936	4644	2708	y	6930	6930	6930	6930
	8	aaactaaa	1925	4660	2735	y	6930	6930	6930	6930
	9	aaaaacttg	1828	4677	2849	y	6930	6930	6930	6930
	11	aagttactctt	1796	4727	2931	y	6930	6930	6930	6930
	7	aaaataa	1719	4748	3029	y	1340	6930	6930	6930
	6	tatgat	1702	4770	3068	y	230	1417	6930	6930
	8	atttgatt	1592	4798	3206	y	6930	6930	6930	6930
	10	aatggtttta	1573	4842	3269	y	6930	6930	6930	6930
	4	tggt	1488	4863	3375	y	200	188	6930	6930
	10	attccaaata	1476	4951	3475	y	6930	6930	6930	6930
	12	gaggatcatagt	1434	4978	3544	y	6930	6930	6930	6930
	6	aagata	1422	4997	3575	y	124	283	6930	6930
	8	aaaattaa	580	5062	4482	y	6930	6930	6930	6930
	6	aagaga	554	5155	4601	y	6930	320	6930	6930
	7	ttgctga	524	5173	4649	y	6930	1120	6930	6930
	13	tattatgattaat	368	5222	4854	y	6930	6930	6930	6930
	10	gagtttttaa	261	5245	4984	y	6930	6930	6930	6930
	8	ttaaagta	248	5323	5075	y	6930	6930	6930	6930
	7	gtaatta	177	5343	5166	y	6930	997	6930	6930
	9	ataaaatga	156	5430	5274	y	6930	6930	6930	6930
	9	aataacttt	137	5445	5308	y	6930	6930	6930	6930

Continued on next page

			position				knotDS		knotDSe	
MIC	size	pointer	p1	p2	distance	reverse	left	right	left	right
	11	aaagtgaagct	120	5561	5441	y	6930	6930	6930	6930
	7	aacaatt	104	5578	5474	y	6930	399	6930	6930
	7	tgatatg	62	5862	5800	y	6930	6930	6930	6930
	6	tttgta	36	5889	5853	y	6930	94	6930	6930
	7	tgatttt	21	5973	5952	y	6930	6930	6930	6930
	7	agagggt	0	5988	5988	y	6930	6930	6930	6930
	7	aaattat	6108	6393	285	n	1143	6930	6930	6930
	2	ta	6510	6720	210	n	12	36	14	38
	3	aat	6781	6797	16	n	90	67	93	6930

Table D.2: Result of KnotD, and KnotDSe3 on all 13 micronuclear genes

			position				knotD		knotDSe3	
MIC	size	pointer	p1	p2	distance	reverse	left	right	left	right
38	12	cttactacacat	2007	2213	206	y	30	38	911	2374
	13	cggagtcgtcaag	427	1958	1531	y	78	17	2374	2374
	4	aatc	916	948	32	n	138	75	2374	452
	13	ctcccaagtccat	1382	1668	286	n	15	29	25	2374
	6	agcccc	1410	1700	290	n	119	14	1016	2374
	10	caaaactcta	1548	1725	177	n	14	15	1162	2374
	6	ggttga	1833	2245	412	n	4	66	2374	2374
	11	aggttgaatga	1850	2255	405	n	30	2374	477	2374
35	4	tctc	341	406	65	n	70	26	989	989
1	11	agaccaacaaa	1818	2039	221	y	15	2115	77	2115
	12	aaggctggtttc	0	1774	1774	y	2115	15	2115	300
	4	tctc	341	409	68	n	213	50	2115	455
	13	agctcccaagtca	979	1218	239	n	11	37	2115	2115
	8	tattgcca	1015	1245	230	n	100	59	824	59
	8	tgaggaat	1154	1272	118	n	47	9	2115	2115
	10	tgaaacttaa	1355	2078	723	n	66	19	2115	2115
	11	gggttgaatga	1385	2104	719	n	7	2115	2115	2115
	7	caaaaat	1445	1728	283	n	72	28	1399	361
28	4	aatc	217	408	191	n	55	54	1600	175
	12	cagagctcccaa	839	1079	240	n	234	120	1600	1600
	9	attgccagc	871	1110	239	n	4	101	1600	454
	9	ctatcttta	1016	1141	125	n	130	13	1600	1600
32	8	caaagaaa	200	687	487	n	4	54	2756	923
	8	ttgtcttg	251	965	714	n	119	13	2756	2756
	17	cattcacagacttggag	288	989	701	n	67	59	2756	2756
	5	agaat	312	1014	702	n	105	115	2756	1128
	6	actcag	472	1046	574	n	31	142	2756	2756
	5	aatga	511	1075	564	n	4	40	399	40
	11	ttgttcaaaac	524	1085	561	n	14	86	2756	263
	3	aat	1115	1165	50	n	11	23	507	83
	8	agcttaag	558	1303	745	n	72	76	2756	2756
	11	tcaagttttct	577	1331	754	n	13	16	2756	78
	7	aggttgt	598	1387	789	n	29	50	2756	104
	11	tcagctactta	615	1409	794	n	7	53	2756	2756
	7	aaaagaa	1437	1485	48	n	60	39	797	231
	7	acaagaa	1650	1722	72	n	74	49	434	156
	6	tcgtta	1956	2014	58	n	45	46	2756	2756
	7	cagaatt	2243	2358	115	n	63	27	2064	2756
42	9	aacctcaaa	150	885	735	n	11	60	3205	127
	5	ttccc	430	1044	614	n	4	136	3205	2024
	7	aaccaag	445	1057	612	n	46	14	3205	3205
	8	acgctaag	473	1163	690	n	110	10	3205	3205
	7	cagattg	507	1189	682	n	9	80	3205	3205
	8	tacaacca	530	1235	705	n	79	131	3205	3205
	7	cttctct	655	1271	616	n	14	23	271	1375

