Formal Model and Simulation of the Gene Assembly Process in Ciliates

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

By
MD. Sowgat Ibne Mahmud

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

The construction process of the functional macronucleus in certain types of ciliates is known as the ciliate gene assembly process. It consists of a massive amount of DNA excision from the micronucleus and the rearrangement of the rest of the DNA sequences (in the case of stichotrichous ciliates). While several computational models have tried to represent certain parts of the gene assembly process, the real process remains not completely understood. In this research, a new formal model called the *Computational 2JLP* model is introduced based on the recent biological 2JLP model.

For justifying the formal model, a simulation is created and tested with real data. Several parameters are introduced in the model that are used to test ambiguities or edge cases of the biological model. Parameters are systematically tested from the simulation to try to find their optimal values. Interestingly, a negative correlation is found between a parameter (which is used to filter out scnRNAs that are similar to IES specific sequences from the macronucleus) and the outcome of the simulation. It indicates that if a scnRNA consists of both an MDS and IES, then from the perspective of maximizing the outcome of the simulation, it is desirable to filter out this scnRNA.

The simulator successfully performs the gene assembly process whether the inputs are scrambled or unscrambled DNA sequences. It is desirable for this model to serve as a foundation for future computational and mathematical study, and to help inform and refine the biological model.

Acknowledgements

I am heartily grateful to my supervisor Dr. Ian McQuillan. It is an honour for me to work on this research with his great supervision and support. I have learned from him attitudes towards work as a good researcher and how to be a great mentor, which will be a priceless treasure for the rest of my life.

I would like to express my sincere gratitude to my valuable committee members Dr. Anthony J. Kusalik and Dr. Mark Keil.

Thanks to my friends for helping me during the work.

Thanks to my parents and my lovely wife, for everything.

Contents

List of Tables

List of Figures

List of Abbreviations

MDS	Macronuclear Destined DNA Sequence
IES	Internal Eliminated Sequence
mac	Macronucleus Sequence
mic	Micronucleus Sequence
scnRNA	Scan RNA / Small RNA
bp	Base Pair
2JLP	2 Jönsson Lipps Postberg

Chapter 1

Introduction

Ciliates are a group of protozoans characterized by the presence of hair-like organelles called cilia and by the presence of two nuclei in the same cell, called the *micronucleus* and the *macronucleus* [20]. Worldwide, 4,500 different species of ciliates are known and there are more that are unknown [3]. The macronucleus produces all the RNA needed for cell operations, and the micronucleus remains silent. The inert micronucleus is activated during the period of sexual reproduction. By mixing ciliates of two different mating types, sexual reproduction can occur. A schematic presentation of mating for the case of stichotrichs (a group of ciliates) is shown in Figure 1.1. At first, the two cell membranes are fused and a connecting channel is formed in the area of sticking. Then the diploid micronuclei in each cell undergoes meiosis and creates four haploid micronuclei. In the case of stichotrichs, it has two diploid micronuclei and two macronuclei [8], and after meiotic division it has eight haploid micronuclei. The two cells then interchange one haploid micronucleus through the cytoplasmic channel. The interchanged haploid micronucleus then fuses with a resident haploid micronucleus forming a new diploid micronucleus. After that, the two cells detach from each other and heal their cell membranes. The remaining six unused haploid micronuclei and the two macronuclei in each cell begin to degenerate. At the same time the new diploid micronucleus in each cell divides by mitosis without an accompanying division of the cell, creating two daughter micronuclei. One of them remains as the new diploid micronucleus and the other develops into a new polyploid macronucleus during the next three days. During that period the unused haploid micronuclei and the old macronuclei completely disappear. At the completion of the development, the new micronucleus and macronucleus divide (without cell division) and creates two micronuclei and two macronuclei, which is the end mark of the mating.

Here, the structure of the micronucleus gene is introduced and the construction process of macronuclei is discussed. The micronucleus has two classes of specific DNA sequences— the non-coding DNA segments known as *IESs* (internal eliminated sequences) and the gene segments known as *MDSs* (macronuclear destined DNA sequences). A functional macronucleus can be constructed by deleting IESs and merging MDSs from the micronucleus. The process of converting the micronucleus into a macronucleus is known as the gene assembly process in ciliates. Different ciliates perform the gene assembly process in different ways. In the case of two genera of ciliates *Tetrahymena* and *Euplotes*, the MDSs of the micronucleus are interrupted by IESs but the MDSs occur in the same order as in the macronucleus. In these genera, only removing IESs and splicing the rest of the sequences from the micronucleus can generate a functional macronuclear gene. But in

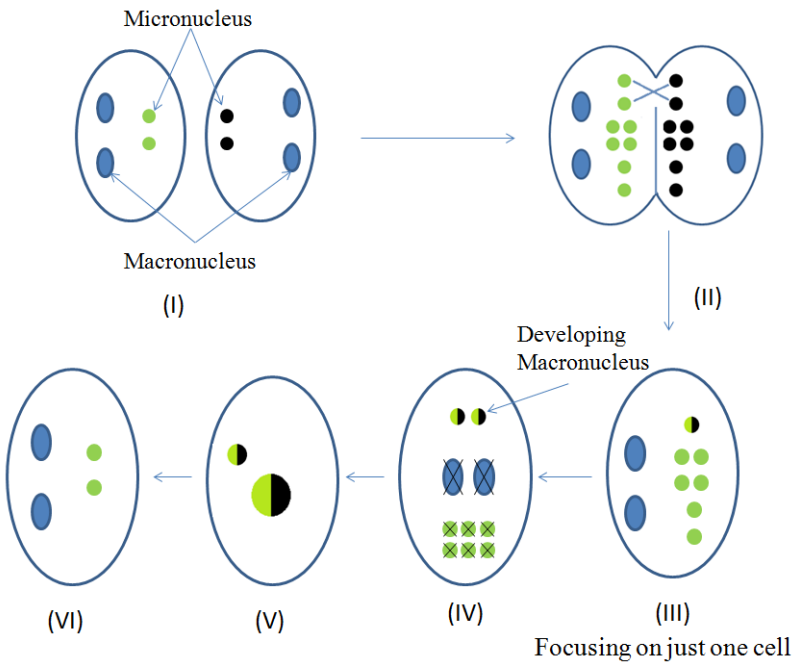


Figure 1.1: Conjugation in stichotrichs. (I) A stichotrich with two macronuclei and two micronuclei. (II) Two stichotrichs attach and form a cytoplasmic channel. The two diploid micronuclei form eight haploid micronuclei. (III) The two cells exchange one haploid micronucleus, and the two stichotrichs separate from each other. The exchanged haploid micronucleus fuses with a host haploid micronucleus forming a new diploid micronucleus (half green and half black). (IV) The new diploid micronucleus divides by mitosis and creates two daughter micronuclei, and the unused six haploid micronuclei and the two macronuclei degenerate. (V) One of the new daughter micronuclei develops into a new macronucleus and another one remains same as the new micronucleus. The unused six haploid micronuclei and the two old macronuclei disappear from the cell. (VI) At the end the conjugation, the new micronucleus and macronucleus divide (without cell division), resulting in two micronuclei and two macronuclei.

the case of stichotrichs, the MDSs are not only interrupted by IESs, but the MDSs also occur in a scrambled order. It is believed that approximately 30% of the micronuclear genes are scrambled. That is why the gene assembly requires a massive quantity of DNA excision and rearrangement from the micronucleus [19, 11, 13]. Figure 1.2 shows a diagram of the gene assembly process.

The DNA of the micronucleus consists of extremely long molecules typical of eukaryotic chromosomes. The number of chromosomes per micronucleus of stichotrichous ciliates has been calculated to be over 100 [1] with estimated chromosome sizes between $2-50$ Mbp [20]. It is already known that the micronuclear DNA contains both IESs and MDSs, and after removing all the IESs it converts into a macronucleus which only contains $2-4\%$ of the micronuclear DNA sequences [1]. Generally the size of each IES is between 5 and 100 bp, and they are AT-rich. In stichotrichous ciliates IESs are flanked by repeat sequences called *pointers* [27]. These pointers are $2-20$ bp in size with one copy of the pointer at the $3'$ -end of one MDS and the other copy at the $5'$ -end of the next MDS (next MDS according to the correct ordering in the macronucleus) [21, 24]. IESs are excised at the boundary between two adjacent MDSs along with one copy of the pointer.

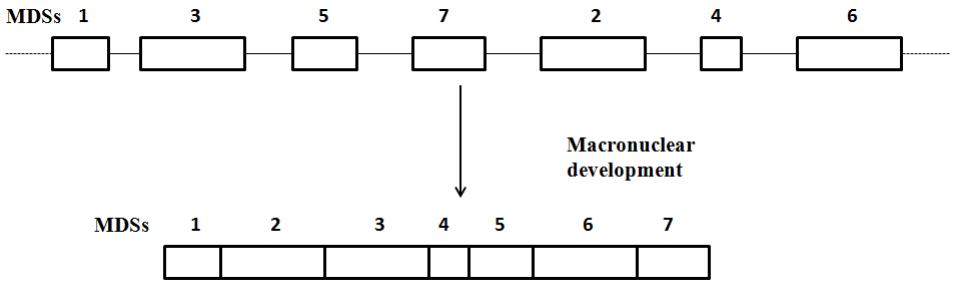


Figure 1.2: A diagram of gene assembly. During macronuclear development, IESs are excised from the micronucleus and the MDSs are joined in the correct order to yield a macronuclear gene.

There are a variety of biological models and hypotheses that have been created to model the gene assembly process in ciliates, such as the intramolecular model [23], the intermolecular model [12], the *scnRNA model* [15], the *template guided model* [24], and the *2JLP model* [10] (many of which are described in Chapter 2). And from a number of those, computational models have been created in an attempt to formally capture the biological models. All the existing models of the gene assembly appear to capture at least part of the gene assembly process, even though these models have some experimentally verified limitations [16] such as (discussed further in Chapter 2):

- the scnRNA model does not address MDS reordering,
- the template-guided model can not explain the imprecise IES excision (too many or too few base pairs excised) in the micronucleus during rearrangement of MDSs which is known to occur,
- both intermolecular and intramolecular models are based on predetermined pointers, but these models do not address the method of pointer identification.

A more recent biological model named the 2JLP model [10] was created as a solution to the gene assembly process by overcoming the above mentioned shortcomings. This model considers a combination of the scnRNA model for excising IESs from the micronucleus, and the template-guided model for removing the remaining IESs, for rearranging MDSs, and for a proofreading process.

Although the 2JLP model is generated by combining many of the previous models together that may have some computational models, there is not any computational model for this new biological model. It is desirable to create a computational model which attempts to do the following:

- tries to accurately capture the biological model,
- determine if it is consistent with real data or not,
- build simulations to test the feasibility of the model,
- use the simulation to test ambiguities or edge cases of biological models through systematic variation of parameters.

This work starts by reviewing relevant literature, previous biological models and computational models in Chapter 2. Then, a formal model is defined based on this 2JLP model called the *Computational 2JLP* model. This model is described in Chapter 3 of this thesis. Then, a simulation is developed with the required algorithms to demonstrate the proposed model. The implemented algorithms are provided and discussed in Chapter 4. In Chapter 5, the outcome of the simulation is discussed based on its use with real micronuclear and macronuclear genes. From the simulation outputs, it is evident that the model is suitable for all cases (regardless of whether they are scrambled or unscrambled DNA sequences). In the simulator, some important parameters are considered such as the minimum value needed for sufficient similarity between scnRNAs (small RNAs) and MDSs, and the minimum value to classify IES specific sequences from the new micronucleus, among others. These parameters are used to deal with ambiguities in the biological model and to determine optimal values according to the simulation. Finally in Chapter 6, conclusions, discussions and some future works are mentioned.

Chapter 2

Background

Researchers have been trying to resolve the mystery of the gene assembly process in stichotrichous ciliates for many years. Both computer scientists and biologists are working on this problem. A variety of biological and computational models have emerged that attempt to explain different parts of the process. In this chapter, some of the more recent models are described from both a computational and biological perspective, and presented them in a chronological fashion.

2.1 Biological models

In 1996, David M. Prescott and Michelle L. DuBois mentioned that recombination of identical pointer sequences seems to be one of the major operations of the gene assembly process [22]. Initially, it was thought that pointers may play a role in identifying and removing IESs from the micronucleus at the time of recombination. But after considering the length of pointers ($2 - 20$bp), it was determined that the possibility of multiple occurrences of the same pointer within an adjacent IES would make pointer alignment on its own insufficient [21]. Indeed, any segment of length 2 can occur at random every $2^2 = 4$ nucleotides. Thus, a repeat sequence does not provide enough information to identify IESs perfectly. Prescott suggested that pointers could be used to ligate the ends of MDSs left by IES removal, rather than used for identification and removal of IESs from the micronucleus. He also mentioned as a hypothesis that the old macronucleus might guide the IES removal procedure.

Since then, various biological models have been proposed to explain the gene assembly process; including the *scnRNA model*, the *template guided model*, and the *2JLP model* [10, 14]. Among these, both the *scnRNA* and the *template guided* models have some experimental validation over different species. However, these two models are individually not enough to predict the entire gene assembly process. In this section all of these three models are described while addressing how they function, their experimental support, and their limitations.

2.1.1 Scan RNA model

A model for the gene assembly process was proposed by Mochizuki et al. in 2002 based on small RNAs and they named it the *scan RNA* (scnRNA) model (Figure 2.1) [15]. They proposed that during the early

conjugation period a RNAi-related pathway starts with a bi-directional transcription of the micronucleus. From that genetic material, it generates small RNAs of size $28 - 29$ bp. They are also known as *scnRNAs* because they are used to identify IESs from the micronuclear genome. These scnRNAs are sent to the parental macronucleus for localization. Then all scnRNAs that are similar to some segment of the parental macronucleus degrades. The rest of the scnRNAs that fail to degrade are largely similar to IES-specific sequences. Then these IES-specific scnRNAs travel to the developing macronucleus where they eliminate subsequences that are similar. Upon completion, this generates the new macronucleus.

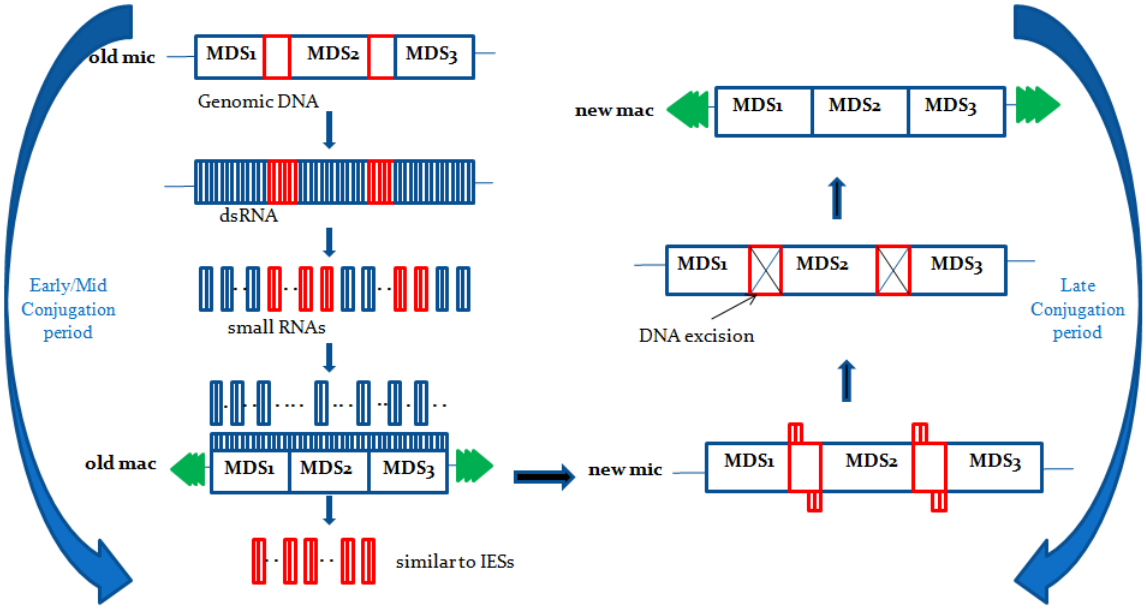


Figure 2.1: The scan (scn) RNA model. During the early conjugation period, the genome of the micronucleus gets transcribed bi-directionally and the resulting transcripts generate double-stranded RNAs (dsRNA) molecules. The dsRNAs are processed into small RNAs (scnRNAs). The scnRNAs travel to the parental macronucleus and any scnRNAs similar to DNA sequences in the parental macronucleus degrade. In late conjugation stages, the rest of the scnRNAs (similar to IESs) transfer to the developing macronucleus, where they target and identify IESs to be eliminated by base pairing.

One of the key points of this model is that it seems to address the case of DNA sequences without scrambling (where the MDSs in the micronucleus are in the correct order), however this model does not address MDS reordering. This is why this model is not sufficient for ciliate species that have scrambled MDSs. Moreover, the model does not easily explain IES removal for cases where IESs are smaller than scnRNAs.

Another limitation of the scnRNA model can be seen by examining the notion of *cryptic pointers*, which are direct repeats of length 1-8 that are in close proximity to real pointers. In fact, despite not being the real pointers, ciliates sometimes use cryptic pointers for splicing. It was observed in an experiment [16] that IESs are deleted randomly and sometimes imprecisely (when IESs are removed based on cryptic pointers) at the middle-late stage of macronuclear development. It was also observed that if multiple repeats (either cryptic or real pointers) are available near MDS-IES junction then the IES deletion may favour longer repeats. This

may cause inaccurate excision of IESs from the new micronucleus. If cryptic pointers are used instead of real pointers, then ultimately they get corrected later in macronuclear development (which is discussed further in Section 2.1.3).

2.1.2 Template guided model

In a key experiment on the ciliate *Paramecium tetraurelia* (that does not have scrambling, but does have IESs), an *IES* was injected into a macronucleus before mating (so that a portion of the macronuclear gene "looked like" the micronuclear version, with two MDSs separated by an IES). Then, the ciliate was allowed to traverse into the sexual cycle [21]. At the end of this experiment it was found that the particular *IES* was present in the structure of the new macronucleus. As a result of this experiment, it was thought that some sequence-specific information must be transferred from the parental macronucleus to the new macronucleus. Hence, a biological model of gene assembly was introduced by Prescott et al. in 2003 and is known as the *template guided model* (Figure 2.2) [24]. In this model a long macronuclear molecule (later determined to be RNA in [18]) which has been generated from the parental macronucleus is used as a template to guide both IES removal and MDS reordering in the developing macronucleus (Figure 2.2a). During early conjugation, these template RNAs are produced by telemore-to-telemore transcription of the short gene-sized macronuclear chromosomes (Figure 2.2a). As further support of this model, Nowacki et al. [18] performed an experiment by injecting mutated artificial sequences into the template RNAs, which eventually changed the order of MDSs in the new macronucleus (Figure 2.2b).