Continued on next page

			position				knotD		knotDSe3	
MIC	size	pointer	p1	p2	distance	reverse	left	right	left	right
	8	aaaagaat	687	1303	616	n	4	65	319	177
	8	tagttcaa	705	1318	613	n	48	77	3205	3205
	10	caatacttta	725	1347	622	n	11	163	287	3205
	7	gttattt	746	1364	618	n	10	13	3205	684
	6	aatctg	771	1494	723	n	4	68	3205	185
	13	aaagttttaattt	1600	2170	570	n	290	26	759	26
	11	aattcaatata	2873	3051	178	n	36	23	168	23
41	10	aaagagaggc	37	476	439	n	1834	171	1834	1174
	3	aga	65	750	685	n	10	52	1834	1834
	2	ga	101	770	669	n	9	5	65	92
	4	gact	117	802	685	n	52	8	1834	1834
	4	tttc	247	824	577	n	65	5	1834	1834
	5	acacc	267	847	580	n	85	10	1834	1834
	7	ttgttca	294	864	570	n	34	47	1834	1834
	10	ctcaacaata	312	888	576	n	65	60	1834	1834
	5	gattt	340	918	578	n	76	100	205	1834
	10	ctaaagctta	357	1030	673	n	4	46	1834	1834
	9	actttcttc	384	1049	665	n	96	110	1834	1834
	5	taaga	402	1089	687	n	39	59	58	629
	12	tctgctacttat	431	1110	679	n	66	87	1834	1834
	4	aagt	1144	1159	15	n	60	9	195	77
	2	aa	1674	1720	46	n	9	4	171	4
37	11	gaaggcgctgc	277	759	482	n	110	46	2700	2700
	5	gccac	327	1033	706	n	82	6	2700	2700
	4	tcat	370	1054	684	n	116	5	2700	2700
	12	agagctaccctc	411	1089	678	n	34	164	151	659
	9	tcaagcaag	564	1127	563	n	47	18	2700	521
	10	ttgagaagaa	584	1146	562	n	34	42	2700	2700
	9	agaacctga	611	1182	571	n	56	99	112	2700
	6	aaggac	640	1349	709	n	50	24	230	134
	9	aagtgttct	656	1368	712	n	49	95	362	115
	6	agaact	673	1412	739	n	12	166	441	2700
	14	agatcagccactta	697	1448	751	n	4	142	258	2700
	5	cccaa	1476	1513	37	n	243	140	583	2700
	3	act	2224	2282	58	n	222	25	667	213
33	13	taaagacggcgca	211	792	581	n	182	14	2686	46
	9	tagtcttat	341	1072	731	n	35	96	2686	200
	5	ggaga	391	1153	762	n	4	57	2686	2686
	7	actctca	419	1180	761	n	4	72	2686	155
	7	ccattcg	554	1208	654	n	6	24	2686	2686
	10	ttgagaagag	623	1239	616	n	21	74	2686	2686
	6	ccttct	642	1257	615	n	109	169	2686	594
	11	ctcaagttgaa	668	1423	755	n	74	61	2686	2686
	9	aagttttct	693	1463	770	n	73	23	2686	115
	7	ggagaag	719	1513	794	n	64	121	2686	2686
	13	atcagctacttat	739	1531	792	n	66	30	2686	2686
	3	aag	1565	1604	39	n	57	34	223	117
	3	aat	2114	2162	48	n	87	96	2008	366
44	2	ta	2619	2663	44	n	19	24	46	24
	9	aaaaatgat	2231	3464	1233	y	3	14	465	5656
	12	tttccttctatt	2101	3492	1391	y	30	11	983	1496
	6	aataat	2087	3524	1437	y	47	10	200	174
	8	tatcttgt	2024	3543	1519	y	24	33	5656	5656
	4	ttta	1982	3622	1640	y	7	37	506	357
	7	aaataag	1957	3642	1685	y	57	22	598	499
	8	tgccaata	1920	3751	1831	y	72	109	278	153
	8	aaaattaa	1888	3768	1880	y	7	47	174	435
	8	tttgaatg	1869	3825	1956	y	4	18	1164	661
	6	aagacc	1754	3878	2124	y	11	30	5656	5656
	9	ttaaacaga	1717	3944	2227	y	28	177	269	239
	10	aatgccttta	1534	3973	2439	y	113	19	390	55
	6	tcttta	1506	4032	2526	y	32	78	66	496

Continued on next page

			position				knotD		knotDSe3	
MIC	size	pointer	p1	p2	distance	reverse	left	right	left	right
	8	ttattgct	1435	4046	2611	y	10	9	5656	5656
	13	aaataaaaagttt	1407	4158	2751	y	39	8	1089	895
	6	ttcata	1375	4173	2798	y	11	36	532	44
	13	aagataagtaact	1354	4188	2834	y	9	19	5656	208
	16	aaacagtatgtatggt	496	4258	3762	y	22	53	5656	5656
	7	aggcaga	483	4336	3853	y	27	106	433	5656
	8	ctgagaat	447	4364	3917	y	53	16	148	414
	7	actgact	435	4402	3967	y	43	12	5656	5656
	5	gtttt	317	4439	4122	y	16	53	5656	561
	13	atgttaaagttat	277	4512	4235	y	10	54	5656	5656
	4	ctag	200	4546	4346	y	8	24	5656	876
	6	ttggtg	178	4616	4438	y	7	30	5656	5656
	8	caatatat	138	4639	4501	y	16	21	5656	306
	12	aaaaagtgaagc	106	4740	4634	y	12	16	5656	5656
	7	gcttcat	85	4763	4678	y	11	203	5656	203
	7	ctaagaa	49	4991	4942	y	5656	65	5656	5656
	11	cagagaaaaag	24	5010	4986	y	17	22	5656	5656
	2	ta	5494	5539	45	n	6	64	8	5656
19	5	agata	2252	2302	50	n	40	30	644	1230
	4	atta	2903	3113	210	n	231	11	760	598
	13	atgagtggaatta	1859	3407	1548	y	27	27	1148	98
	4	aaca	1847	3453	1606	y	3	92	674	6500
	9	agaaatatg	1813	3482	1669	y	40	13	1472	1325
	8	ttatcatt	1745	3515	1770	y	9	4	518	496
	8	aaaataat	1718	3546	1828	y	34	40	202	425
	8	gtttcttg	1656	3569	1913	y	12	29	6500	2746
	7	atgcaaa	1624	3596	1972	y	34	47	633	568
	8	taaaatga	1597	3624	2027	y	10	40	733	6500
	7	agaggag	1573	3643	2070	y	15	20	6500	6500
	10	taatgatgga	1524	3677	2153	y	100	19	6500	6500
	8	atggtgag	1489	3783	2294	y	264	34	1377	606
	8	aaaatcaa	1469	3810	2341	y	15	112	77	156
	12	aaagcatgcttg	1440	3892	2452	y	36	33	6500	360
	7	aagaaaa	1356	3931	2575	y	22	36	770	438
	10	gttactcttg	1316	3985	2669	y	37	64	6500	6500
	12	agctcaataaaa	1240	4030	2790	y	13	86	763	1021
	3	atc	1196	4063	2867	y	8	6	6500	1134
	7	aaaactt	1111	4089	2978	y	10	37	6500	1190
	10	gagagataga	1077	4173	3096	y	61	20	199	775
	9	tagttgctc	1020	4220	3200	y	18	5	6500	6500
	8	aagctaga	980	4339	3359	y	48	8	168	442
	8	ggaggatc	953	4393	3440	y	46	135	6500	6500
	8	caagataa	935	4412	3477	y	55	62	55	521
	15	ataagactttgatga	803	4463	3660	y	56	58	6500	794
	8	ctaatgaa	191	250	59	n	6	69	6500	365
	2	ag	0	4541	4541	y	6500	13	6500	82
	6	ataaaa	6119	6194	75	n	75	13	75	159
	5	tatat	6352	6364	12	n	28	14	1258	14
26	2	ta	2074	2116	42	n	123	44	131	223
	15	atctaagaatgatga	1697	2981	1284	y	26	18	1012	450
	7	aataagc	1664	3020	1356	y	7	17	6588	2030
	8	aaaactat	1644	3040	1396	y	49	41	100	1365
	7	attgatg	1573	3068	1495	y	26	82	99	2919
	8	aaataatg	1556	3085	1529	y	29	52	98	87
	5	taata	1505	3121	1616	y	21	86	623	793
	4	aatg	1489	3138	1649	y	18	10	6588	6588
	7	ctaaaat	1445	3172	1727	y	33	19	251	30
	4	agag	1425	3195	1770	y	13	10	6588	6588
	8	aaaatcaa	1396	3219	1823	y	60	25	6588	2873
	6	atggag	1354	3345	1991	y	18	117	6588	2965
	12	cttgaaaatcaa	1332	3380	2048	y	15	16	6588	29
	9	atgcttgaa	1300	3471	2171	y	77	16	775	2862