2.1.3 2JLP model

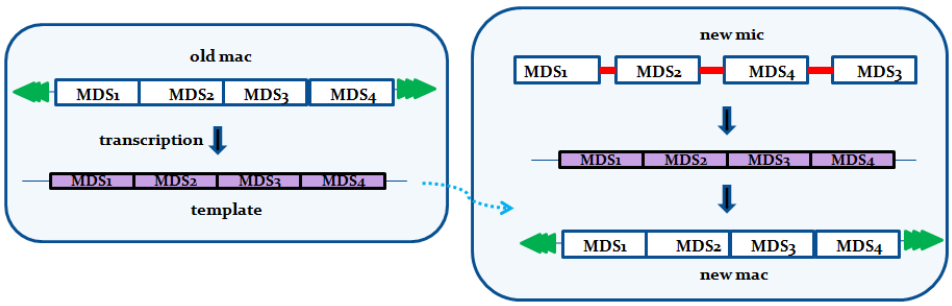
According to the previous two models, there is some evidence that the *scnRNA* model filters out IESs from the new micronucleus. There is also other evidence that some parts of the template model must also be true, with template molecules being present, and influencing the resulting macronucleus. Based on this idea, a more recent biological model of gene assembly was proposed by Jönsson et al. in 2009 [10]. It is known as the *2JLP* model, and it largely combines together portions of all the previous models, which all occur within a temporal procedure (Figure 2.3).

Definition 1 (2JLP Model)**.** *This model can be defined by the following steps:*

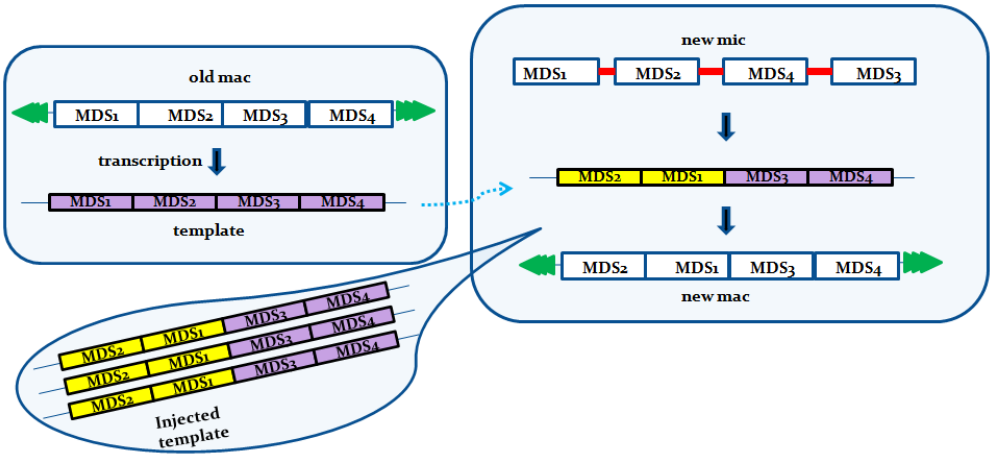
1. *During the early period after conjugation the scnRNA model generates scnRNAs. Here, the genome of the micronucleus is transcribed bi-directionally and the resulting transcripts generate double-stranded RNA molecules which are eventually processed into scnRNAs.*

2. *These scnRNAs travel to the parental macronucleus and any scnRNAs similar to DNA sequences in the parental macronucleus are degraded.*

3. *In the late conjugation stages, the remaining portion of the scnRNAs (that are similar to IESs) are*



(a) The template guided mobel based on the template generated from the old macronucleus



(b) The template guided mobel based on the injected template

Figure 2.2: Examples of template guided model. (a) The template RNAs (shaded rectangles) are produced by telemore-to-telemore transcription of short gene-sized macronuclear chromosomes. During macronuclear development these template RNAs act as scaffolds to guide DNA unscrambling. (b) Microinjection of artificial templates (in this example, RNA having a $MDS2-MDS1-MDS3-MDS4$ sequence) alters the order of the DNA unscrambling pattern in the new macronucleus.

transferred to the developing new macronucleus, where they target and identify IESs to be eliminated by base pairing.

4. *At the same time, the template guided model generates template RNAs from the parental macronucleus to guide the alignment of MDSs and their pointer sequences, and produces the new macronucleus.*

5. *In the case of scrambled genes, the template RNAs perform unscrambling of MDSs according to their order in the macronuclear chromosomes. Homologous recombination between the aligned pointers splice out IESs. For IES excision, if cryptic pointers are used instead of real pointers, a proofreading mechanism guided by the template ensures the missing sequences are filled in and the extra sequences are removed to create full-length chromosomes.*

Both *scnRNA* and *template guided* models seem to tell a part of the story. By combining both of these well known models, it is possible to get a more complete understanding of the gene assembly process. An example of the *2JLP* model based on artificial data is shown in Figure 2.4.

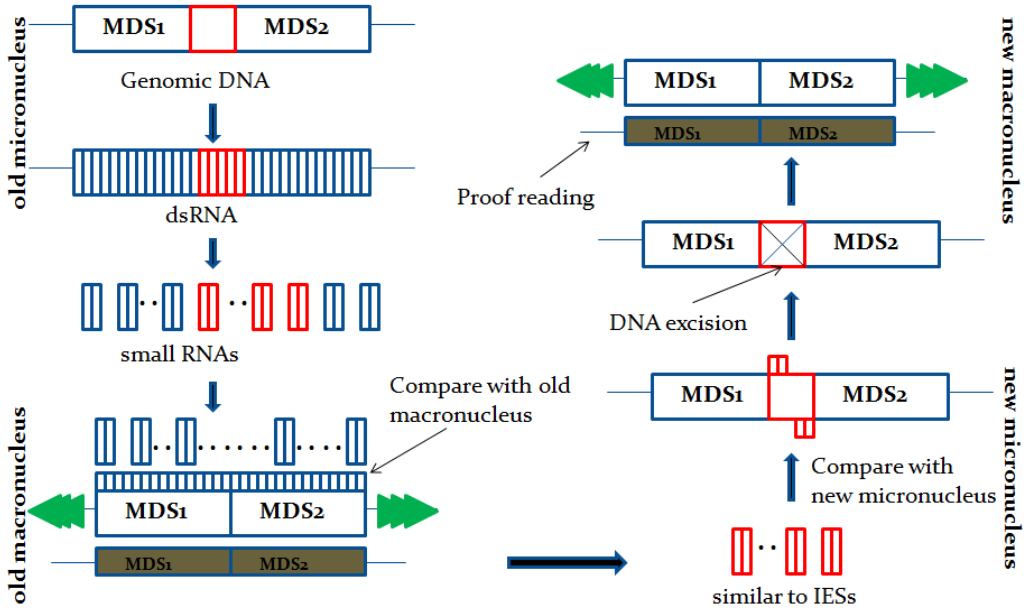


Figure 2.3: The 2JLP model combines aspects from the scanRNA model and the template-guided model to explain gene assembly in ciliates. The whole micronuclear genome is transcribed early in macronuclear development into long double-stranded transcripts, which are processed into small RNAs (scnRNAs). These complexes invade the old macronucleus. There, scnRNAs similar to macronuclear sequences (dark blue) are localized. The rest of the scnRNAs (red) are sent to the new micronucleus for marking and excision of IESs by recruiting chromatin-modifying proteins to the micronuclear-specific sequences. IESs are small in stichotrichous ciliates and cryptic pointers may be used instead of real pointers, which cause imprecise excision. Imprecisely processed sequences will be corrected by a proofreading mechanism that is guided by template RNAs (gray). These template RNAs originate from the old macronucleus. In scrambled genes the template RNAs guide alignment of micronuclear MDSs into the correct order, creating a new macronucleus.

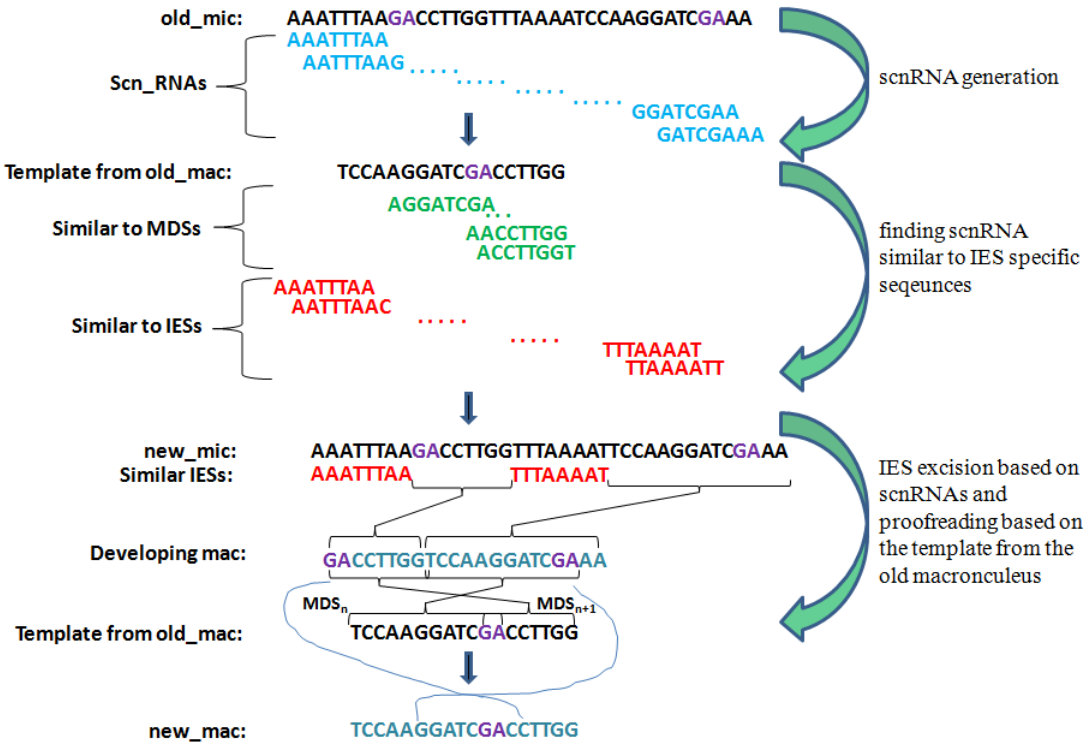


Figure 2.4: An example of the 2JLP model based on artificial data.

2.2 Computational models

There have been various computational models of the gene assembly process based on the above discussed biological models. Among these models the *intermolecular model*, *intramolecular model* and *template guided recombination* [12, 23, 9, 7] are described here. All of these models are designed on string operations.

2.2.1 Intermolecular model

The first primary theoretical model for gene assembly was proposed by Laura F. Landweber and Lila Kari in 1999 [12] and it consists of one unary intramolecular and two binary intermolecular operations of DNA recombination on pointers. The whole model is known as the *intermolecular model* because two of these three operations are intermolecular operations.

For the intramolecular recombination operation (Figure 2.5), the input is a linear molecule of the form $c\rho x\rho d$ where c, ρ, x, d are subsequences of a DNA strand with c, d representing *MDSs* and ρ representing a pointer sequence. In this operation ρ guides the homologous recombination. After ρ finds its second occurrence in the input linear molecule ($c\rho x\rho d$), then it recombines and leads to the formation of two new molecules: a linear DNA molecule $c\rho d$ and a circular molecule $[\rho x]$. Here brackets are used to represent a circular molecule.

The first binary intermolecular operation (Figure 2.6) of the model is the inverse operation of the intramolecular recombination. In this operation, two DNA molecules are needed to take as inputs. One of them is a linear DNA molecule of the form $c\rho d$, where c, ρ, d are subsequences of a DNA strand. Another one is a circular DNA molecule of the form $[\rho x]$, where ρ, x are subsequences of a DNA strand. This operation is accomplished by inserting ρx or $x\rho$ in the linear molecule $c\rho d$. The output of this operation is the linear DNA molecule of the form $c\rho x\rho d$.

The second binary intermolecular operation (Figure 2.7) of the model performs homologous DNA recombination over two linear DNA molecules of the form $c\rho d$ and $u\rho v$ where c, ρ, d, u, v are subsequences of a DNA strand with c, d, u, v representing MDSs and ρ representing a pointer sequence. In this operation the molecules undergo a homologous recombination in ρ and generates two new linear DNA molecules of the form: $c\rho v$ and $u\rho d$.

The basic idea of the above explained operations is to do homologous recombination at ρ and to create the longer, composite MDS by merging two MDSs together. The *intermolecular* model is reversible and that is why the authors of this paper expected that the macronucleus is generated from the micronucleus by using these three operations iteratively. If we closely observe these three operations, we can see that the *intermolecular model* is very much related with the biological concept given by David M. Prescott [21] where he mentioned that the recombination of pointers as one of the major operations of the gene assembly process. Moreover, all of these operations are based on predetermined pointers. There is no information in the model regarding the mechanism of pointer identification from the DNA molecule.

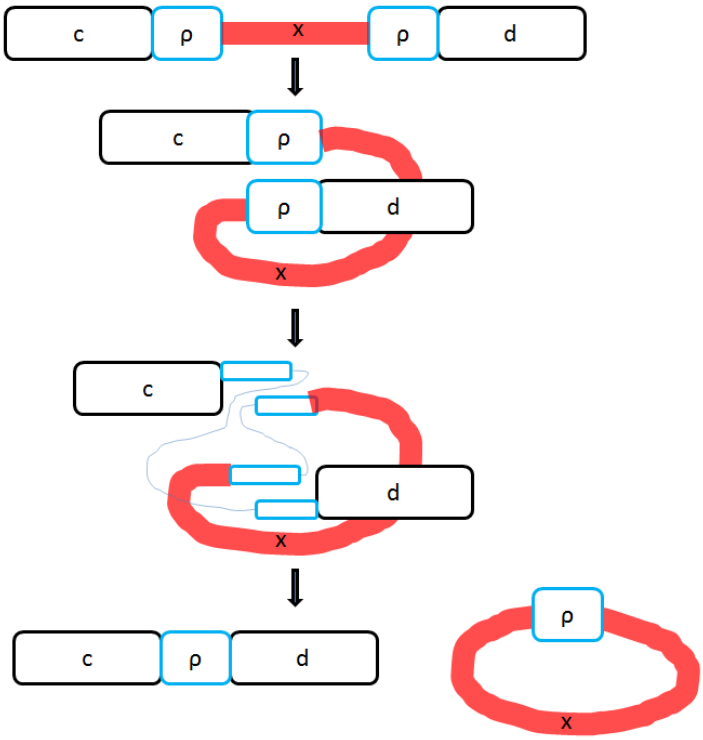


Figure 2.5: Unary intramolecular operation of DNA recombination. At the beginning, it takes a linear molecule of the form $c\rho x\rho d$ as input. Then it aligns two occurrences of ρ in parallel and the molecule forms a loop. It performs DNA recombination operation at repeat ρ. Finally it generates one linear and one circular DNA molecule. In the figure, black rectangles are used to represent MDSs, blue rectangles are used for pointers, red lines are used for IESs and transparent blue lines are used for homologous recombination.

2.2.2 Intramolecular model

Another primary theoretical model for gene assembly was introduced by Prescott et al. [23] and Ehrenfeucht et al. [9] in 2001. It consists of three unary molecular operations based on pointers. All of these operations take one linear DNA molecule as the input and that is why the model is referred to as the *intramolecular model*.

The first operation is called *loop recombination* or in short, *ld*. This operation is exactly the same as the previously discussed intramolecular DNA recombination (Figure 2.5). In this operation, the pair of repeat sequences (*pointers*) of the input align in proximity of each other. Then it performs homologous DNA recombination and generates one linear molecule and one circular molecule.

The second operation is called *hairpin recombination* or in short, *hi* (Figure 2.8). This operation is applicable to a portion of a scrambled DNA molecule where it has a pair of pointers such that one occurrence is an inversion of the other one. To define the operation, let $\Sigma = \{A, C, T, G\}$ be the alphabet of DNA nucleotides, Σ^* the set of words over Σ, and Σ^+ the set of non-empty words over Σ. If $\rho = a_1a_2a_3...a_{n-1}a_n$ is a pointer in Σ^+ where $a_i \in \Sigma^*$, and $1 \leq i \leq n$, then an inverse pointer is $\overline{\rho} = a_na_{n-1}...a_3a_2a_1$ [6]. Assume a

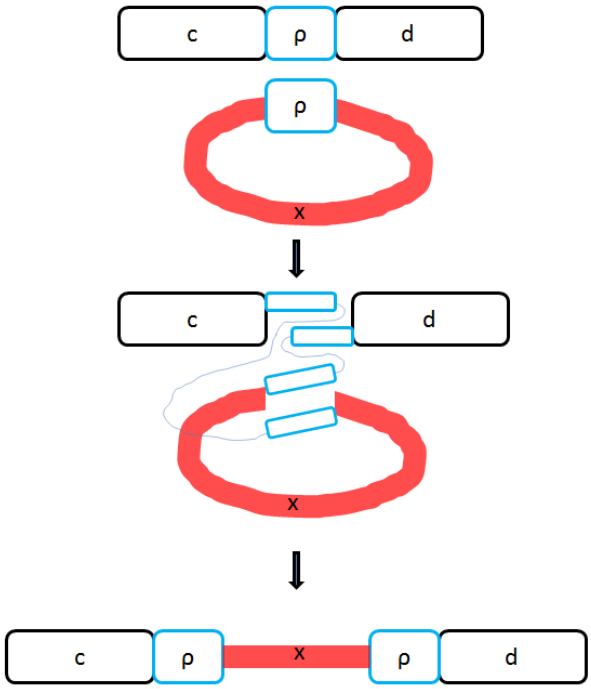


Figure 2.6: First binary intermolecular operation takes two inputs: a linear molecule of the form $c\rho d$ and a circular molecule of the form ρx. Then, it performs DNA recombination operation on ρ of these two input molecules. Finally it generates one linear DNA molecule. In the figure, black rectangles are used to represent MDSs, blue rectangles are used for pointers, red lines are used for IESs and transparent blue lines are used for homologous recombination.

DNA molecule of the form $\overline{d\rho'}x'c\rho'x''$ where ρ' and $\overline{\rho'}$ are a pointer and its inverse respectively, and $\overline{d}, x', c, x''$ are subsequences of a DNA strand. In this operation the molecule folds in such way that it brings two pointers with the same polarity together and form a structure of a hairpin. Then homologous recombination between pointers generates a new linear and unscrambled DNA molecule of the form $\overline{x''\rho'}x'c\rho'd$.