Continued on next page

			position				knotD		knotDSe3	
MIC	size	pointer	p1	p2	distance	reverse	left	right	left	right
	8	aagaaaac	1217	3506	2289	y	136	41	6588	6588
	8	atgatctt	1188	3570	2382	y	34	18	6588	38
	7	aaatgag	1110	3593	2483	y	21	82	6588	6588
	9	tgtttggtt	1085	3615	2530	y	45	18	6588	6588
	8	gatggcaa	1005	3659	2654	y	12	171	6588	843
	12	gaatatttaata	882	3712	2830	y	16	22	6588	688
	8	gtggctca	805	3747	2942	y	12	44	6588	657
	7	agtttga	786	3865	3079	y	17	52	6588	325
	7	gatccta	756	3897	3141	y	63	28	6588	6588
	7	aagataa	737	3913	3176	y	25	36	329	821
	10	aagtctagct	639	3946	3307	y	51	73	6588	6588
	13	ataagattttgat	561	3968	3407	y	147	18	6588	6588
	11	aagtatgctgg	199	5597	5398	y	58	22	6588	722
	13	tgtatggttgctt	4019	5988	1969	n	22	220	6588	6588
	9	tagagaaac	4092	6034	1942	n	13	122	6588	6588
	4	ttgc	4116	6063	1947	n	16	120	3998	6588
	12	attatgatcaat	4166	6089	1923	n	83	34	673	401
	12	ttttcaagagtt	4209	6201	1992	n	42	29	527	6588
	5	aatga	4277	6226	1949	n	25	9	1074	6588
	7	gaaatta	4308	6305	1997	n	54	20	2772	279
	10	atattggaga	4386	6323	1937	n	6	79	1185	153
	7	atcaatt	4424	6362	1938	n	5	13	3233	6588
	6	gaaaga	4526	6394	1868	n	7	23	1593	6588
	8	aataattt	4546	6417	1871	n	26	40	660	6588
	3	cat	4580	6453	1873	n	4	123	1503	6588
	3	aag	4592	6467	1875	n	114	34	1686	6588
	5	tggat	4834	6498	1664	n	4	34	6588	6588
	9	aatcctatt	4849	6511	1662	n	91	10	91	6588
	8	aattatga	4936	6553	1617	n	21	19	993	6588
	7	aaggaaa	4964	6581	1617	n	4	6588	1100	6588
	5	tcatg	5253	5281	28	n	14	106	775	645
36	4	atgc	2752	2766	14	n	8	90	6930	3100
	4	aaat	2814	2832	18	n	91	33	183	139
	4	ttaa	3283	3694	411	n	172	5	870	138
	4	gaaa	3703	3928	225	n	4	80	957	563
	6	aagcag	2486	4267	1781	y	5	48	1429	1047
	9	aaagcaaca	2457	4283	1826	y	67	26	6930	6930
	8	gattataa	2228	4308	2080	y	29	54	53	138
	9	attatcttt	2153	4316	2163	y	4	18	313	556
	8	aaaataat	2135	4348	2213	y	99	6	566	211
	9	aatatgtct	2077	4367	2290	y	15	15	6930	735
	9	agaaatata	2062	4407	2345	y	15	133	351	133
	4	aaat	2047	4416	2369	y	36	20	1765	91
	7	actctta	2028	4439	2411	y	59	70	947	397
	11	ataataagtta	2001	4450	2449	y	37	12	67	181
	6	tagcat	1986	4568	2582	y	134	18	230	6930
	6	aagtat	1956	4589	2633	y	63	64	682	575
	5	aaaca	1949	4633	2684	y	51	4	6930	6930
	7	aatgctt	1936	4644	2708	y	72	16	6930	6930
	8	aaactaaa	1925	4660	2735	y	15	18	278	98
	9	aaaaacttg	1828	4677	2849	y	108	21	1761	6930
	11	aagttactctt	1796	4727	2931	y	57	29	6930	901
	7	aaaataa	1719	4748	3029	y	6	23	298	59
	6	tatgat	1702	4770	3068	y	39	8	6930	6930
	8	atttgatt	1592	4798	3206	y	8	16	964	35
	10	aatggtttta	1573	4842	3269	y	36	30	110	338
	4	tggt	1488	4863	3375	y	174	75	6930	6930
	10	attccaaata	1476	4951	3475	y	49	84	49	891
	12	gaggatcatagt	1434	4978	3544	y	79	28	6930	6930
	6	aagata	1422	4997	3575	y	58	14	891	14
	8	aaaattaa	580	5062	4482	y	49	47	286	895
	6	aagaga	554	5155	4601	y	12	28	6930	86

Continued on next page

			position				knotD		knotDSe3	
MIC	size	pointer	p1	p2	distance	reverse	left	right	left	right
	7	ttgctga	524	5173	4649	y	54	53	6930	6930
	13	tattatgattaat	368	5222	4854	y	49	27	92	113
	10	gagtttttaa	261	5245	4984	y	18	22	69	108
	8	ttaaagta	248	5323	5075	y	9	34	6930	6930
	7	gtaatta	177	5343	5166	y	13	4	6930	1160
	9	ataaaatga	156	5430	5274	y	26	6	6930	69
	9	aataacttt	137	5445	5308	y	54	22	6930	1164
	11	aaagtgaagct	120	5561	5441	y	6930	56	6930	872
	7	aacaatt	104	5578	5474	y	31	21	81	808
	7	tgatatg	62	5862	5800	y	6930	17	6930	346
	6	tttgta	36	5889	5853	y	27	45	6930	6930
	7	tgatttt	21	5973	5952	y	15	6	6930	177
	7	agagggt	0	5988	5988	y	6930	15	6930	6930
	7	aaattat	6108	6393	285	n	26	68	86	373
	2	ta	6510	6720	210	n	12	36	14	38
	3	aat	6781	6797	16	n	90	67	93	6930

Table D.3: Result of KnotS, and KnotSe on all 13 micronuclear genes

			position				knotS		knotSe	
MIC	size	pointer	p1	p2	distance	reverse	left	right	left	right
38	12	cttactacacat	2007	2213	206	y	2374	2374	2374	2374
	13	cggagtcgtcaag	427	1958	1531	y	2374	2374	2374	2374
	4	aatc	916	948	32	n	1	1	32	32
	13	ctcccaagtccat	1382	1668	286	n	2374	2374	2374	2374
	6	agcccc	1410	1700	290	n	1	2	205	290
	10	caaaactcta	1548	1725	177	n	8	148	2374	2374
	6	ggttga	1833	2245	412	n	4	1	394	11
	11	aggttgaatga	1850	2255	405	n	382	2374	2374	2374
35	4	tctc	341	406	65	n	1	1	65	28
1	11	agaccaacaaa	1818	2039	221	y	48	2115	2115	2115
	12	aaggctggtttc	0	1774	1774	y	2115	216	2115	2115
	4	tctc	341	409	68	n	1	1	68	28
	13	agctcccaagtca	979	1218	239	n	2115	535	2115	2115
	8	tattgcca	1015	1245	230	n	4	24	2115	2115
	8	tgaggaat	1154	1272	118	n	27	8	2115	2115
	10	tgaaacttaa	1355	2078	723	n	75	26	2115	2115
	11	gggttgaatga	1385	2104	719	n	130	267	2115	2115
	7	caaaaat	1445	1728	283	n	7	1	2115	2115
28	4	aatc	217	408	191	n	1	1	21	92
	12	cagagctcccaa	839	1079	240	n	1600	1600	1600	1600
	9	attgccagc	871	1110	239	n	21	31	1600	1600
	9	ctatcttta	1016	1141	125	n	31	33	1600	1600
32	8	caaagaaa	200	687	487	n	1	2	2756	436
	8	ttgtcttg	251	965	714	n	5	15	2756	667
	17	cattcacagacttggag	288	989	701	n	2756	2756	2756	2756
	5	agaat	312	1014	702	n	1	1	408	294
	6	actcag	472	1046	574	n	1	1	2756	33
	5	aatga	511	1075	564	n	1	1	355	209
	11	ttgttcaaaac	524	1085	561	n	90	53	2756	2756
	3	aat	1115	1165	50	n	1	1	12	12
	8	agcttaag	558	1303	745	n	1	19	2756	316
	11	tcaagttttct	577	1331	754	n	53	38	2756	2756
	7	aggttgt	598	1387	789	n	9	2	2756	2756
	11	tcagctactta	615	1409	794	n	38	637	2756	2756
	7	aaaagaa	1437	1485	48	n	3	1	48	2756
	7	acaagaa	1650	1722	72	n	1	1	72	72
	6	tcgtta	1956	2014	58	n	1	3	2756	2756
	7	cagaatt	2243	2358	115	n	8	10	2756	2756