The third operation is called *double-loop recombination* or in short, *dlad* (Figure 2.9). This operation is applicable to a portion of a scrambled DNA molecule having two pairs of pointers in which one pair of pointers overlaps the area covered by the other pair of pointers. Assume that the molecule has the form $c\rho x_1\rho' e\rho x_2\rho' d$, where ρ, ρ' are pointers and c, d, e, x_1, x_2 are subsequences of a DNA strand. In this operation the molecule aligns similar pointers where one copy of similar pointer creates the first loop and another copy of similar pointer creates the second loop. Then, it performs double homologous recombination between pointers and produces a new linear, unscrambled DNA molecule of the form $c\rho x_2\rho' e\rho x_1\rho' d$.

It is important to note that the loop recombination process generates a circular molecule and if this circular molecule contains an MDS then it cannot be spliced again by performing any one of these three operations. Moreover, all three operations are based on the recombination of pointers described by Prescott [21]. However, one limitation of these models is that they do not discuss the process of pointer identification in their model. All of these three operations then represent only a step of the gene assembly rather than the whole.

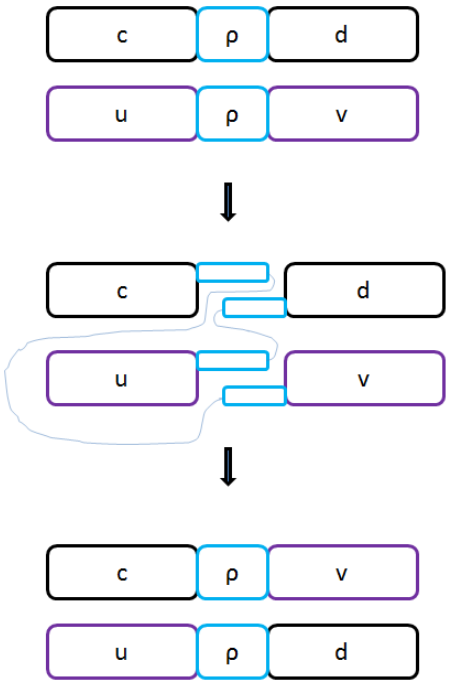


Figure 2.7: Second binary intermolecular operation takes two linear DNA molecules as input of the form $c\rho d$ and $u\rho v$. Then, it performs a homologous recombination in ρ of these two input molecules. Finally it generates two linear DNA molecules. In the figure, black and purple rectangles are used to represent MDSs, blue rectangles are used for pointers, and transparent blue lines are used for homologous recombination.

2.2.3 Template guided recombination

A generalized version of the *template guided model* [24] was proposed by Daley and McQuillan in 2004 [7] and is known as the *template guided recombination* operation. Previously, it has been discussed that both intermolecular and intramolecular gene assembly models are based on a process whereby pointers are already known. Both the inter- and intramolecular models are unable to deal with the identification mechanism of pointers which is necessary for arranging all MDSs in the correct order, and even for removing IESs from unscrambled genes. The *template guided recombination* operation resolves this issue of finding the pointer positions in the micronucleus. In this model, the assembled gene from the old pre-conjugation macronucleus is used as the template for finding the correct order of MDSs as well as corresponding pointers.

The model (Figure 2.10) considers an iterated recombination based on the template. The model takes two DNA segments and merges them together, if it matches with a portion of a template. Assume $u\alpha\rho d$ and $e\rho\gamma v$ are two DNA segments where $u, v, \alpha, \rho, d, e, \gamma$ are subsequences of a DNA segment in which α, γ represent MDSs and ρ represents a pointer sequence. Now we need a portion of a template with the form $\alpha\rho\gamma$ for splicing these two DNA segments. If we compare two DNA segments with the template then $\alpha\rho$ and $\rho\gamma$ are common portions from the first and second DNA segments consecutively. For splicing these two DNA segments based on the template, at first we need to cleave and remove d and $e\rho$ or ρd and e from the first

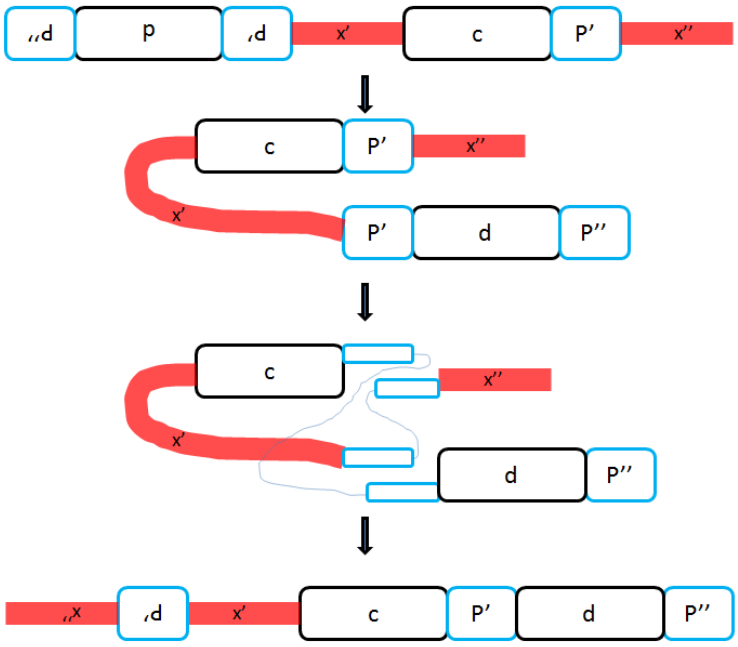


Figure 2.8: Hairpin recombination. This operation takes one linear, scrambled DNA molecule having reverse pointers as input, of the form $\overline{d\rho'}x'c\rho'x''$. Then, it folds in such a way that the two pointers come into the same direction. After that, it performs a homologous recombination between pointers. Finally it generates a linear and unscrambled DNA molecule. In the figure, black rectangles are used to represent MDSs, blue rectangles are used for pointers, red lines are used for IESs and transparent blue lines are used for homologous recombination.

and second segments consecutively. Then splice the rest of the portions ($u\alpha\rho$ and γv or $u\alpha$ and $\rho\gamma v$) of these two DNA segments and generate $u\alpha\rho\gamma v$ as a product of a single iteration of template guided recombination. After running this operation arbitrarily many times it can give a complete macronucleus.

In 2007, Angeleska et al. proposed a modification on this model by considering the template as a double-stranded RNA molecule instead of a double-stranded DNA molecule [2]. In an experiment Möllenbeck et al. [16] found that before MDS rearrangement a large number of IESs are excised from the micronucleus, and often imprecisely excised with cryptic pointers. That means rearranging only based on the template RNA is not the complete model of the gene assembly process. If all three biological models are combined then the *2JLP* model (Figure 2.3) best describes the gene assembly process. There has not been any computational model based on the *2JLP* model where IESs are excised first based on scnRNAs and then MDSs are rearranged based on templates generated from the old macronucleus.

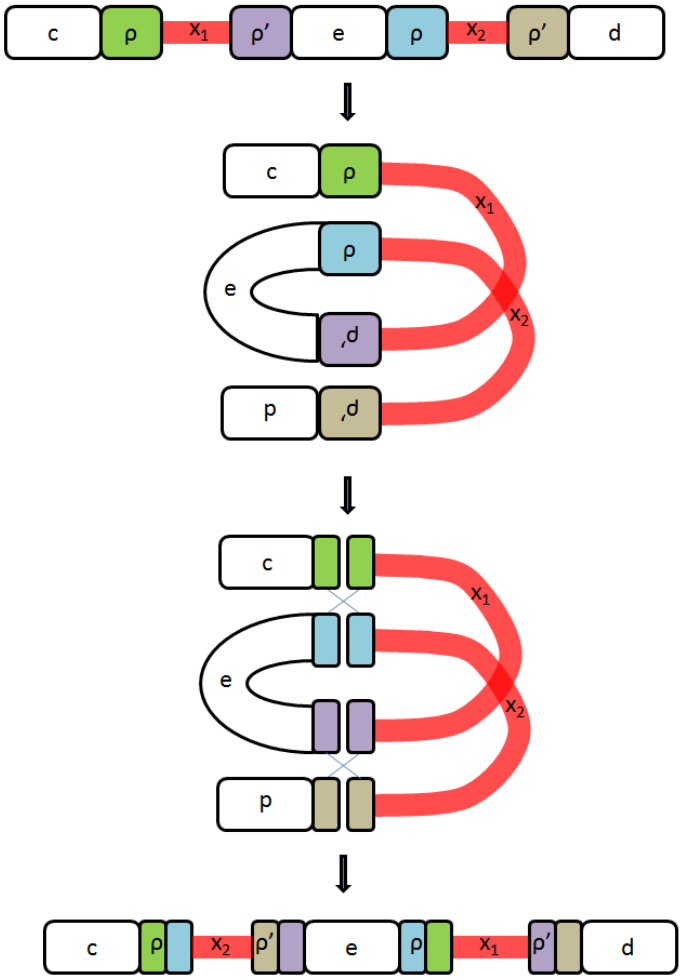


Figure 2.9: Double-loop recombination. This operation takes as input one linear, scrambled DNA molecule having two pairs of pointers in which one pair of pointers overlaps the area covered by the other pair of pointers. This input is of the form $c\rho x_1\rho' e\rho x_2\rho' d$. Then, it folds the molecule in such a way that the two pairs of pointers are aligned, creating a double loop. Then, it performs double homologous recombination between pointers. Finally it generates a linear and unscrambled DNA molecule. In the figure, unshaded rectangles are used to represent MDSs, shaded rectangles are used for pointers, red lines are used for IESs and transparent blue lines are used for homologous recombination.

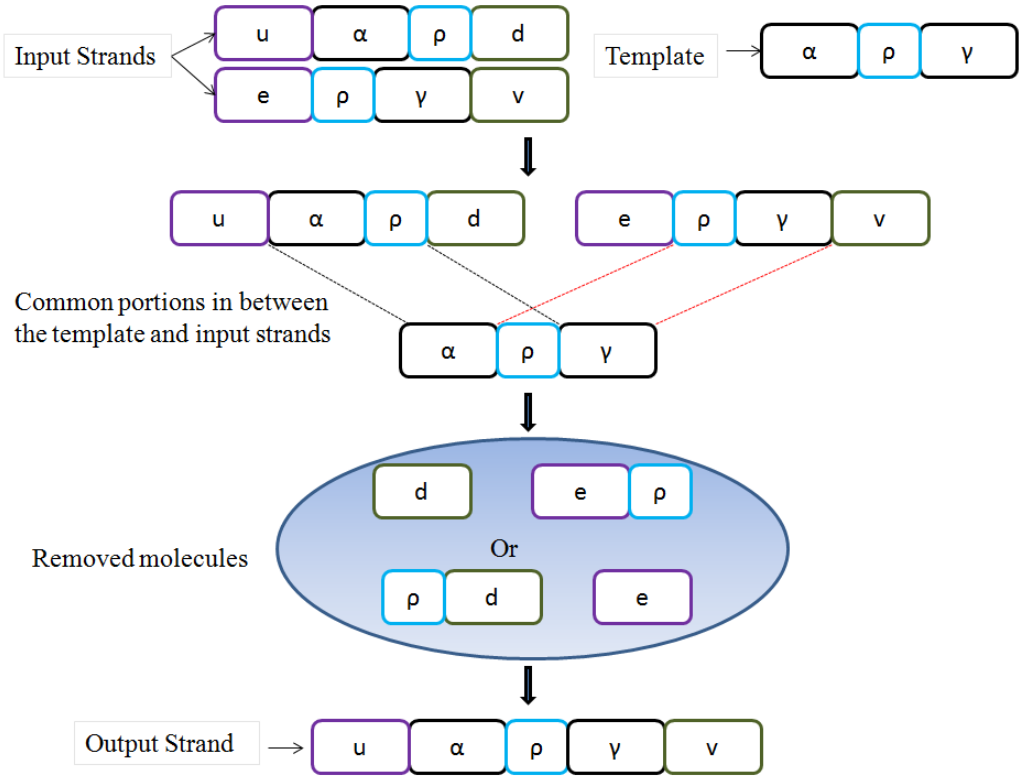


Figure 2.10: Template guided recombination. This operation takes as input two linear DNA segments ($u\alpha\rho d$ and $e\rho\gamma v$) having common pointers (ρ) and a segment of template ($\alpha\rho\gamma$) from the old macronucleus. Then, it finds the common portion between the template and input DNA strands. After that, it cleaves and removes extra molecules from both DNA strands based on the template. Finally, it splices the rest of these molecules and generates the output DNA strand.

Chapter 3

Computational 2JLP model

This chapter formally models the 2JLP biological model. It is known as the *Computational 2JLP* model. In Chapter 4, a simulation based on the computational 2JLP model is built.

In the previous chapter, Definition 1 (the biological 2JLP model) has 5 steps. The first step constructs scnRNAs. The second step finds scnRNAs similar to IES specific sequences. The rest of the steps define the construction process of the new macronucleus by deleting IESs from the new micronucleus, rearranging MDSs and correcting the developing macronucleus based on the old macronucleus. This chapter formally defines these individual steps. Connecting all these individual definitions yields the computational 2JLP model. As the model is defined at the level of sets and strings, and the 2JLP is not described at that level, some simplifying assumptions are necessary. They are stated prior to their use.

Throughout this entire chapter, Σ denotes the fixed alphabet of DNA nucleotides; that is, $\Sigma = \{A, C, T, G\}$. Let Σ^* be the set of all strings over the alphabet Σ, and let Σ^+ be the set of non-empty strings over Σ. Let $\varphi \in \Sigma^*$. Then the *length* of φ is written as $|\varphi|$. If n is a natural number, then Σ^n is the set of all strings over Σ of length n, and $\Sigma^{\leq n}$ is the set of all strings over Σ of length at most n.

3.1 ScnRNAs

In this section, the first step of the 2JLP model (Section 2.1.3) is defined. This step is the construction mechanism of scnRNAs from the old micronucleus. In the model, the size of scnRNAs is assumed to be 28 bp (which is consistent with biological evidence as discussed in Section 2.1.1) and the set of scnRNAs is generated by taking each consecutive 28 nucleotides from the old micronucleus. Suppose the old micronucleus is a string of the form $a_1a_2a_3\cdots a_n$, then the first scnRNA is $a_1\cdots a_{28}$, the second scnRNA is $a_2\cdots a_{29}$ and in the same way, the last scnRNA is $a_{n-28+1}\cdots a_n$. Although there is more than one micronuclear chromosome, the micronucleus can be abstractly represented as a single string by concatenating distinct micronuclei together.

The formalism to turn a RNA into a DNA is briefly defined. Let $\Sigma_R = \{A, C, U, G\}$. Let h be a homomorphism from Σ^* to ${\Sigma_R}^*$ that maps each character of Σ^* to its correspondent RNA character (so A maps to A, C to C, T to U and G to G). The formal definition of the set of scnRNAs is given below:

Definition 2 (ScnRNAs)**.** *Let $\varphi \in \Sigma^*$ be a string representing the old micronucleus, where $\varphi = a_1a_2\cdots a_n, a_i \in \Sigma$, $1 \leq i \leq n$. Then, $\gamma_{28}(\varphi) = \{x \mid |x| = 28, \varphi = yxz, y, z \in \Sigma^*\}$, and $h(\gamma_{28})$ is called the* set of scnRNAs*.*

In the above expression, each string of $\gamma_{28}(\varphi)$ does not represent RNA; rather, it is a piece of a DNA sequence. Then the set $h(\gamma_{28}(\varphi))$ is the set of scnRNAs. Figure 3.1 illustrates the scnRNA definition.

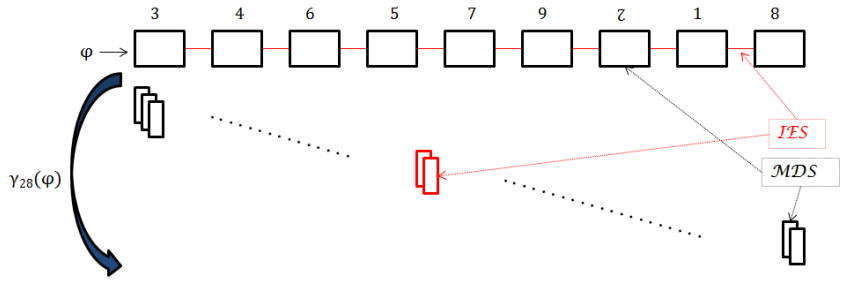


Figure 3.1: Definition of ScnRNAs. Here, φ represents the old micronucleus, $h(\gamma_{28}(\varphi))$ represents the set of scnRNAs, and $\gamma_{28}(\varphi)$ is the set of scnRNAs changed to DNA. Red and black rectangles consecutively represent IESs and MDSs from the old micronucleus, although at this stage, the model does not discriminate between the two types.

3.2 Finding putative IESs

From the scnRNA model (Section 2.1.1), it is evident that after construction, the scnRNAs travel to the parental macronucleus. The model dictates that if a scnRNA is similar to a portion of a chromosome in the parental macronucleus, it gets filtered out while the remaining scnRNAs continue to the next stage of the model (Figure 3.2). The notion of similarity for this filtering is not exactly known. For example, it is unknown to what degree the strings need to be similar, both throughout the sequences and at the ends of the scnRNAs (for example, if 25 of the first 28 nucleotides match, a scnRNA could still become filtered out). For the computational model, a binary relation is used to discuss similarity. This will allow trial of different binary relations in Chapter 4 as part of a simulation reflecting different notions of similarity.

The scnRNAs which are filtered out are largely similar to IES specific sequences. The formal definition for finding the putative IES specific sequences is:

Definition 3 (Similarity, and determination of putative IESs)**.** *Let $\Xi \subseteq \Sigma^{28} \times \Sigma^*$ be a binary relation. This will reflect the notion of similarity, whereby $(w, y) \in \Xi$ if w is similar to y. Since we only consider scnRNAs for the first component, it suffices that the first component be of length* 28*.*

Let $\varphi \in \Sigma^*$. Also, let $Q \subseteq \Sigma^*$ be a finite set of strings representing the old macronucleus. Then,

$\beta_{28}(Q) = \{w \mid w \in \gamma_{28}(\varphi)$*, for each* $\pi \in Q$*, and each* $x, y, z \in \Sigma^*$ *with* $\pi = xyz$*, then* $(w, y) \notin \Xi\}$*.*

That is, if an element of $\gamma_{28}(\varphi)$ is similar (using the relation Ξ) to a subword of some $\pi \in Q$, then that element is not in the set $\beta_{28}(Q)$. However, if some element w of $\gamma_{28}(\varphi)$ is not similar to any subword of any $\pi \in Q$, then $w \in \beta_{28}(Q)$.