Continued on next page

			position				knotS		knotSe	
MIC	size	pointer	p1	p2	distance	reverse	left	right	left	right
42	9	aacctcaaa	150	885	735	n	12	7	3205	3205
	5	ttccc	430	1044	614	n	1	1	394	484
	7	aaccaag	445	1057	612	n	5	1	3205	1760
	8	acgctaag	473	1163	690	n	11	8	3205	3205
	7	cagattg	507	1189	682	n	8	4	3205	3205
	8	tacaacca	530	1235	705	n	7	6	3205	3205
	7	cttctct	655	1271	616	n	4	5	3205	3205
	8	aaaagaat	687	1303	616	n	4	15	3205	3205
	8	tagttcaa	705	1318	613	n	15	3	3205	3205
	10	caatacttta	725	1347	622	n	65	60	3205	3205
	7	gttattt	746	1364	618	n	7	1	3205	3205
	6	aatctg	771	1494	723	n	1	4	3205	3205
	13	aaagttttaattt	1600	2170	570	n	32	204	3205	3205
	11	aattcaatata	2873	3051	178	n	355	47	3205	3205
41	10	aaagagaggc	37	476	439	n	32	182	1834	1834
	3	aga	65	750	685	n	1	1	2	7
	2	ga	101	770	669	n	1	1	5	2
	4	gact	117	802	685	n	1	1	522	163
	4	tttc	247	824	577	n	1	1	67	6
	5	acacc	267	847	580	n	1	1	1834	555
	7	ttgttca	294	864	570	n	3	3	1834	1834
	10	ctcaacaata	312	888	576	n	180	9	1834	1834
	5	gattt	340	918	578	n	1	1	163	52
	10	ctaaagctta	357	1030	673	n	35	80	1834	1834
	9	actttcttc	384	1049	665	n	18	24	1834	1834
	5	taaga	402	1089	687	n	1	1	398	230
	12	tctgctacttat	431	1110	679	n	1834	1834	1834	1834
	4	aagt	1144	1159	15	n	1	1	15	66
	2	aa	1674	1720	46	n	1	1	1	4
37	11	gaaggcgctgc	277	759	482	n	2700	134	2700	2700
	5	gccac	327	1033	706	n	1	1	330	106
	4	tcat	370	1054	684	n	1	1	140	21
	12	agagctaccctc	411	1089	678	n	2700	286	2700	2700
	9	tcaagcaag	564	1127	563	n	35	2	2700	2700
	10	ttgagaagaa	584	1146	562	n	55	7	2700	2700
	9	agaacctga	611	1182	571	n	19	9	2700	2700
	6	aaggac	640	1349	709	n	1	1	2700	568
	9	aagtgttct	656	1368	712	n	23	2	2700	2700
	6	agaact	673	1412	739	n	1	8	2700	361
	14	agatcagccactta	697	1448	751	n	2700	2700	2700	2700
	5	cccaa	1476	1513	37	n	1	1	37	2700
	3	act	2224	2282	58	n	1	1	54	4
33	13	taaagacggcgca	211	792	581	n	2686	528	2686	2686
	9	tagtcttat	341	1072	731	n	8	114	2686	2686
	5	ggaga	391	1153	762	n	1	1	173	207
	7	actctca	419	1180	761	n	4	6	2686	2686
	7	ccattcg	554	1208	654	n	4	14	2686	2686
	10	ttgagaagag	623	1239	616	n	17	30	2686	2686
	6	ccttct	642	1257	615	n	1	2	2686	2686
	11	ctcaagttgaa	668	1423	755	n	3	63	2686	2686
	9	aagttttct	693	1463	770	n	23	18	2686	2686
	7	ggagaag	719	1513	794	n	5	1	2686	2686
	13	atcagctacttat	739	1531	792	n	528	2686	2686	2686
	3	aag	1565	1604	39	n	1	1	3	13
	3	aat	2114	2162	48	n	1	1	5	13
44	2	ta	2619	2663	44	n	1	1	2	9
	9	aaaaatgat	2231	3464	1233	y	1	3	5656	5656
	12	tttccttctatt	2101	3492	1391	y	9	5	5656	5656
	6	aataat	2087	3524	1437	y	1	1	352	440
	8	tatcttgt	2024	3543	1519	y	7	9	5656	5656
	4	ttta	1982	3622	1640	y	1	1	33	36
	7	aaataag	1957	3642	1685	y	2	1	5656	873

Continued on next page

			position				knotS		knotSe	
MIC	size	pointer	p1	p2	distance	reverse	left	right	left	right
	8	tgccaata	1920	3751	1831	y	12	3	5656	5656
	8	aaaattaa	1888	3768	1880	y	2	1	1124	5656
	8	tttgaatg	1869	3825	1956	y	17	6	5656	5656
	6	aagacc	1754	3878	2124	y	1	1	5656	5656
	9	ttaaacaga	1717	3944	2227	y	6	4	5656	5656
	10	aatgccttta	1534	3973	2439	y	3	19	5656	5656
	6	tcttta	1506	4032	2526	y	1	1	5656	184
	8	ttattgct	1435	4046	2611	y	1	4	5656	5656
	13	aaataaaaagttt	1407	4158	2751	y	53	30	5656	5656
	6	ttcata	1375	4173	2798	y	1	4	5656	90
	13	aagataagtaact	1354	4188	2834	y	30	53	5656	5656
	16	aaacagtatgtatggt	496	4258	3762	y	5656	5656	5656	5656
	7	aggcaga	483	4336	3853	y	3	3	5656	5656
	8	ctgagaat	447	4364	3917	y	3	14	5656	3245
	7	actgact	435	4402	3967	y	2	4	5656	5656
	5	gtttt	317	4439	4122	y	1	1	158	1956
	13	atgttaaagttat	277	4512	4235	y	123	219	5656	5656
	4	ctag	200	4546	4346	y	1	1	102	25
	6	ttggtg	178	4616	4438	y	1	1	2967	57
	8	caatatat	138	4639	4501	y	3	3	5656	2402
	12	aaaaagtgaagc	106	4740	4634	y	163	48	5656	5656
	7	gcttcat	85	4763	4678	y	1	1	5656	2507
	7	ctaagaa	49	4991	4942	y	1	1	1778	1505
	11	cagagaaaaag	24	5010	4986	y	220	82	5656	5656
	2	ta	5494	5539	45	n	1	1	8	13
19	5	agata	2252	2302	50	n	1	1	50	50
	4	atta	2903	3113	210	n	1	1	42	6
	13	atgagtggaatta	1859	3407	1548	y	120	967	6500	6500
	4	aaca	1847	3453	1606	y	1	1	5	92
	9	agaaatatg	1813	3482	1669	y	3	8	6500	6500
	8	ttatcatt	1745	3515	1770	y	1	7	6500	6500
	8	aaaataat	1718	3546	1828	y	1	1	1494	648
	8	gtttcttg	1656	3569	1913	y	4	6	6500	6500
	7	atgcaaa	1624	3596	1972	y	1	1	6500	6500
	8	taaaatga	1597	3624	2027	y	1	1	6500	1489
	7	agaggag	1573	3643	2070	y	2	1	6500	6500
	10	taatgatgga	1524	3677	2153	y	9	16	6500	6500
	8	atggtgag	1489	3783	2294	y	2	5	6500	6500
	8	aaaatcaa	1469	3810	2341	y	3	1	414	6500
	12	aaagcatgcttg	1440	3892	2452	y	200	138	6500	6500
	7	aagaaaa	1356	3931	2575	y	1	1	242	854
	10	gttactcttg	1316	3985	2669	y	56	18	6500	6500
	12	agctcaataaaa	1240	4030	2790	y	138	87	6500	6500
	3	atc	1196	4063	2867	y	1	1	3	28
	7	aaaactt	1111	4089	2978	y	2	2	633	6500
	10	gagagataga	1077	4173	3096	y	3	17	6500	6500
	9	tagttgctc	1020	4220	3200	y	12	20	6500	6500
	8	aagctaga	980	4339	3359	y	1	1	6500	6500
	8	ggaggatc	953	4393	3440	y	9	6	6500	6500
	8	caagataa	935	4412	3477	y	13	3	6500	6500
	15	ataagactttgatga	803	4463	3660	y	6500	445	6500	6500
	8	ctaatgaa	191	250	59	n	9	6	6500	6500
	2	ag	0	4541	4541	y	1	1	31	14
	6	ataaaa	6119	6194	75	n	1	1	75	6500
	5	tatat	6352	6364	12	n	1	1	12	12
26	2	ta	2074	2116	42	n	1	1	18	14
	15	atctaagaatgatga	1697	2981	1284	y	1513	2642	6588	6588
	7	aataagc	1664	3020	1356	y	1	2	6588	6588
	8	aaaactat	1644	3040	1396	y	10	4	430	6588
	7	attgatg	1573	3068	1495	y	1	2	6588	823
	8	aaataatg	1556	3085	1529	y	3	2	6588	6588
	5	taata	1505	3121	1616	y	1	1	9	81