It can be seen from the definition that all scnRNAs ($\gamma_{28}(\varphi)$) are checked against the old macronucleus (Q) and all scnRNAs which are not matched according to the binary relation to any substrings of strings in

the old macronucleus are filtered out. For example, if $\Xi = \{(v, v) \mid v \in \Sigma^{28}\}$, then the relation matches all scnRNAs where there is an exact match with some subword of a macronuclear chromosome. In this case, $\beta_{28}(Q)$ would match scnRNAs that are not subwords of any macronuclear chromosomes. Less strict relations are considered for Ξ that allow for similarity as described above. In this case, the new set of scnRNAs consists of sequences that are largely similar to IES specific sequences, and this set is called the *putative IESs* ($\beta_{28}(Q)$). This definition is able to successfully define the second step of the 2JLP model (Section 2.1.3).

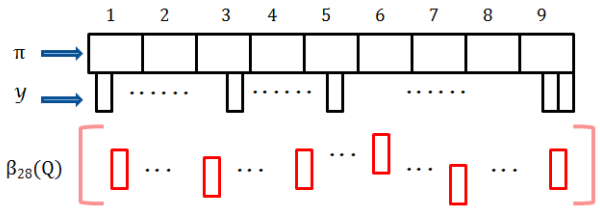


Figure 3.2: This process finds putative IESs by comparing each string π from the old macronucleus (Q) and the set of scnRNAs ($\gamma_{28}(\varphi)$). Here, y is an element from the set of scnRNAs that is similar to a subword of some string in the old macronucleus. Then, $\beta_{28}(Q)$ indicates the set of scnRNAs that are not similar to the old macronucleus and are therefore putative IESs.

3.3 Construction process of the new macronucleus

In this section, the construction process of the new macronucleus from the new micronucleus is modeled by using the putative IES specific sequences and the template (generated from the old macronucleus). This section has two parts. The first part models the IES elimination by the putative IES specific sequences (Figure 3.3) and the second part involves the modelling of the rearrangement and the correction (the IES removal or the MDS insertion) of the developing macronucleus based on the template (Figure 3.4).

3.3.1 Iterated deletion of IESs using scnRNAs

From the third step of the 2JLP model (Section 2.1.3), it can be seen that the filtered scnRNAs (thought to be similar to IES specific sequences) are transferred to the developing macronucleus. These scnRNAs target and remove similar IESs from the new micronucleus which then generates the developing macronucleus.

To define this step, intuitively, each putative IES specific sequence (that is taken from the set $\beta_{28}(Q)$) is checked with the new micronucleus (φ_0), and those sections of the strings are marked by using new barred letters. Then, repeats are identified (which could be either pointers or cryptic pointers) within a certain range (provided by a parameter called *neighbourhood*) at the beginning of a barred section (the section that is similar to the putative IES sequences), and at the end of the barred section. If such repeats exist, then one of the copies of the repeat along with the intermediate string between those repeats are removed. This excision operation is iterated throughout the marked new micronuclues ($\overline{\varphi_0}$) and the developing macronucleus (φ_1) is generated.

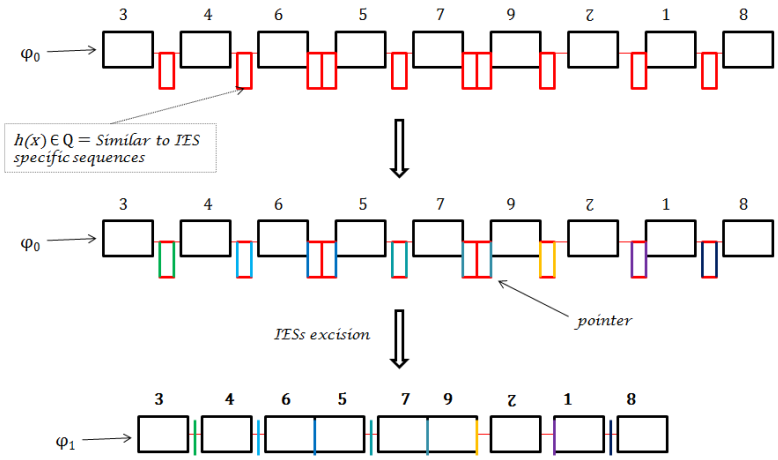


Figure 3.3: Definition of IES Deletion from the new micronucleus (φ_0). At first, all elements of the set $\beta_{28}(Q)$ are compared with φ_0 and these mark the IES specific sequences. Then, IES specific sequences are removed based on pointers (represented with bars of the same colour) at the beginning and the end of all individual marked sequences throughout φ_0. Finally, the developing marconucleus (φ_1) is generated after removing all marked sequences between repeats (either real or cryptic pointers) and one copy of each repeat.

Indeed, Möllenbeck et al. [16] observed deleted segments between repeats in close proximity to the real MDS-IES junction. That is, sometimes cryptic pointers are used instead of real pointers if they are nearby. But they did not mention any approximate range for defining this close proximity. In this model, this proximity is called the *neighbourhood* and considered a parameter.

First the formalism to mark a character is briefly defined, which can be used to represent matching between scnRNAs from $\beta_{28}(Q)$ before the deletion occurs.

Let $\overline{\Sigma} = \{A, C, T, G, \overline{A}, \overline{C}, \overline{T}, \overline{G}\}$. Let $\overline{h}$ be a homomorphism from $\overline{\Sigma}^*$ to $\overline{\Sigma}^*$ that maps each character of $\overline{\Sigma}$ to its barred version (so for example, $\overline{h}(A) = \overline{h}(\overline{A}) = \overline{A}$). Also, let h be a homomorphism from $\overline{\Sigma}^*$ to Σ^* that maps each letter to its unbarred version. And, let ν be a positive integer variable that represents the size of the neighbourhood.

Definition 4 (IES marking)**.** *The first barring of characters is defined as follows:*

- *One iteration: Let $u, v \in \overline{\Sigma}^*$. Then, $u \Rightarrow_{IES} v$, if $u = zxy, v = z\overline{h}(x)y$, there exists $w \in \beta_{28}(Q)$, such that $(w, h(x)) \in \Xi$.*
- *Multiple iterations: $u \Rightarrow^*_{IES} v$, if either $u = v$, or if $u \Rightarrow_{IES} u_1 \Rightarrow_{IES} u_2 \Rightarrow_{IES} \ldots \Rightarrow_{IES} u_m = v$ where $m \geq 1$, and $u_1, \ldots, u_m \in \overline{\Sigma}^*$.*
- *Termination: $u \Rightarrow^{\#}_{IES} v$, if $u \Rightarrow^*_{IES} v$, and $v \Rightarrow_{IES} \overline{v}$ implies $v = \overline{v}$ (in other words, there does not exist any $x \in \beta_{28}(Q)$ that is not barred in v).*

Let $\varphi_0 \in \Sigma^*$ represent the new micronucleus. Then let $\overline{\varphi_0}$ be such that $\varphi_0 \Rightarrow^{\#}_{IES} \overline{\varphi_0}$. Then, $\overline{\varphi_0} \in \overline{\Sigma}^*$ represents the micronucleus with marked IES specific sequences.

Definition 5 (IES deletion)**.** *IES deletion is defined as follows:*

- *One deletion: Let* $u \in \overline{\Sigma}^*$*. Then,* $u \to_{IES} v$*, if* $u = z\rho x\rho y, v = z\rho y$*,* $2 \leq |\rho| \leq 20$*,* $x \in \Sigma^{\leq\nu}\{\overline{A}, \overline{C}, \overline{T}, \overline{G}\}^*\Sigma^{\leq\nu}$*, and if* x *starts with a letter from* $\{\overline{A}, \overline{C}, \overline{T}, \overline{G}\}$*, then* $z\rho$ *ends with at most* ν *barred letters, and if* x *ends with a letter from* $\{\overline{A}, \overline{C}, \overline{T}, \overline{G}\}$*, then* ρy *starts with at most* ν *barred letters.*

- *Multiple deletions:* $u \to^*_{IES} v$*, if either* $u = v$*, or if* $u \to_{IES} u_1 \to_{IES} u_2 \to_{IES} \ldots \to_{IES} u_m = v$ *where* $m \geq 1$*, and* $u_1, \ldots, u_m \in \overline{\Sigma}^*$*.*

- *Termination:* $u \to^{\#}_{IES} v$*, if* $u \to^*_{IES} v$*, and there does not exist any* y *such that* $v \to_{IES} y$ *(in other words, it removes all barred characters possible between a beginning and an end repeat along with one copy of the repeat).*

Let $\overline{\varphi_0}$ *be such that* $\varphi_0 \Rightarrow^{\#}_{IES} \overline{\varphi_0}$*, and so* $\overline{\varphi_0}$ *is the output of the termination phase of the previous operation. Let* $\overline{\varphi_1}$ *be such that* $\overline{\varphi_0} \to^{\#}_{IES} \overline{\varphi_1}$*. Then, any barred letters that were not removed are no longer relevant. Therefore, let* $\varphi_1 = h(\overline{\varphi_1})$*, which represents the next stage of the developing macronucleus.*

3.3.2 Additional IES excision and MDS rearrangment by using templates

Next, the remaining two steps of the 2JLP model (Section 2.1.3) are defined. The fourth step of the 2JLP model is based on the template guided model (Section 2.1.2). In this step, MDSs of a developing macronucleus (after deleting IESs based on the scnRNA model) are rearranged based on the template RNAs generated from the old macronucleus and produces the next iteration of the developing macronucleus. The final step of the 2JLP model is the proofreading which also involves the templates. In this step, a developing macronucleus is compared with the template RNA, filling in missing nucleotides and removing extra nucleotides. This is because IES removal precision is imprecise and there is a hypothesized proof-checking process needed before generating the final macronucleus.

To define these two steps, it is necessary to define both intramolecular (occurring within a sting) and intermolecular (occurring between strings) operations. They are performed together with a template, which is a string from the old macronucleus Q. (Although in reality, the template is RNA derived from the old macronucleus, from the perspective of the formal model, this will not make any difference.)

Here, the '$\approx$' symbol represents similarity.

Definition 6 (Intramolecular operation)**.** *Let* $y \in \Sigma^*$ *and* $Q \subseteq \Sigma^*$ *(the old macronucleus). Then* $y \rightsquigarrow_{intra} \{z, x\rho\}$*, where* $x, z \in \Sigma^*$*,* $\pi = ru'\rho v's \in Q$*,* $y = tu\rho x\rho vm$*,* $z = tu\rho vm$*, and* $m, r, s, t, u, u', v, v'\rho \in \Sigma^+$*, and* $2 \leq |\rho| \leq 20$*,* $u \approx u'$*,* $v \approx v'$*.*

This operation is used to rearrange two MDSs (likely u and v) via pointer ρ. These segments, $u\rho v$ must match (or be similar to) the template. The segment x in the middle (which could contain other MDSs if u and v were scrambled) along with one copy of the pointer, is kept as a separate molecule (string).

Definition 7 (Intermolecular operation)**.** *Let* $y, x \in \Sigma^*$ *and* $Q \subseteq \Sigma^*$ *(the old macronucleus). Then* $\{y, x\} \rightsquigarrow_{inter} \{z, j, d\rho\}$ *and* $\{y, x\} \rightsquigarrow_{inter} \{z, \rho j, d\}$ *where* $y = tu\rho j$*,* $x = d\rho vs$*,* $\pi = ru'\rho v'j \in Q$ *and* $z = tu\rho vs$*, and* $j, r, s, t, u, u', v, v'\rho \in \Sigma^+$*, and* $2 \leq |\rho| \leq 20$*,* $u \approx u'$*,* $v \approx v'$*.*

This is essentially the same as Definition 6 except it will work if the two MDSs to be joined together next are on separate strings. So, for example, after an application of Definition 6 bringing together two MDSs with other MDSs in between, those middle MDSs get excised on a separate string. Those can then rejoin the "main string" with an application of Definition 7.

Now, both the intramolecular and the intermolecular operations are applied arbitrarily over a developing macronucleus (φ_1) in the definition of the second last step of the 2JLP model which is as follows:

Definition 8 (Rearrangement)**.** *The rearrangement is defined as follows:*

- *Rearrangement of two consecutive MDSs with an IES excision or an MDS insertion: Let* $X_1 = \{\varphi_1\}$*. This is the starting point. We iteratively construct a set* X_{i+1} *from* X_i *until some stopping point. Then,* $X_i \to X_{i+1}$ *if either*
  - $u \in X_i$*,* $u \rightsquigarrow_{intra} \{v, x\}$*,* $X_{i+1} = X_i - \{u\} \cup \{v, x\}$*.*
  - $u, x \in X_i$*,* $\{u, x\} \rightsquigarrow_{inter} Z$*,* $X_{i+1} = X_i - \{u, x\} \cup Z$*.*
- *Multiple excision or insertion with rearrangement:* $X_1 \to^* X'$*, if either* $X_1 = X'$*, or if* $X_1 \to X_2 \to \ldots \to X_n = X'$ *where* $n \geq 1$*, and* $X_1, \ldots, X_n \subseteq \Sigma^*$
- *Termination: Let* Q *be the old macronucleus. Then,* $X_1 \to^\# X'$ *if* $X_1 \to^* X'$*, and every string* $w \in Q$ *is similar to some string in* X'*.*

Let $\varphi_1 \in \Sigma^*$ *represents the developing macronucleus. Then, let* φ_2 *be the longest string in* X' *such that* $\varphi_1 \to^\# X'$*.*

The longest string is of primary interest as iterations of the rearrangement create (perhaps many) strings containing primarily IESs that do not rejoin the "main string" as they do not match the template.

Then, to do the final proof correction stage, it is not necessary to formally model this step, but instead it is algorithmically described. A pairwise alignment algorithm is used to align φ_2 with each string of the old macronucleus. For every gap along φ_2, the sequence information from the string of Q is used instead. And for every gap along the string of Q, the corresponding segment of φ_2 is deleted. Thus, this simulates the final removal and insertion of small segments of the developping macronucleus according to the template.

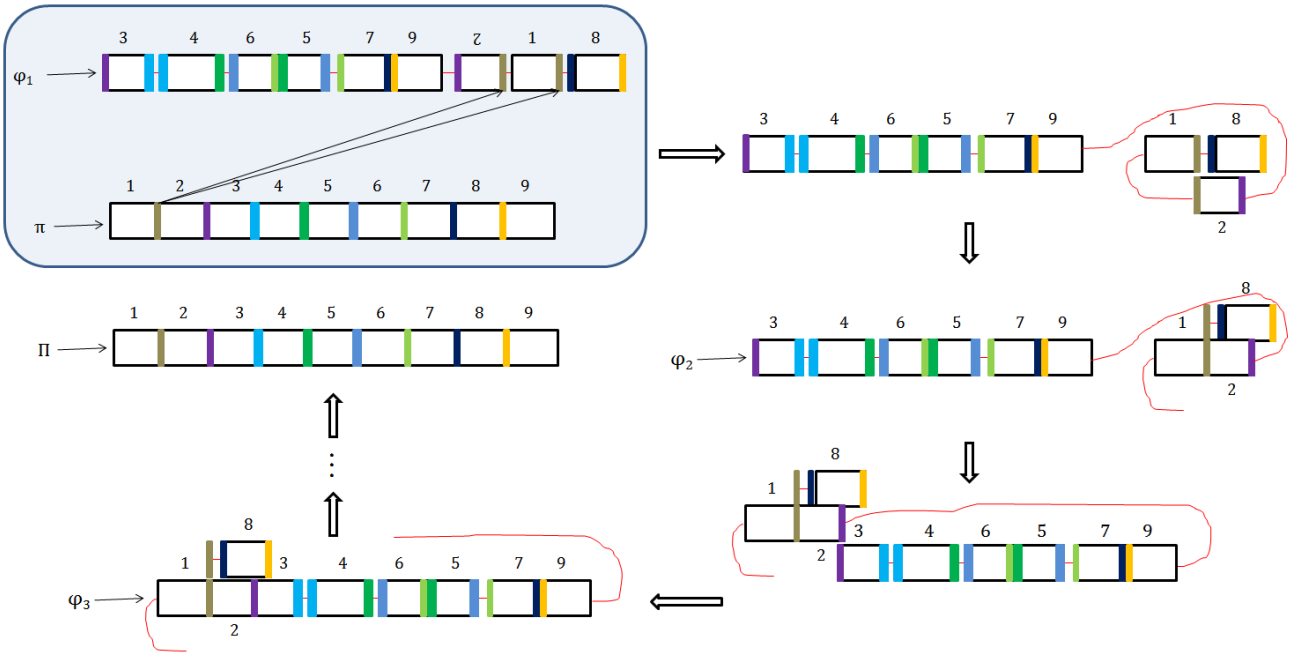


Figure 3.4: Definition of Rearrangement of the developing macronucleus (φ_1) by using the old macronucleus (π).

Chapter 4

Algorithm

This section describes implemented algorithms based on the Computational 2JLP model (Section 3.1), a formal model of the biological 2JLP model (Section 2.1.3). These algorithms are built to demonstrate each definition and to check the feasibility of the computational model. Another purpose of developing the algorithms is to find out some important aspects about the gene assembly process by analysing the results of the simulation with real data. These findings can be helpful for refining the 2JLP model. For example, in the algorithm, certain values are parameterized that were left ambiguous or not described in the biological model. Then, it can be tested which values for the parameters give better results.

This chapter is divided into two sections: Section 4.1 discusses important parameters, and Section 4.2 gives a detailed description of the implemented algorithms. Figure 4.1 shows the flow diagram of the pipeline used to simulate the Computational 2JLP model.

In the algorithms for doing string comparison, global [17] , semi-global [4], and local [25] alignment techniques are used. Knowledge of these algorithmic techniques is assumed in this thesis. For scoring alignments a match scores 1, mismatch scores -1, and gap scores -2. These values are chosen for simplicity since there are already many parameters being systematically varied for the simulation.

From Figure 4.1, it is evident that the simulator has five major functions (green shaded rectangles). These are *scnRNA*, *IES*, *IES_deletion*, *rearrangement*, and *correction*. Among these, the scnRNA function closely demonstrates the mechanism of Definition 2. The IES function demonstrates the mechanism of Definition 3. The *IES_deletion* function demonstrates the mechanism of Definition 4 and 5. For demonstrating the mechanism of Definition 8, both the *rearrangement* and the *correction* functions are used.

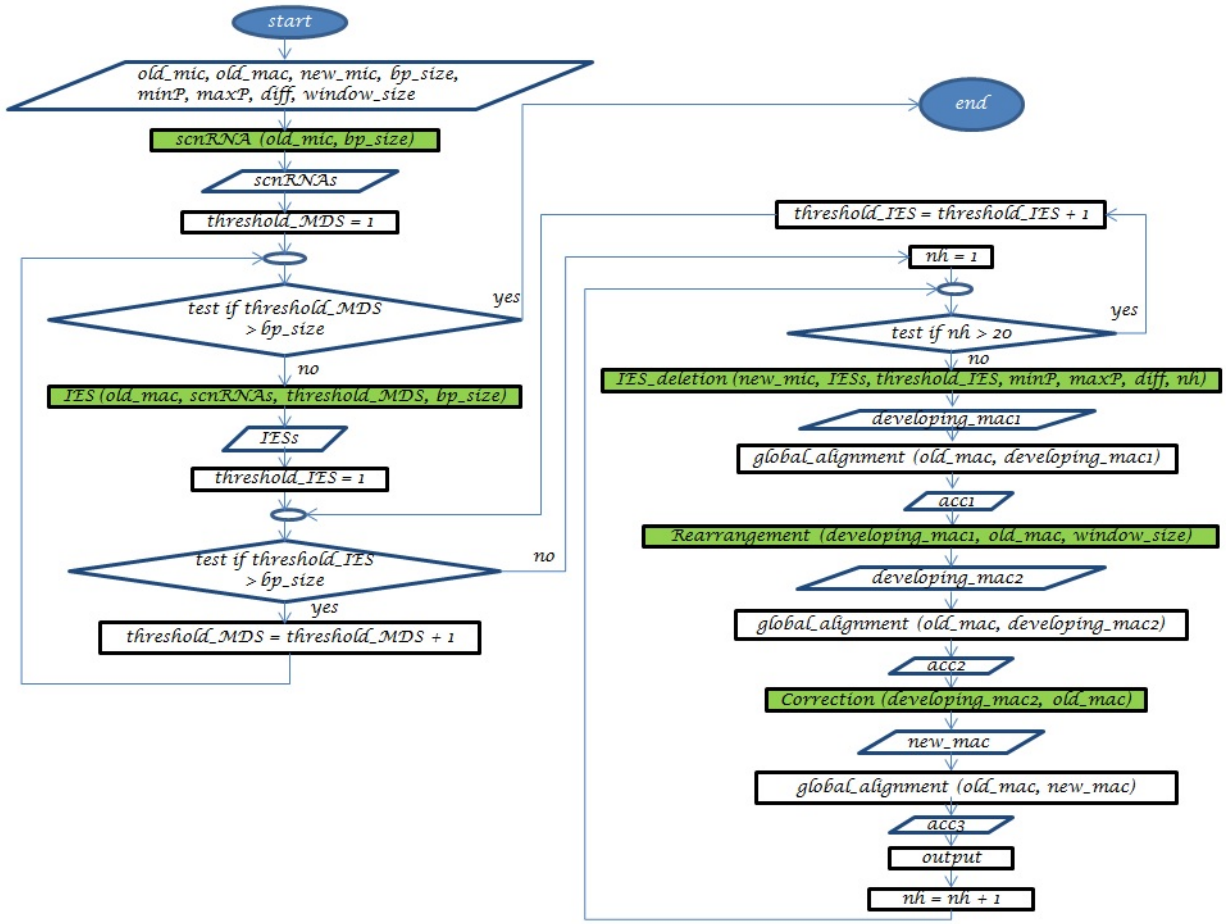


Figure 4.1: Flow diagram of the simulator. Major functions of the simulator are represented by green shaded rectangles. The parameters are explained in Table 4.1. Each part of the pipeline will be explained in Section 4.2.

4.1 Important parameters

In the algorithm, some important parameters are considered. Figure 4.1 shows the use of these parameters in the implemented algorithms. Table 4.1 shows the list of important parameters along with the possible range of assigned values (some of which are a single constant value) and the reason for selecting these values. The simulation systematically tries all integer values in the range of all parameters for every input macronuclear/micronuclear gene pair. If eventually preferred values for these parameters are determined, it could be possible to alter the simulation to only try those values, significantly speeding up the simulation. These important parameters are explained further in Section 4.2.

Table 4.1: List of the important parameters

Parameter name (purpose)	Assigned Value	Reason for assigning this value
bp_size (the size of scnRNAs)	28	From the Scan RNA model (Section 2.1.1), it is known that the size of scnRNAs is 28 or 29 bp.
threshold_MDS (the minimum score of the semi-global alignment needed for sufficient similarity between scnRNAs and old_mac)	1 to bp_size	There is no evidence known in the literature as to what value might be appropriate for this variable, so a brute force approach is taken and a large range for this parameter is tested.
threshold_IES (the minimum score of the semi-global alignment needed for sufficient similarity between filtered scnRNAs and new_mic)	1 to bp_size	There is no evidence known in the literature as to what value might be appropriate for this variable, so a brute force approach is taken and a large range for this parameter is tested.
minP (the minimum size of a pointer)	2	In Chapter 1, it is mentioned that the minimum size of a pointer is 2 bp [21].
maxP (the maximum size of a pointer)	20	In Chapter 1, it is mentioned that the maximum size of a pointer is 20 bp [21].
diff (the minimum difference between two consecutive IESs)	2	There is no evidence in the literature as to what value might be appropriate for this variable, so a simplifying assumption is made.
nh (the size of the neighbourhood, indicates the area around where filtered scnRNAs match the developing macronucleus)	1 to 20	There is no evidence known in the literature as to what value might be appropriate for this variable, so a brute force approach is taken and a large range for this parameter is tested.
window_size (the initial length for selecting a MDS from the variable old_mac)	5	There is no evidence in the literature as to what value might be appropriate for this variable, so a simplifying assumption is made.

4.2 Implemented algorithms

As discussed above, the major functions of the pipeline are the *scnRNA*, *IES*, *IES_deletion*, *rearrangement* and *correction* functions demonstrating the mechanisms of Definitions 1, 2, 3, 4 and 5 respectively. Each of these functions are described individually from Subsection 4.2.1 to Subsection 4.2.5. The high-level pseudocode of the simulation is given in Algorithm 1.

From Algorithm 1, it is evident that the implemented algorithm takes three input strings: *old_mic*, *old_mac*, and *new_mic*. Among these input strings, *old_mic* and *old_mac* are a single matching micronuclear gene and macronuclear gene. And for lack of data, *new_mic* is considered to be the same as the *old_mic*, though biologically they are slightly different, as the micronucleus is altered after mating. At the beginning of the algorithm, some variables are initialized to hold intermediate results like *scnRNAs* and *IESs* which are both one dimensional arrays of strings, and *dev_mac1*, *dev_mac2*, *new_mac* as strings. Then, some important parameters are assigned with desired values (see Table 4.1). The simulation systematically varies all parameters in Table 4.1 in all possible combinations for every macronuclear and micronuclear gene pair, and for each will calculate three outputs: *acc1*, *acc2* and *acc3*. These values represent the similarity of the old macronucleus to the developing macronucleus at three different stages and time orders: first after deleting

Algorithm 1 Implemented_Algorithm (*old_mic, old_mac, new_mic*)

```
INIT scnRNAs, IESs, dev_mac1, dev_mac2, new_mac;
INIT acc1, acc2, acc3;
bp_size ← 28;
minP ← 2;
maxP ← 20;
diff ← 2;
window_size ← 5;
scnRNAs ← scnRNA(old_mic, bp_size);
for i ← 1 to bp_size do
    threshold_MDS ← i;
    IESs ← IES(old_mac, scnRNAs, threshold_MDS, bp_size);
    for j ← 1 to bp_size do
        threshold_IES ← j;
        for nh ← 1 to 20 do
            dev_mac1 ← IES_deletion(new_mic, IESs, threshold_IES, bp_size, minP, maxP, diff, nh);
            acc1 ← global_alignment(old_mac, dev_mac1);
            dev_mac2 ← rearrangement(dev_mac1, old_mac, window_size);
            acc2 ← global_alignment(old_mac, dev_mac2);
            new_mac ← correction(dev_mac2, old_mac);
            acc3 ← global_alignment(old_mac, new_mac);
            printf(acc1,acc2,acc3);
        end for
    end for
end for
```

IESs from the micronucleus (*dev_mac1*), second after rearrangement based on templates (*dev_mac2*), and third after proof checking (*new_mac*). These three similarity values are discussed in further detail in Chapter 5.

The *scnRNA* function generates all possible scnRNAs through a "sliding window" technique. In the *scnRNA* function, *old_mic* is divided according to the *bp_size* (28) variable to construct a set of *scnRNAs*. This function is explained and justified in Subsection 4.2.1. After generating *scnRNAs*, a loop is used to systematically change *threshold_MDS* by increments of 1 (as there is no evidence that it is known in the literature as to what value might be appropriate for this variable). Indeed, *threshold_MDS* represents the minimum score of the semi-global alignment needed for sufficient similarity between *scnRNAs* and *old_mac*. Within this loop, *scnRNAs* is taken along with *old_mac*, *bp_size*, and *threshold_MDS* as inputs to the *IES* function to generate *IESs* (IES specific sequences). The *IES* function is explained and justified in Subsection 4.2.2. Then, a nested loop is used to change *threshold_IES* (as there is no evidence in the literature as to what value might be appropriate for this variable). Indeed, *threshold_IES* represents the minimum score of the semi-global alignment needed for sufficient similarity between *IESs* and *new_mic*. Within this nested loop, another loop is run to change the size of neighbourhood (*nh*) as there is no evidence in the literature as to what value might be appropriate for this variable. The size of neighbourhood indicates the area around where filtered scnRNAs match the developing macronucleus. Within this loop, *new_mic*, *IESs*, *threshold_IES*, *bp_size*, *minP*, *maxP*, *diff*, *nh* are used as inputs to the *IES_deletion* function to generate *dev_mac1*. This function is explained

and justified in Subsection 4.2.3. Then, the similarity between *old_mac* and *dev_mac1* is calculated by using the *global_alignment* function and the result is stored in *acc1*. After that, the *rearrangement* function is used to generate the *dev_mac2* by taking *dev_mac1*, *old_mac*, and *window_size* as inputs. The *rearrangement* function is explained and justified in Subsection 4.2.4. Then, the similarity between *old_mac* and *dev_mac2* is calculated by using the *global_alignment* function and the result is stored in *acc2*. After that, the *correction* function is applied to correct *dev_mac2* and the *new_mac* is generated based on the template (*old_mac*). The *correction* function is explained and justified in Subsection 4.2.5. Then, the similarity between *old_mac* and *new_mac* is calculated by using the *global_alignment* function and the result is stored in *acc3*. Finally, the values of *acc1, acc2*, and *acc3* are displayed before ending the innermost nested loop. After finishing the outermost loop (used to vary *threshold_MDS*), *acc1*, *acc2*, and *acc3* are printed for the specific values of *threshold_MDS*, *threshold_IES* and *nh*.

4.2.1 The *scnRNA* function

The main objective of this function is to demonstrate the mechanism of Definition 2 which is the construction of scnRNAs from the old micronucleus. Here, one of the input strings "*old_mic*" is divided into strings of a certain length based on the *bp_size* variable and then those strings are stored in a one-dimensional array. Algorithm 2 gives the pseudocode of the *scnRNA* function. It takes *old_mic* and *bp_size* as inputs. At the very beginning of the algorithm, a variable called *scnRNAs* is initialized. Then, a loop is run from *0* to the length of *old_mic* minus *bp_size*. After that, a variable (k) is declared to make each possible substring of length *bp_size* from *old_mic*. Then, a inner loop is run from *0* to *bp_size* and in this loop strings are stored in *scnRNAs* from *old_mic*. After finishing both loops the *scnRNA* function returns the variable *scnRNAs* to its calling function. An example of this function is illustrated in Figure 4.2. The reverse complements are not added yet to *scnRNAs*, as is done in the biological 2JLP model. For simplicity, the reverse complements are considered in the *rearrangement* function.

Algorithm 2 scnRNA (*old_mic, bp_size)*

```
1: INIT scnRNAs;
2: for i ← 0 to (length[old_mic] − bp_size) do
3:     k ← i;
4:     for j ← 0 to bp_size do
5:         scnRNAs[i][j] ← old_mic[k];
6:         k ← k + 1;
7:     end for
8: end for
9: return scnRNAs;
```

4.2.2 *IES* function

The *IES* function demonstrates the functionality of Definition 3 by comparing all elements of the variable *scnRNAs* (output of the *scnRNA* function) with the input string *old_mac* and filtering out a set of largely

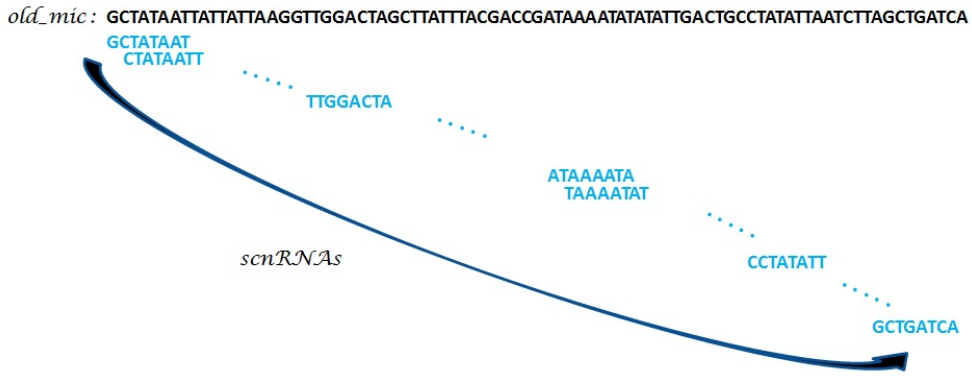


Figure 4.2: An example of the *scnRNA* function. The example shows that the *old_mic* is divided into substrings (*scnRNAs*) based on the *bp_size* variable.

IES specific sequences (*IESs*). Algorithm 3 gives the pseudocode of this function. This function compares each element from *scnRNAs* with *old_mac*, and if there is a match that is "similar enough", it gets filtered out as it will largely be MDS specific, and the scnRNAs that remain are placed in the array *IESs*. To test the similarity of an element from *scnRNAs* to the variable *old_mac*, the *semi_global_alignment* function is used and the result is compared with *threshold_MDS*. If the result is greater than or equal to the value of *threshold_MDS* then that element is considered to be "similar enough" with *old_mac*.

The semi-global alignment technique is used to find an element, if it exists, from the variable *scnRNAs* where either the prefix or suffix is similar to a substring of the variable *old_mac*. This function is also run several times by changing the value of *threshold_MDS* to do approximate string matching. Indeed, it is known that allelic variation (an allele is a variant of a gene where the DNA sequence differs between two or more variants) may occur in a biological sequence. Interestingly, some MDSs are less than 28 bp long. It is desirable to know how the simulation works on those, and *threshold_MDS* is the parameter that is giving that information. Furthermore, the case where a *scnRNA* contains part of an MDS and part of an IES is of interest. If this is the case, then a semi-global alignment between the *scnRNA* and *old_mac* will give a score similar to the length of the MDS portion. Then systematically varying this parameter is desirable. This parameter is needed because it helps to filter out a set of largely IES specific sequences which plays a significant role in IES deletion from the new micronucleus.

Algorithm 3 IES $(old\_mac, scnRNAs, threshold\_MDS, bp\_size)$

```
1: INIT IESs, max_max;
2: j ← 0;
3: for i ← 0 to no_of[scnRNAs] do
4:     max_max ← semi_global_alignment(old_mac, scnRNAs[i]);
5:     if max_max < threshold_MDS then
6:         IESs[j] ← scnRNAs[i];
7:         j ← j + 1;
8:     end if
9: end for
10: return IESs;
```

An example of the *IES* function is illustrated in Figure 4.3. The algorithm is accomplishing the main task of Definition 3 in the formal model (using semi-global alignment to represent similarity).

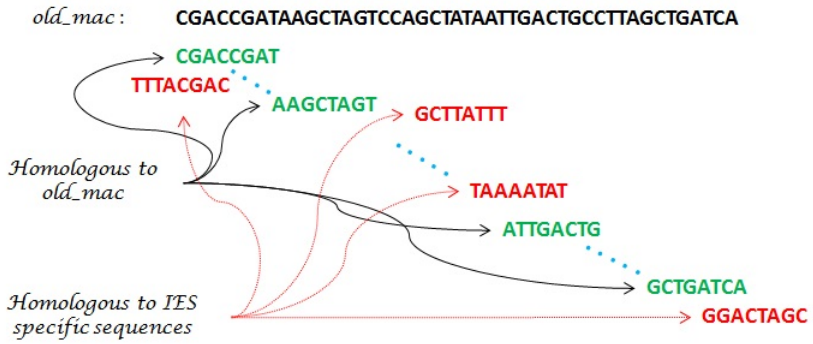


Figure 4.3: An example of the *IES* function. The elements of *scnRNAs* are in green if the result of the *semi_global_alignment* function is greater than or equal to the variable *threshold_MDS*. Otherwise, they are in red and considered largely similar to an IES specific sequence.

4.2.3 The *IES_deletion* function

The *IES_deletion* function mainly deletes IES specific sequences by comparing all strings of *IESs* (output of the *IES* function) with the input string *new_mic*. This function represents an algorithm inspired by both Definition 4 and 5. This task is accomplished by breaking it into two main steps. The first step captures Definition 4 by comparing all strings of *IESs* with the *new_mic* and performs a marking (by keeping track of the start and end positions in *new_mic* where it matches the string of *IESs*). The second step captures Definition 5 by removing substrings from the *new_mic* if it has a repeated string (a potential pointer sequence, or a cryptic pointer) of a certain length ($\geq minP$ and $\leq maxP$) close to the ends of the marked portion (where "close to" means within the size of neighbourhood *nh*). The experimental result of Möllenbeck et al. [16] shows that usually cryptic or real pointers are present around the MDS-IES junction and the chances of considering longer repeats around the MDS-IES junction as a cryptic pointer is higher. Also, it is possible that the repeated sequence is a part of an IES (which would result in a portion of the IES remaining after deletion), but it is also possible that the repeated sequence could be part of an MDS as well (which would result in part of that MDS being missing). That is why in the algorithm, a parameter named "Neighbourhood (*nh*)" is taken to address the range of possible cryptic or real pointers in proximity to the marked portion (which is likely close to the MDS-IES junction). All neighbourhood values only up to 20 are tested.