Continued on next page

			position				knotS		knotSe	
MIC	size	pointer	p1	p2	distance	reverse	left	right	left	right
	4	aatg	1489	3138	1649	y	1	1	49	18
	7	ctaaaat	1445	3172	1727	y	1	1	1058	6588
	4	agag	1425	3195	1770	y	1	1	15	38
	8	aaaatcaa	1396	3219	1823	y	1	1	64	165
	6	atggag	1354	3345	1991	y	1	1	53	286
	12	cttgaaaatcaa	1332	3380	2048	y	250	57	6588	6588
	9	atgcttgaa	1300	3471	2171	y	20	7	94	6588
	8	aagaaaac	1217	3506	2289	y	1	1	6588	6588
	8	atgatctt	1188	3570	2382	y	5	3	6588	6588
	7	aaatgag	1110	3593	2483	y	1	1	6588	217
	9	tgtttggtt	1085	3615	2530	y	8	14	6588	6588
	8	gatggcaa	1005	3659	2654	y	1	1	6588	6588
	12	gaatatttaata	882	3712	2830	y	320	250	6588	6588
	8	gtggctca	805	3747	2942	y	7	11	6588	6588
	7	agtttga	786	3865	3079	y	1	2	6588	6588
	7	gatccta	756	3897	3141	y	1	3	6588	6588
	7	aagataa	737	3913	3176	y	1	1	6588	6588
	10	aagtctagct	639	3946	3307	y	37	22	6588	6588
	13	ataagattttgat	561	3968	3407	y	375	985	6588	6588
	11	aagtatgctgg	199	5597	5398	y	11	62	6588	6588
	13	tgtatggttgctt	4019	5988	1969	n	6588	6588	6588	6588
	9	tagagaaac	4092	6034	1942	n	8	15	6588	6588
	4	ttgc	4116	6063	1947	n	1	1	49	386
	12	attatgatcaat	4166	6089	1923	n	100	1	6588	6588
	12	ttttcaagagtt	4209	6201	1992	n	43	755	6588	6588
	5	aatga	4277	6226	1949	n	1	1	311	23
	7	gaaatta	4308	6305	1997	n	2	2	6588	1757
	10	atattggaga	4386	6323	1937	n	95	11	6588	6588
	7	atcaatt	4424	6362	1938	n	1	1	130	469
	6	gaaaga	4526	6394	1868	n	1	1	567	1015
	8	aataattt	4546	6417	1871	n	1	1	539	6588
	3	cat	4580	6453	1873	n	1	1	26	85
	3	aag	4592	6467	1875	n	1	1	17	24
	5	tggat	4834	6498	1664	n	1	1	837	827
	9	aatcctatt	4849	6511	1662	n	13	19	6588	6588
	8	aattatga	4936	6553	1617	n	4	1	246	1371
	7	aaggaaa	4964	6581	1617	n	2	1	441	936
	5	tcatg	5253	5281	28	n	1	1	28	28
36	4	atgc	2752	2766	14	n	1	1	14	14
	4	aaat	2814	2832	18	n	1	1	18	18
	4	ttaa	3283	3694	411	n	1	1	54	50
	4	gaaa	3703	3928	225	n	1	1	46	94
	6	aagcag	2486	4267	1781	y	1	1	6930	1359
	9	aaagcaaca	2457	4283	1826	y	5	25	6930	6930
	8	gattataa	2228	4308	2080	y	20	1	6930	1175
	9	attatcttt	2153	4316	2163	y	4	32	6930	6930
	8	aaaataat	2135	4348	2213	y	3	1	1823	161
	9	aatatgtct	2077	4367	2290	y	12	6	6930	6930
	9	agaaatata	2062	4407	2345	y	4	3	6930	1846
	4	aaat	2047	4416	2369	y	1	1	7	18
	7	actctta	2028	4439	2411	y	1	1	6930	2293
	11	ataataagtta	2001	4450	2449	y	4	37	6930	6930
	6	tagcat	1986	4568	2582	y	1	1	1533	266
	6	aagtat	1956	4589	2633	y	1	1	6930	737
	5	aaaca	1949	4633	2684	y	1	1	213	422
	7	aatgctt	1936	4644	2708	y	1	4	839	6930
	8	aaactaaa	1925	4660	2735	y	3	4	6930	6930
	9	aaaaacttg	1828	4677	2849	y	10	5	6930	6930
	11	aagttactctt	1796	4727	2931	y	86	10	6930	6930
	7	aaaataa	1719	4748	3029	y	1	2	400	221
	6	tatgat	1702	4770	3068	y	1	1	178	455
	8	atttgatt	1592	4798	3206	y	1	1	128	1926

Continued on next page

			position				knotS		knotSe	
MIC	size	pointer	p1	p2	distance	reverse	left	right	left	right
	10	aatggtttta	1573	4842	3269	y	5	6	6930	6930
	4	tggt	1488	4863	3375	y	1	1	19	66
	10	attccaaata	1476	4951	3475	y	9	13	6930	6930
	12	gaggatcatagt	1434	4978	3544	y	12	96	6930	6930
	6	aagata	1422	4997	3575	y	1	1	938	139
	8	aaaattaa	580	5062	4482	y	1	1	2200	199
	6	aagaga	554	5155	4601	y	1	1	344	111
	7	ttgctga	524	5173	4649	y	1	2	3795	6930
	13	tattatgattaat	368	5222	4854	y	148	197	6930	6930
	10	gagtttttaa	261	5245	4984	y	4	5	6930	6930
	8	ttaaagta	248	5323	5075	y	2	2	6930	1320
	7	gtaatta	177	5343	5166	y	1	1	6930	1252
	9	ataaaatga	156	5430	5274	y	1	1	1017	6930
	9	aataacttt	137	5445	5308	y	2	6	3600	6930
	11	aaagtgaagct	120	5561	5441	y	93	11	6930	6930
	7	aacaatt	104	5578	5474	y	1	1	631	1610
	7	tgatatg	62	5862	5800	y	1	2	5480	149
	6	tttgta	36	5889	5853	y	1	1	6930	834
	7	tgatttt	21	5973	5952	y	3	1	6930	1818
	7	agagggt	0	5988	5988	y	6	4	6930	6930
	7	aaattat	6108	6393	285	n	1	1	177	108
	2	ta	6510	6720	210	n	1	1	2	3
	3	aat	6781	6797	16	n	1	1	12	16