Algorithm 6 depicts the pseudocode of this function. From the pseudocode it is viewable that this function takes eight inputs and gives one output. Inputs to this function are *new_mic*, *IESs*, *threshold_IES*, *bp_size*, *minP*, *maxP*, *diff*, and *nh* and the output is *dev_mac1*. At the beginning of this function, five variables are defined. Among these, *dev_mac1* represents a string for holding the final result string, *st_po*, and *end_po* represents two array of integers consecutively for storing the start and end positions of the marked IES specific sequences from the *finding_IES* function. The variables *cut_point1* and *cut_point2* are also defined

as arrays of integers to store start and end positions of marked strings in *new_mic* which is used later for removing IES specific sequences. These two values are found by applying the *selecting_IES* function. At this point in the algorithm, the strings are needed to cut according to *cut_point1* and *cut_point2* from the *new_mic* and it is stored into the *dev_mac1* variable. At the end of this algorithm, the function returns the variable *dev_mac1* to its calling function. An example of this function is illustrated in Figure 4.4.

During the calling of the *finding_IES* function (Algorithm 4), three variables (*new_mic, IESs, threshold_IES*) are given as parameters. In the function, four more new variables are defined: *max_vrow*, *max_position*, *start_position*, and *end_position*. Then a loop is started for doing approximate string matching of all elements of *IESs* with *new_mic* by calling the function *semi_global_alignment*. Here the semi-global alignment technique is chosen to find an element, if it exists, from the variable *IESs* that has a suffix or prefix similar to a substring of the variable *new_mic*. Each comparison result is stored in the variable *max_vrow* and *max_position*. The variable *max_vrow* stores the maximum alignment score and the variable *max_position* stores a position from the variable *new_mic* which indicates the ending position of the maximum similar region with an element of IESs. Then, it is checked whether the element of *IESs* is similar to a substring of the variable *new_mic* or not by comparing *max_vrow* with *threshold_IES*. Here, the parameter *threshold_IES* is considered as the comparing factor because of the allelic variation in the biological sequence. Interestingly, some IESs are less than 28 bp long, and some scnRNAs contain part of an IES and part of an MDS. It is desirable to know how the simulation works with such sequences. Indeed, *threshold_IES* is the parameter which can adjust for various amounts of similarity between the IESs and the scnRNAs. Now, if the value of *max_vrow* is greater than or equal to the value of *threshold_IES* then the start and end positions of the marked IES specific sequences are stored consecutively in the variable *start_position* and *end_position*. Finally after finishing the loop, the algorithm returns the variables *start_position* and *end_position* to its calling function.

During the calling of the *selecting_IES* function (Algorithm 5), nine variables (*new_mic, IESs, st_po, end_po, diff, minP, maxP, bp_size, nh*) are used as parameters. At the beginning of this algorithm, eight new variables are initialized. Among these, *s_IES* and *e_IES* are integer variables to hold the first and last marked *IESs* array number, if there are multiple consecutive *IESs* array marked in the variable *new_mic*. Variable *max_max* is an integer variable to store the score of the *local_alignment* function between the start and end selected portions based on the neighbourhood close to the ends of the marked IES specific sequences in the variable *new_mic*. Variable *max_row* and *max_col* are also integer variables to store consecutively the end position of the first repeat from the selected portions, and the end position of the second repeat from the selected portions. Two integer arrays *cut_point1* and *cut_point2* are also defined to hold initial and end positions of an IES specific sequence in the variable *new_mic*. Integer variable *counter* indicates the index position of *cut_point1* and *cut_point2*. After the declaration, a loop is started to go through all marked IES specific sequences and to calculate *cut_point1, cut_point2*. Then, a nested loop is run for finding the first and last marked IES numbers if there are consecutive marked IESs present in a portion of the variable *new_mic*. Here, the consecutiveness is measured based on the parameter *diff*. If the distance between two adjacent

elements of the marked IESs is less than the value of *diff* then they are classified as being from the same IES specific sequence and the loop is continued. And if not then the loop is stopped and the first and last marked IES numbers are stored. After getting the first and last marked IESs among consecutive marked IESs, the start and the end positions of the first and last marked IES are collected respectively from the variables *st_po, end_po*. Then, two strings (*sWindow, eWindow*) are initialized for storing the selected start and end portions based on start and end positions with respect to the variable *nh* (neighbourhood). Two individual loops are run for selecting start and end selected portions close to the ends of the marked IES specific sequences. Then, the local alignment algorithm is applied over these two selected portions to find the longest similar portion. This is done because it is known that longer repeats are favourable at the time of IES deletion based on scnRNAs from the new micronucleus [16]. After performing the alignment operation, it is checked whether or not the alignment score (*max_max*) is between the minimum (*minP*) and the maximum (*maxP*) pointer values. If it is, then *cut_point1* is calculated by adding the start position of the first marked IES with *max_row* (maximum score position of the first marked IES) and *cut_point2* is calculated by adding the end position of the last marked IES with *max_col* (maximum score position of the last marked IES). At the end of the outermost loop, the function returns *cut_point1* and *cut_point2* to its calling function.

In the *IES_deletion* function, initially the IES specific sequences are selected by using both the *finding_IES* and the *selecting_IES* functions. Then those IES specific sequences are removed from the variable *new_mic*. As seen from Section 2.1.3, this accomplishes the third step of the 2JLP model.

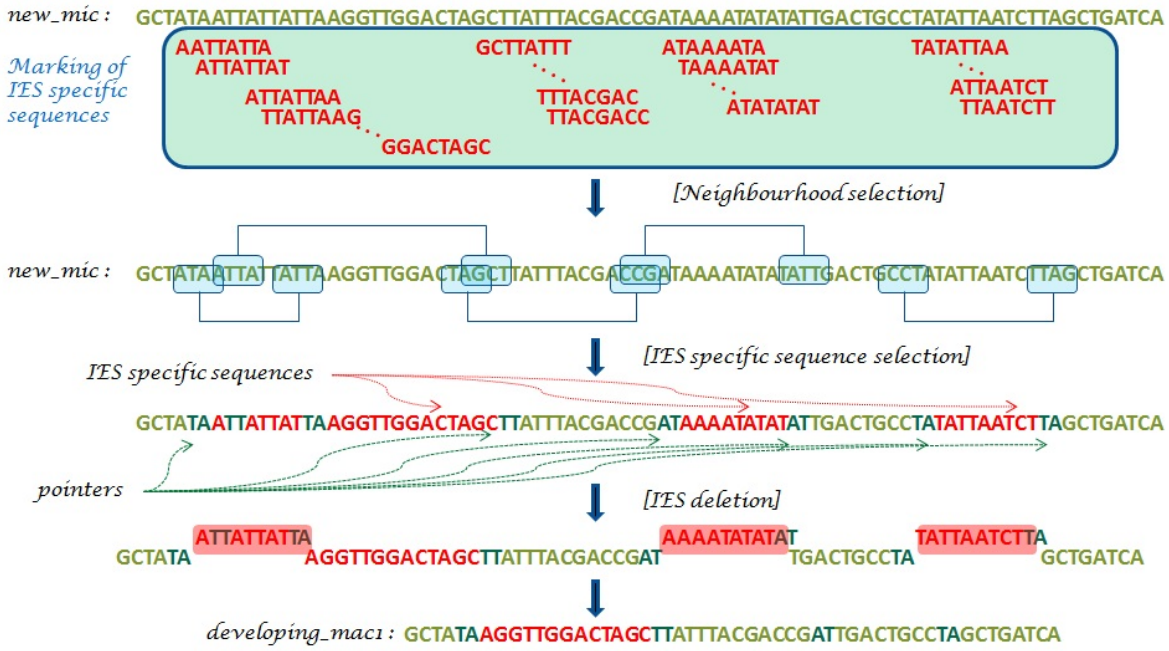


Figure 4.4: An example of the *IES_deletion* function. At the beginning, all elements of *IESs* are aligned according to *end_po*. In the figure, *IESs[s_IES]* and *IESs[e_IES]* are marked with blue rectangles and pointers are marked as green characters. Then, IES specific sequences are considered from the end of each starting pointer to the beginning of each ending pointer. After marking the IES specific sequences, those sequences are removed from *new_mic* and *dev_mac1* is constructed.

Algorithm 4 finding_IES ($new\_mic, IESs, threshold\_IES$)

```
1: INIT max_vrow, max_position, start_position, end_position;
2: l ← 0;
3: for i ← 0 to no_of[IESs] do
4:     (max_vrow, max_position) ← semi_global_alignment(new_mic, IESs[i]);
5:     if max_vrow ≥ threshold_IES then
6:         end_position[l] ← max_position;
7:         start_position[l] ← end_position[l] − max_vrow;
8:         l ← l + 1;
9:     end if
10:    i ← i + 1;
11: end for
12: result[0] ← end_position;
13: result[1] ← start_position;
14: return result;
```

4.2.4 The *rearrangement* function

The main purpose of this function is to rearrange MDSs from *dev_mac1* based on the old macronucleus (*old_mac*) and to perform the fourth step of the 2JLP model (Section 2.1.3). It is beyond the scope of this thesis (or current biological knowledge) to be able to predict the order in which MDSs descramble. Therefore, MDSs are instead rearranged based on randomness. A substring of length *window_size* is randomly selected from the variable *old_mac*. The parameter *window_size* is used for selecting a certain substring (or template) from the old macronucleus to search for a similar substring in *dev_mac1*. Then, it is desirable to find that portion in *dev_mac1* by performing the *semi_global_alignment* function. For finding a substring in a string, the semi-global alignment technique is applied. If that substring is found then the *tracking* function (discussed below) is applied. Otherwise, the reverse complement of that selected substring is taken and it is desired to find it in the variable *dev_mac1*. This reverse operation is done because sometimes MDSs can be present in the new micronucleus in an inverted order as mentioned in Section 2.2.2. If that substring is still not found, then a random new substring is again selected from the old macronucleus and an alignment is performed. After finding the selected substring in *dev_mac1*, the alignment is extended to the right and left until there is a mismatch. When a mismatch is found in the left direction then the substring is marked in *old_mac* and the newly discovered substring's start and end positions are stored from *dev_mac1* as a node in a linked-list. Then, a substring is again randomly selected from the variable *old_mac* which is unmarked and the previously mentioned procedure is continued until there is no unmarked sequence left in the variable *old_mac*. For putting information regarding all discovered portions in the linked-list, a loop is run and two functions named *createNode* and *insertNodeBetween* are applied. After that, all nodes of the linked-list are rearranged based on *old_mac*. Finally, characters from the variable *dev_mac1* are retrieved by applying linked-list information and put into a new variable named *dev_mac2*. Algorithm 10 describes the pseudocode of this function.

At the time of rearrangement of the newly found portions from the variable *dev_mac1*, the data structure

Algorithm 5 selecting_IES ($new\_mic, IESs, st\_po, end\_po, diff, minP, maxP, bp\_size, nh$)

INIT $sIES, eIES, max\_max, max\_row, max\_col$;
INIT $cut\_point1, cut\_point2, counter$;
for $j \leftarrow 0$ to $no\_of[IESs]$ **do**
    $flag \leftarrow TRUE$;
    $sIES \leftarrow j$;
    while $flag$ **do**
        if $((end\_po[j+1] - end\_po[j]) < diff) \&\& (j < no\_of[IESs])$ **then**
            $j \leftarrow j + 1$;
        else
            $flag \leftarrow FALSE$;
        end if
    end while
    $eIES \leftarrow j$;
    $sInd \leftarrow (st\_po[sIES] - nh)$;
    $sp \leftarrow sInd$;
    $eInd \leftarrow (end\_po[eIES] + 1 - nh)$;
    $ep \leftarrow eInd$;
    INIT $sWindow, eWindow$;
    for $x \leftarrow 0$ to $(nh * 2) - 1$ **do**
        $sWindow[x] \leftarrow new\_mic[sInd]$;
        $sInd \leftarrow sInd + 1$;
    end for
    for $y \leftarrow 0$ to $(nh * 2) - 1$ **do**
        $eWindow[y] \leftarrow new\_mic[eInd]$;
        $eInd \leftarrow eInd + 1$;
    end for
    $(max\_max, max\_row, max\_col) \leftarrow local\_alignment(sWindow, eWindow)$;
    if $(max\_max \geq minP) \&\& (max\_max \leq maxP)$ **then**
        $cut\_point1[counter] \leftarrow (sp + max\_row)$;
        $cut\_point2[counter] \leftarrow (ep + max\_col)$;
        $counter \leftarrow counter + 1$;
    end if
    $j \leftarrow j + 1$;
end for
$result[0] \leftarrow cut\_point1$;
$result[1] \leftarrow cut\_point2$;
return $result$;

Algorithm 6 IES_deletion $(new\_mic, IESs, threshold\_IES, bp\_size, minP, maxP, diff, nh)$

```
1: INIT dev_mac1, end_po, st_po;
2: INTI cut_point1, cut_point2;
3: (end_po, st_po) ← finding_IES(new_mic, IESs, threshold_IES);
4: (cut_point1, cut_point2) ← selecting_IES(new_mic, IESs, st_po, end_po, diff, minP, maxP, bp_size, nh);
5: for k ← 0 to length[cut_point1] do
6:     if (k + 1) < length[cut_point1] then
7:         if k == 0 then
8:             for n ← 0 to cut_point1[k] do
9:                 dev_mac1[n] ← new_mic[n];
10:            end for
11:            for m ← cut_point2[k] to cut_point1[k + 1] do
12:                dev_mac1[n] ← new_mic[m];
13:                n ← n + 1;
14:            end for
15:        else
16:            for m ← cut_point2[k] to cut_point1[k + 1] do
17:                dev_mac1[n] ← new_mic[m];
18:                n ← n + 1;
19:            end for
20:        end if
21:    else
22:        for m ← cut_point2[k] to length[new_mic] do
23:            dev_mac1[n] ← new_mic[m];
24:            n ← n + 1;
25:        end for
26:    end if
27: end for
28: return dev_mac1;
```

of a linked-list is applied instead of the intermolecular and intramolecular operations which are used in the computational model (Definition 8). This is done because it is not known how a ciliate keeps track of its MDSs in a developing macronucleus including which order they need to rearrange, how they determine which portions are already arranged and which are not, and the parts that are rearranged in parallel. In the algorithm, a single temporal order is not assumed for the rearrangement.

During the calling of the *tracking* function (Algorithm 7), six variables (*po*, *m_value*, *m_po*, *window_size*, *old_mac*, *dev_mac1*) are taken as parameters. At the beginning of the pseudocode, *t_old* (for tracking previously visited substrings of *old_mac*), *t_dev* (for tracking previously visited substrings of *dev_mac1*), *end_o* (array for storing the end positions of selected portions in *old_mac*), *start_o* (array for storing the start positions of selected portions in *old_mac*), *end_n* (array for storing end positions of newly found portions in *dev_mac1*), and *start_n* (array for storing start positions of newly found portions in *dev_mac1*) are defined. After initialization, a loop is run as it may be possible to find multiple portions in the variable *dev_mac1*. This loop is run until all positions having the maximum score *m_value* (from the previously performed semi-global alignment) in the variable *dev_mac1* are checked. Then, it is checked whether or not the right adjacent characters from both selected portions of *old_mac* and the newly found portion of *dev_mac1* are the same or not. If they are the same, then it keeps searching in the right direction, otherwise searching the adjacent characters to the left until a mismatch is found. The final start and end positions of the selected portion are stored from *old_mac* into the variable *start_o* and *end_o*. The final start and end positions of the newly found portion are stored from *dev_mac1* in the variables *start_n* and *end_n*. This extension to the right and to the left is performed for all newly found portions of the variable *dev_mac1*. Finally, the portion that is the largest is selected and saved into the variables *t_old* and *t_dev*. After that, *t_old* and *t_dev* are returned to its calling function. The main purpose of this function is to track all MDSs of the variable *dev_mac1* based on the variable *old_mac* and send it to the *rearrangement* function. Figure 4.5 and Figure 4.6 illustrate the functionality of this function.

The linked list data structure used in the *rearrangement* function is briefly discussed. The data structure of the linked-list is given below:

```
typedef struct dlist {
    int element[2];
    int position[2];
    struct dlist *next;
    struct dlist *prev;
} NODE;
```

A node of the linked-list stores six pieces of data. These are the start and end positions of the selected portion in *old_mac*, the start and end positions of the newly found portion in *dev_mac1*, and the address of the previous and next nodes.

For creating a node, a function called *createNode* (Algorithm 8) is used. During the calling of the

createNode function, the start and end positions of the selected portion in *old_mac* (*item*), and the start and end positions of the newly found portion in *dev_mac1* (*pos*) are given as parameters. Here, values are assigned in *element* and *position* based on *item* and *pos* respectively.

Now for performing the rearrangement of all newly found portions from *dev_mac1*, after creating a node it has to connect with its previous and next nodes as the previous and next MDSs are connected. To accomplish this task, a function named *insertNodeBetween* (Algorithm 9) is introduced. During the calling of the *insertNodeBetween* function, *head* (start node of the linked-list), *item* (the start and end positions of the selected portion in *old_mac*) and *pos* (the start and end positions of the newly found portion in *dev_mac1*) are used as parameters. At first, a new node is created by putting all current node information into the *createNode* function. Then, the appropriate position of this node is searched by comparing the *element* value of the new node with all *element* values of existing nodes. After finding the appropriate position of the new node, the previous and next nodes of that node are updated accordingly.

Figure 4.5: Process of finding MDSs in *dev_mac1*.

4.2.5 The *correction* function

The main purpose of this function is to demonstrate the fifth step (the proofreading step) of the 2JLP model (Section 2.1.3). In this function, the final macronucleus (*final_mac*) is generated by comparing *dev_mac2* and *old_mac*. Intuitively, based on *old_mac* extra characters are removed from *dev_mac2* and missing characters are inserted into *dev_mac2*.

Algorithm 11 describes the pseudocode of this function. In this function, *dev_mac2* and *old_mac* are arguments. Initially, two variables: *new_mac* (for holding the final macronucleus) and *matrix* are defined. The *global_alignment* function is applied for performing the pairwise sequence alignment between *dev_mac2* and *old_mac*.

Algorithm 7 tracking $(po, m\_value, m\_po, w\_size, old\_mac, d\_mac)$

INIT t_old, t_dev;
INIT $end\_o, start\_o, end\_n, start\_n$;
$count \leftarrow 0$;
$temp \leftarrow 0$;
while all positions having m_value are not checked **do**
    $max\_max \leftarrow m\_value$;
    $n \leftarrow (po + w\_size + 1)$;
    $m \leftarrow (m\_po[count] + 1)$;
    while $(old\_mac[n] == d\_mac[m])\&\&(n \neq length[old\_mac])\&\&(m \neq length[d\_mac])$ **do**
        $n \leftarrow n + 1$;
        $m \leftarrow m + 1$;
        $max\_max \leftarrow max\_max + 1$;
    end while
    $end\_o[count] \leftarrow n - 1$;
    $end\_n[count] \leftarrow m - 1$;
    $n \leftarrow (po - 1)$;
    $m \leftarrow (m\_po[count] - (w\_size + 1))$;
    while $(old\_mac[n] == d\_mac[m])\&\&(n \geq 0)\&\&(m \geq 0)$ **do**
        $n \leftarrow n - 1$;
        $m \leftarrow m - 1$;
        $max\_max \leftarrow max\_max + 1$;
    end while
    $start\_o[count] \leftarrow n + 1$;
    $start\_n[count] \leftarrow m + 1$;
    if $max\_max > temp$ **then**
        $temp \leftarrow max\_max$;
        $index \leftarrow count$;
    end if
    $count \leftarrow count + 1$;
end while
$t\_old[0] \leftarrow start\_o[index]$;
$t\_old[1] \leftarrow end\_o[index]$;
$t\_dev[0] \leftarrow start\_n[index]$;
$t\_dev[1] \leftarrow end\_n[index]$;
$result[0] \leftarrow t\_old$;
$result[1] \leftarrow t\_dev$;
return $result$;

Algorithm 8 createNode $(item, pos)$

NODE $newNode \leftarrow NULL$;
$(newNode \rightarrow element[0]) \leftarrow item[0]$;
$(newNode \rightarrow element[1]) \leftarrow item[1]$;
$(newNode \rightarrow position[0]) \leftarrow pos[0]$;
$(newNode \rightarrow position[1]) \leftarrow pos[1]$;
$(newNode \rightarrow next) \leftarrow NULL$;
$(newNode \rightarrow prev) \leftarrow NULL$;
return $newNode$;

Algorithm 9 insertNodeBetween $(head, item, pos)$

```
1: NODE newNode ← NULL;
2: NODE runNode ← head;
3: newNode ← createNode(item, pos);
4: if head == NULL then
5:     return newNode;
6: else
7:     while ((runNode → next) ≠ NULL)&&((runNode → element[0]) ≤ item[0]) do
8:         runNode ← (runNode → next);
9:     end while
10:    if ((runNode → next) == NULL)&&((runNode → element[0]) > item[0]) then
11:        if ((runNode → prev) ≠ NULL) then
12:            (newNode → prev) ← (runNode → prev);
13:            ((newNode → prev) → next) ← newNode;
14:            (newNode → next) ← runNode;
15:            (newNode → prev) ← newNode;
16:        else
17:            (newNode → next) ← runNode;
18:            (newNode → prev) ← newNode;
19:            head ← newNode;
20:        end if
21:    else if ((runNode → next) == NULL)&&((runNode → element[0]) < item[0]) then
22:        (runNode → next) ← newNode;
23:        (newNode → prev) ← runNode;
24:    else
25:        if ((runNode → prev) == NULL) then
26:            (runNode → prev) ← newNode;
27:            (newNode → next) ← runNode;
28:            head ← newNode;
29:        else
30:            (newNode → next) ← runNode;
31:            (newNode → prev) ← (runNode → prev);
32:            (runNode → prev) ← newNode;
33:            ((newNode → prev) → next) ← newNode;
34:        end if
35:    end if
36: end if
37: return head;
```

Algorithm 10 rearrangement $(dev\_mac1, old\_mac, window\_size)$ [continued on the next page]

INIT $dev\_mac2, track\_o, track\_n, is\_used$;
INIT po, m_value, m_po;
$ct \leftarrow 0$;
$w\_size \leftarrow window\_size$;
$d\_mac \leftarrow dev\_mac1$;
$max \leftarrow (length[old\_mac] - (w\_size))$;
while all elements of the *old_mac* or the *d_mac* are not traversed **do**
    while randomly selected substring from *old_mac* is marked **do**
        $po \leftarrow GetRand(0, max, is\_used)$;
        $track\_o[ct][0] \leftarrow po$;
        $track\_o[ct][1] \leftarrow (po + w\_size)$;
    end while
    $(m\_value, m\_po) \leftarrow semi\_global\_alignment(d\_mac, old\_mac, track\_o[ct])$;
    if $(m\_value == w\_size)$ **then**
        $(track\_o[ct], track\_n[ct]) \leftarrow tracking(po, m\_value, m\_po, w\_size, old\_mac, d\_mac)$;
        for $j \leftarrow track\_o[ct][0]$ to $track\_o[ct][1]$ **do**
            $is\_used[j] \leftarrow 1$;
            $j \leftarrow j + 1$;
        end for
        $ct \leftarrow ct + 1$;
    else
        Reverse the selected string and do the same operation of the previous If;
    end if
end while
NODE $head \leftarrow NULL$;
for $i \leftarrow 0$ to $ct - 1$ **do**
    if $i == 0$ **then**
        $head \leftarrow createNode(track\_o[i], track\_n[i])$;
    else
        $head \leftarrow insertNodeBetween(head, track\_o[i], track\_n[i])$;
    end if
end for

```
33: NODE runNode ← head;
34: in ← 0;
35: while runNode ≠ NULL do
36:     if runNode → element[0] > runNode → element[1] then
37:         for s ← (runNode → position[1]) to (runNode → position[0]) do
38:             if d_mac[s] == 'A' then
39:                 dev_mac2[in] ← 'T';
40:             else if d_mac[s] == 'T' then
41:                 dev_mac2[in] ← 'A';
42:             else if d_mac[s] == 'C' then
43:                 dev_mac2[in] ← 'G';
44:             else if d_mac[s] == 'G' then
45:                 dev_mac2[in] ← 'C';
46:             end if
47:             in ← in + 1;
48:             s ← s − 1;
49:         end for
50:     else
51:         for s ← (runNode → position[0]) to (runNode → position[1]) do
52:             dev_mac2[in] ← d_mac[s];
53:             in ← in + 1;
54:             s ← s + 1;
55:         end for
56:     end if
57:     runNode ← (runNode → next);
58: end while
59: return dev_mac2;
```

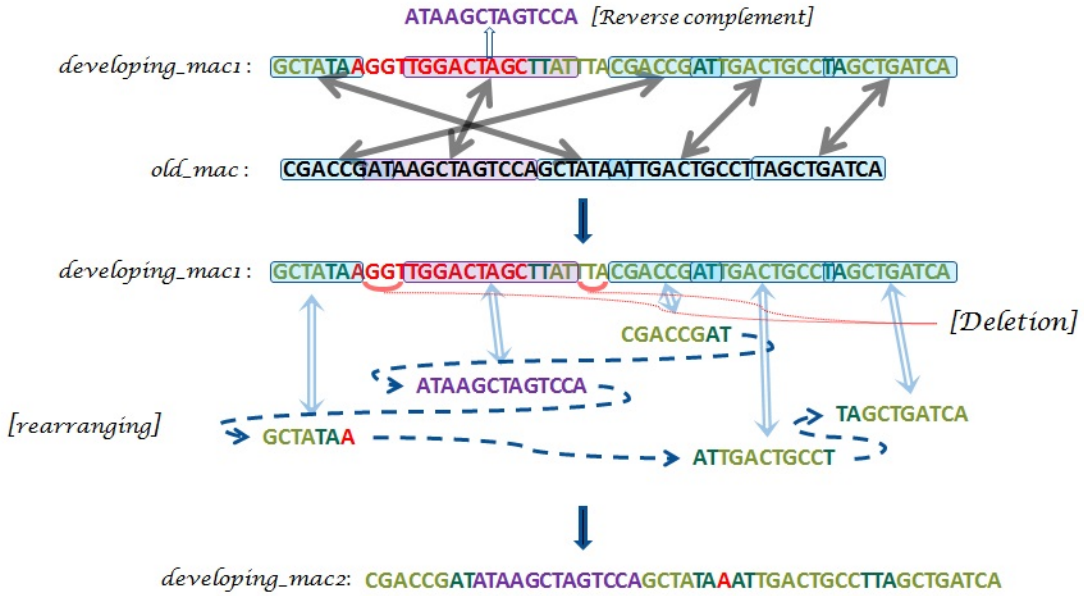


Figure 4.6: Process of development of *dev_mac2*. Initially, exact positions of MDSs are traced out in both *old_mac* and *dev_mac1*. Then, MDSs are rearranged and extra strings in *dev_mac1* are removed on the basis of *old_mac*. Finally, after rearranging MDSs of *dev_mac1*, *dev_mac2* is generated.

When implementing global alignment, a matrix is created where cell $[i][j]$ holds the score of an optimal alignment of the first i characters of the first sequence with the first j of the other sequence. But also, for the purposes of backtracking and actually finding an optimal alignment, some implementations also create a second matrix where the value at cell $[i][j]$ contains the information regarding the final position of an optimal alignment of the first i characters of the first sequence with the first j of the other. Indeed, if this matrix keeps track of whether such an alignment ends with a gap along one sequence, a gap along the other, or an aligned character on both sequences, then backtracking to find an alignment is easy. It is assumed that the *global_alignment* function returns this matrix and stores it in the variable *matrix*. Then, the matrix contains a 1 to indicate that there is an extra character in *dev_mac2*, it contains a 2 to indicate that there is either a correct or a mutated character in *dev_mac2*, it contains a 3 to indicate that there is a gap in *dev_mac2*, and it contains a 0 to indicate cell [0][0]. Indeed, cell [0][0] indicates the end of backtracking operation. For putting values in *new_mac* based on the variable *matrix*, a loop (for doing the backtracking operation) is run until 0 is found in *matrix*. After constructing *new_mac*, it returns to its calling function.

Figure 4.7: An example of the *correction* function. Initially, missing and extra elements are found in *dev_mac2*. Then extra elements are removed, and missing elements are inserted into *dev_mac2*. Finally, the final output *new_mac* is generated.

Algorithm 11 correction (dev_mac2, old_mac)

```
1: INIT new_mac, matrix;
2: matrix ← global_alignment(dev_mac2, old_mac);
3: i ← (length[dev_mac2] − 1);
4: j ← (length[old_mac] − 1);
5: k ← j;
6: while (k ≥ 0)&&(matrix[i][j] ≠ 0) do
7:     if matrix[i][j] == 2 then
8:         new_mac[k] ← dev_mac2[i − 1];
9:         i ← i − 1;
10:        j ← j − 1;
11:        k ← k − 1;
12:    else if matrix[i][j] == 3 then
13:        new_mac[k] ← old_mac[j − 1];
14:        j ← j − 1;
15:        k ← k − 1;
16:    else if matrix[i][j] == 1 then
17:        i ← i − 1; ▷ k is not decremented, because the extra elements from dev_mac2 are removed in here
18:    end if
19: end while
20: return new_mac;
```

Chapter 5

Result Analysis

In the simulation, results are calculated at three different stages to measure the change from the new micronucleus to the new macronucleus. These three different stages are after the *IES_deletion* function, after the *rearrangement* function, and finally after the *correction* function. A term *accuracy* is defined to represent the degree to which descrambling has occurred at the various stages. The following section describes how to calculate accuracy and the findings from the simulation.

5.1 Accuracy

Accuracy is measured by calculating a pairwise sequence alignment between two input sequences. One of these sequences is the old macronucleus. And the other one is the output sequence coming from any of the following functions: the *IES_deletion* function, the *rearrangement* function, and the *correction* function. Then accuracy represents the similarity between the resultant sequence of the previously mentioned functions and the old macronucleus. Equation (5.1) indicates the formula for calculating accuracy where *alignment_score* indicates the pairwise alignment score and *old_mac_size* indicates the size of the old macronucleus.

$$accuracy = \frac{(alignment\_score \times 100)}{old\_mac\_size} \tag{5.1}$$

In the case of performing pairwise sequence alignment the following scoring scheme is used:

match	=	1
mismatch	=	-1
gap	=	- 2.

It should be noted that accuracy as calculated is not a percentage. However, if the sequence matches perfectly, the resulting accuracy is 100. On the other hand, if the sequence matches badly, it is possible to have a negative score. However, it is beneficial to use an alignment score instead of a measure such as "percent identity" so that gaps and mismatches are properly taken into account.

5.2 Data

Input data are collected from the IES MDS Database [5]. From there, 13 real micronucleus and macronucleus matching gene pairs of the ciliate *Oxytricha trifallax* are used by the simulation. Although this is a limited number of pairs of genes, the micronuclear data contains 40,844 base pairs and the macronuclear data contains 32,770 base pairs.

Among these 13 input pairs, pair number 7 (the Actin I gene) has a smaller micronuclear sequence (989 bp) than its macronuclear sequence (1553 bp) due to incomplete data. This pair will indeed appear differently in the results. There is a recent paper [26] on the sequencing and analysis of the macronuclear genome of the ciliate *Oxytricha trifallax*. But, the micronuclear genome is still unavailable. After availability of the micronuclear genome, it is possible to test the simulation with more data.

5.3 General analyses

This section presents accuracy values that are calculated in different stages of execution by the simulation.

At first, the accuracy value is calculated after finishing the *IES_deletion* function (Algorithm 4). This accuracy value is called *acc1*. Output of this function is *dev_mac1*. Figure 5.1 shows the calculation of *acc1* for an (fake) example. In the figure, the size of *old_mac* is 46 and the score from the pairwise sequence alignment is 5. This poor score is obtained because of the large number of gaps needed in every pairwise alignment. After putting these values in Equation 5.1, an accuracy value of 11 is obtained.

Then, accuracy is calculated for the second time after executing the *rearrangement* function (Algorithm 10). Output of this function is *dev_mac2*. After that, a pairwise sequence alignment is performed between the output sequence and *old_mac*. After getting the result from the sequence alignment, the accuracy value is calculated by applying Equation 5.1. Figure 5.2 shows the pairwise sequence alignment score of 27 for the above mentioned sequence pair and the accuracy value equals 87. This accuracy value is called *acc2*.

Finally, the accuracy after running the *correction* function is calculated. In this function, the proof reading is performed over *dev_mac2* based on *old_mac*. As described in Section 4.2.5, the function inserts missing and removes extra characters from *dev_mac2* by comparing it with *old_mac*. After executing this operation, *new_mac* is generated as output. Then, the pairwise sequence alignment is applied between the resultant sequence and *old_mac*. After obtaining the result of the sequence alignment, Equation 5.1 is applied to generate the accuracy value. Figure 5.3 shows the pairwise sequence alignment score of 46 for the above mentioned sequence pair and the accuracy value equals 100. This accuracy value is known *acc3*.

In the case of real data, the simulation is run for each pair with the changing values of *threshold_MDS*, *threshold_IES*, and neighbourhood (*nh*) in every possible way within a range for the parameters. Indeed, for each input pair 15,680 different values of the parameters are tested, each generating a value for *acc1*, *acc2*, and *acc3*. For every combination of the three parameters, the accuracies of the 13 gene pairs are added,

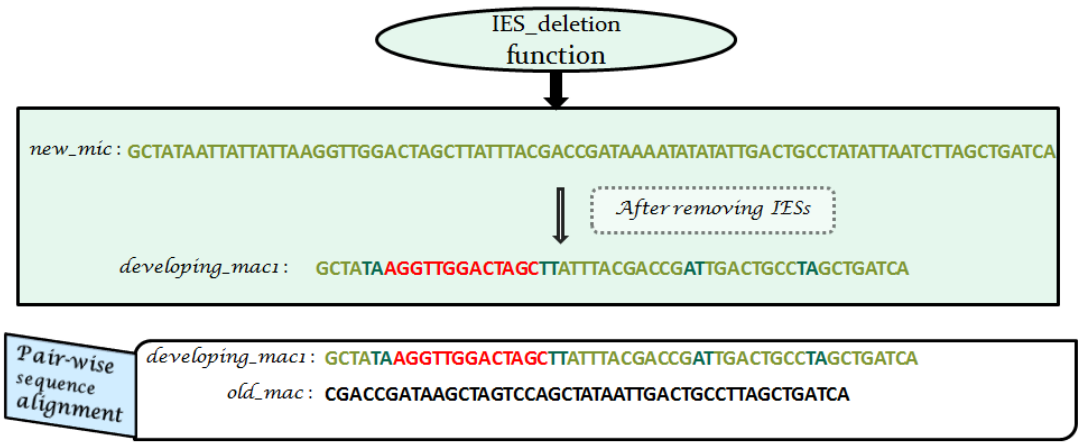


Figure 5.1: Accuracy value is calculated after the *IES_deletion* function. After deleting IES specific sequences from *new_mic*, the function generates *dev_mac1*. Then, a pairwise sequence alignment is performed between *dev_mac1* and *old_mac* for finding the similarity between these two sequences. In this example, after performing the alignment it gives the score of 5, and the size of *old_mac* is 46. Then, the accuracy is (5 * 100) / 46 = 11.

after the *rearrangement* function has been applied. This allows to see which value of the three parameters maximizes the accuracy. The scores are added, as this will end up containing exactly the same information as taking the average since 13 values are always added. Also, the accuracy after the *rearrangement* function is used because this function most accurately represents the success of the simulation. Indeed, using the accuracy after the *IES_deletion* function always gives low accuracies in the case of scrambled genes. And taking the accuracies after the *correction* function often can fix otherwise bad alignments. Furthermore, it seems that in the *IES_deletion* function, use of cryptic pointers to incorrectly descramble genes occurs but the cryptic pointers are relatively close to the real pointers. Thus, using the accuracy after the rearrangement function seems to be the best way to calculate the optimal parameters.

The maximum summation of accuracy values, over all possible values of the three parameters is 906, which occurs when the parameters of *threshold_MDS* is 5, *threshold_IES* is 9, and *nh* is 15. Indeed, these values are considered as the optimal parameters of the simulation. Table 5.1 shows *acc1*, *acc2*, and *acc3* of all 13 input pairs for the optimal parameters.

Table 5.2 shows the average accuracies for each value of *threshold_MDS* (from 1 to 28) where *threshold_IES* and neighbourhood (*nh*) are fixed with their optimal values. Here, average accuracy is calculated by adding accuracies after the *rearrangement* function of all 13 input pairs with the selected set of parameters, and then dividing the summation of the accuracies by 13. For example, 876 is found as the summation of accuracies after the *rearrangement* function of all 13 input pairs where *threshold_MDS* is 10, *threshold_IES* is 9, and *nh* is 15. Then, the average accuracy is (876/13) = 67. In the same way, the average accuracies are calculated for Table 5.3 and Table 5.4. Table 5.3 shows the average accuracies for different values of *threshold_IES* (from 1 to 28) where *threshold_MDS* and neighbourhood (*nh*) are fixed with their optimal values. Also, Table

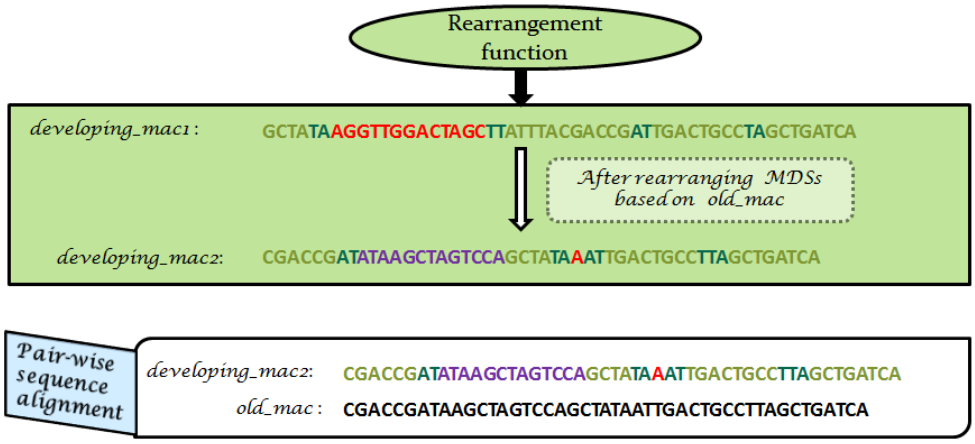


Figure 5.2: Here, accuracy is calculated after the *rearrangement* function. This function rearranges MDSs of *dev_mac1* on the basis of *old_mac* and produces *dev_mac2*. After that, a pairwise sequence alignment is performed between *dev_mac2* and *old_mac* for finding the similarity between these two sequences. After performing the alignment it gives the score of 40. As the size of *old_mac* is 46, the accuracy is (40 * 100) / 46 = 87.

5.4 shows the average accuracies for changing neighbourhood (*nh*) (from 1 to 20) where *threshold_MDS* and *threshold_IES* are fixed with their optimal values. The average accuracies are calculated to find a relationship between the accuracy and the parameters.

Table 5.5 shows the highest *acc2* value for each gene pair, for any possible values of the parameters (15,680 combinations). It also shows the values of the parameters (indexed by the second, third, and fourth column) for which this highest *acc2* is attained (indexed by the sixth column). Indeed, the values of the three parameters are calculated here separately for each of the 13 gene pairs (rather than calculating one global set of optimal parameters).

Table 5.1: For each gene pair (indexed by the first column), *acc1*, *acc2*, and *acc3* are shown for the optimal parameters. Here, optimal values for *threshold_MDS* is 5, *threshold_IES* is 9, and *nh* is 15.

pair_no	threshold_MDS	threshold_IES	nh	acc1	acc2	acc3
1	5	9	15	55.45	73.38	96.12
2	5	9	15	79.31	85.28	99.99
3	5	9	15	18.17	45.82	97.35
4	5	9	15	38.44	80.67	99.43
5	5	9	15	31.71	63.42	99.41
6	5	9	15	60.49	81.67	99.43
7	5	9	15	47.62	45.32	99.21
8	5	9	15	18.25	66.95	98.12
9	5	9	15	52.15	78.02	99.93
10	5	9	15	62.04	87.00	99.01
11	5	9	15	41.81	69.48	99.42
12	5	9	15	35.00	82.34	99.43
13	5	9	15	21.93	52.52	92.44

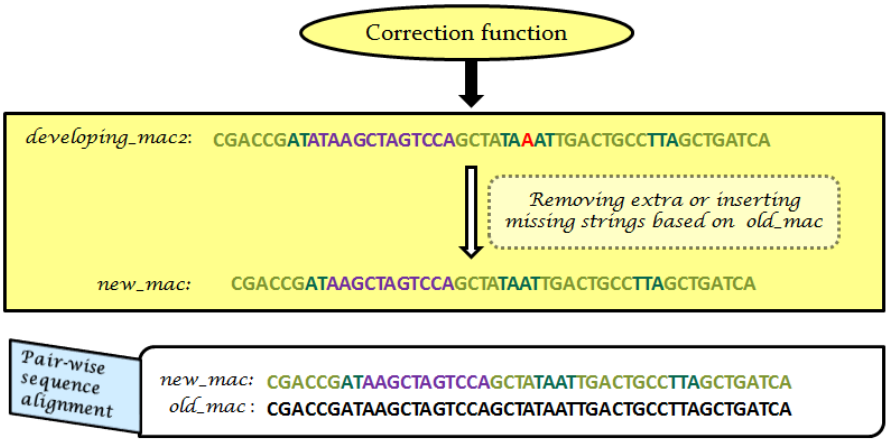


Figure 5.3: Here, the accuracy is calculated after the correction function. This function removes extra and inserts missing characters into *dev_mac2* by comparing it with *old_mac*. At the end of this function, *new_mac* is generated as the outcome. After that, a pairwise sequence alignment is performed between *new_mac* and *old_mac* for finding the similarity between these two sequences. In this example, after performing the alignment it gives the score of 46, and the size of *old_mac* is 46. Then, the accuracy is (46 * 100) / 46 = 100.

Table 5.2: The average accuracies of the 13 pairs by varying *threshold_MDS* (indexed by the first column) where *threshold_IES* and neighbourhood (*nh*) are fixed with their optimal values. Here, average accuracy is calculated by adding accuracies after the *rearrangement* function of all 13 input pairs with the selected set of parameters (indexed by the first three cells in each row), and then dividing the summation of the accuracies by 13.

threshold_MDS	threshold_IES	nh	average accuracy
1	9	15	66.77
2	9	15	67.85
3	9	15	66.85
4	9	15	65.77
5	9	15	69.69
6	9	15	67.77
7	9	15	65.00
8	9	15	64.08
9	9	15	65.46
10	9	15	67.38
11	9	15	58.46
12	9	15	50.23
13	9	15	46.46
14	9	15	46.31
15	9	15	46.08
16	9	15	42.69
17	9	15	38.31
18	9	15	38.23
19	9	15	34.62
20	9	15	33.85
21	9	15	32.00
22	9	15	31.23
23	9	15	29.62
24	9	15	24.85
25	9	15	20.69
26	9	15	17.00
27	9	15	10.38
28	9	15	10.54

Table 5.3: The average accuracies of the 13 pairs by varying *threshold_IES* (indexed by the second column) where *threshold_MDS* and neighbourhood (*nh*) are fixed with their optimal values. Here, the average accuracy is calculated by adding accuracies after the *rearrangement* function of all 13 input pairs with the selected set of parameters (indexed by the first three cells in each row), and then dividing the summation of the accuracies by 13.

threshold_MDS	threshold_IES	nh	average accuracy
5	1	15	67.77
5	2	15	66.54
5	3	15	66.69
5	4	15	67.46
5	5	15	66.00
5	6	15	67.77
5	7	15	67.46
5	8	15	67.46
5	9	15	69.69
5	10	15	67.38
5	11	15	66.85
5	12	15	65.92
5	13	15	67.92
5	14	15	66.85
5	15	15	67.46
5	16	15	66.77
5	17	15	67.46
5	18	15	66.31
5	19	15	67.08
5	20	15	67.54
5	21	15	66.00
5	22	15	67.08
5	23	15	67.15
5	24	15	66.46
5	25	15	66.38
5	26	15	65.85
5	27	15	68.15
5	28	15	65.69

Table 5.4: The average accuracies of the 13 pairs by varying *nh* (indexed by the third column) where *threshold_MDS* (indexed by the first column) and *threshold_IES* (indexed by the second column) are fixed with their optimal values. After the *rearrangement* function, the average accuracy is calculated by adding accuracies of all 13 input pairs with the selected set of parameters (indexed by the first three cells in each row), and then dividing the summation of the accuracies by 13.

threshold_MDS	threshold_IES	nh	average accuracy
5	9	1	67.77
5	9	2	68.46
5	9	3	65.69
5	9	4	67.00
5	9	5	65.85
5	9	6	65.38
5	9	7	68.46
5	9	8	66.69
5	9	9	66.00
5	9	10	66.92
5	9	11	67.54
5	9	12	66.54
5	9	13	65.38
5	9	14	66.15
5	9	15	69.69
5	9	16	67.31
5	9	17	66.15
5	9	18	66.23
5	9	19	66.85
5	9	20	66.69

Table 5.5: This table shows the highest value of *acc2* (indexed by the sixth column) possible for each gene pair, for any possible values of their parameters (15,680 combinations). It also shows the values of the parameters (indexed by the second, third, and fourth column) for which this highest *acc2* is attained (the values of the parameters are calculated separately for each of the 13 gene pairs rather than the global optimal values of the parameters).

pair_no	threshold_MDS	threshold_IES	nh	acc1	acc2	acc3
1	16	2	2	54.29	85.71	98.02
2	2	8	6	79.31	89.33	99.35
3	9	7	9	18.03	53.21	97.29
4	8	8	16	34.21	85.00	99.55
5	12	17	2	27.35	73.16	99.31
6	12	14	19	59.62	89.03	99.10
7	2	17	3	47.31	50.24	99.91
8	2	17	11	18.71	78.62	99.28
9	11	15	4	56.00	86.05	99.33
10	14	11	3	55.22	93.21	99.23
11	6	16	18	41.18	75.78	99.03
12	20	18	2	39.93	89.37	99.44
13	9	9	16	21.20	70.65	97.36

5.4 Other findings

Some additional findings from the simulation are described in the subsections that follow.

5.4.1 Improvement of accuracy in each stage

From the 2JLP model (Section 2.1.3), it can be seen that the macronucleus is generated from the micronucleus in a successive manner. Table 5.1 shows that *acc2* is greater than *acc1* and *acc3* is greater than *acc2* for all input pairs except for pair No 7 where *acc1* is greater than *acc2*. Indeed, the Computational 2JLP model generates the macronucleus from the micronucleus in a successive manner. The exceptional accuracy of pair No 7 is the pair with an incomplete micronuclear sequence. Biologically, the micronuclear sequence should be longer than the macronuclear sequence, but pair No 7 has a smaller micronuclear sequence than its macronuclear sequence.

Figure 5.4 shows a bar graph generated from Table 5.1 where the parameters are set with the optimal values. As the optimal parameters are able to generate good accuracy, one could use these optimal values as the default parameters of the simulation.

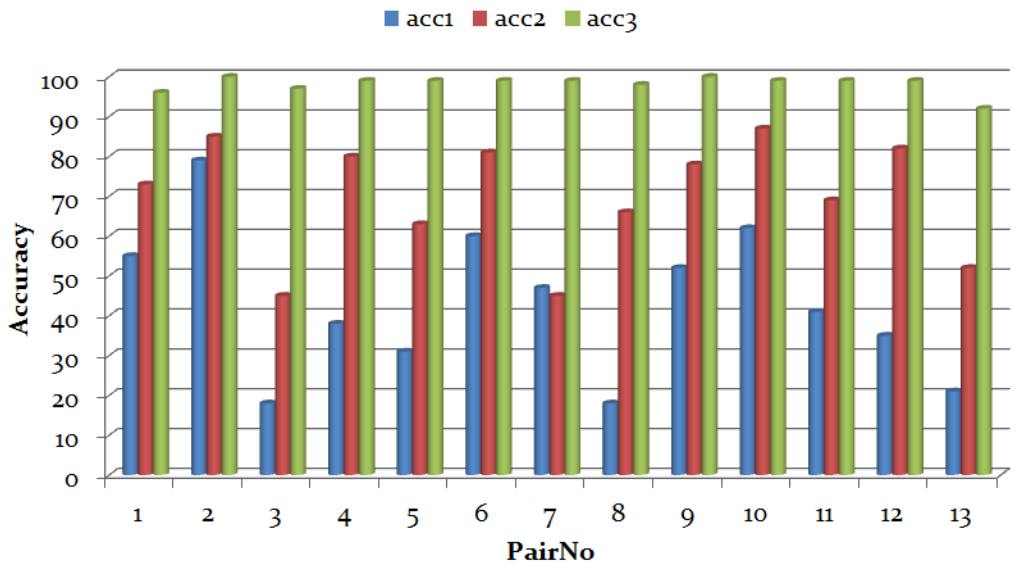


Figure 5.4: For all 13 input pairs, *acc1*, *acc2*, and *acc3* are shown in a bar graph where the optimal values are selected for the parameters. Here, optimal values for *threshold_MDS* is 5, *threshold_IES* is 9, and *nh* is 15. Also, *acc1*, *acc2*, and *acc3* represent consecutively accuracy values after the *IES_deletion* function, after the *rearrangement* function, and after the *correction* function.

5.4.2 Relationship between accuracy and *threshold_MDS*

In Section 4.2.2, the significance of the parameter *threshold_MDS* was discussed. In Figure 5.5, a scatter plot is shown which shows the relationship between *threshold_MDS* and average accuracy. These average accuracies are selected from Table 5.2. The scatter plot is generated by plotting *threshold_MDSs* in the x-axis and average accuracies in the y-axis. From the figure it is visible that the trend-line equation as $y = -2.279x + 78.695$ and the square of the correlation coefficient (R^2) value as 0.9481. In the trend-line equation, the slope value is negative which means that there is a negative correlation present in between these two variables. If the value of *threshold_MDS* is increased it eventually degrades the value of accuracy. As the R^2 value is 0.9481, that indicates approximately 95% of the variation in accuracy can be explained by *threshold_MDS*.

Figure 5.5 shows that a lower value of *threshold_MDS* is good from the perspective of maximizing the accuracy for the simulation. These lower scores for *threshold_MDS* occur when shorter pieces of scnRNAs are matched to the old macronucleus at the time of filtering, similar to IES specific sequences from the set of scnRNAs. Thus, if a scnRNA contains part of an MDS and part of an IES, from the perspective of maximizing the accuracy of the simulation, it is desirable to filter out this scnRNA. This is because if it does not get filtered out then the simulation may discard the matching portion of that scnRNA from the new micronucleus. This may result in an erroneous deletion of an MDS from the micronucleus.

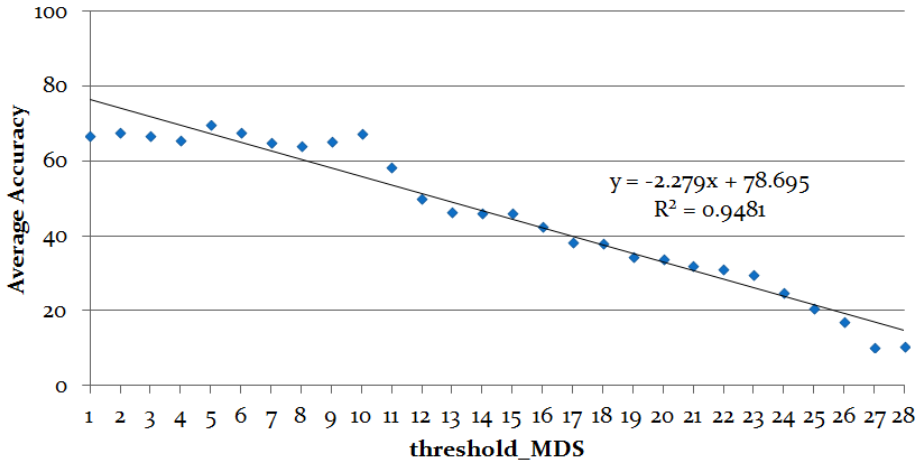


Figure 5.5: Relationship between *threshold_MDS* and average accuracy (after the *rearrangement* function for all 13 input pairs). Here, *threshold_IES* and *nh* are fixed with the optimal values. This figure is generated by using Table 5.2.

5.4.3 Relationship between accuracy and *threshold_IES*

Figure 5.6 shows a scatter plot where *threshold_IESs* are plotted in the x-axis and average accuracies are plotted in the y-axis. These average accuracies are selected from Table 5.3. The figure shows the trend-line equation as $y = -0.03x + 67.476$ and the square of the correlation coefficient (R^2) value as 0.083. In the trend-line equation, the slope value is near to zero which indicates that there is no positive or negative correlation between *threshold_IES* and average accuracy. Moreover, R^2 value is 0.083, that means approximately 92% of the variation in average accuracy cannot be explained by *threshold_IES*.

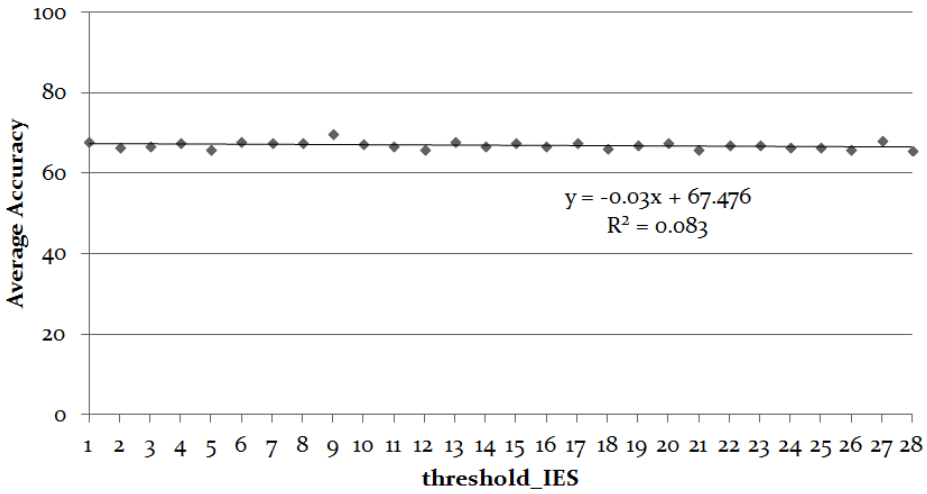


Figure 5.6: Relationship between *threshold_IES* and average accuracy (after the *rearrangement* function for all 13 input pairs). Here, *threshold_MDS* and *nh* are fixed with the optimal values. This figure is generated by using Table 5.3.

5.4.4 Relationship between accuracy and neighbourhood

Figure 5.7 shows a scatter plot where neighbourhood (nh) values are plotted in the x-axis and average accuracies are plotted in the y-axis. These average accuracies are selected from Table 5.4. The figure shows the trend-line equation as $y = -0.0124x + 66.968$ and the square of the correlation coefficient (R^2) value as 0.0043. In the trend-line equation, the slope value is near to zero which indicates that there is no positive or negative correlation between neighbourhood and accuracy. Moreover, R^2 value is 0.0043, that means approximately 99.6% of the variation in average accuracy cannot be explained by neighbourhood.

Interestingly, a relationship among *threshold_MDS*, neighbourhood, and accuracy after the *rearrangement* function is observed. If the value of *threshold_MDS* is greater than 20 then the simulation gives good accuracy after the *rearrangement* function only if the value of *nh* is very small. Table 5.6 shows an example of this situation. It also indicates that when more scnRNAs are considered as similar to IES specific sequences with a high *threshold_MDS* value, then at the time of finding repeats if the size of neighbourhood is increased

it may select a cryptic pointer instead of real pointer and cause erroneous deletion of the MDSs from the micronucleus. This can generate poor accuracy.

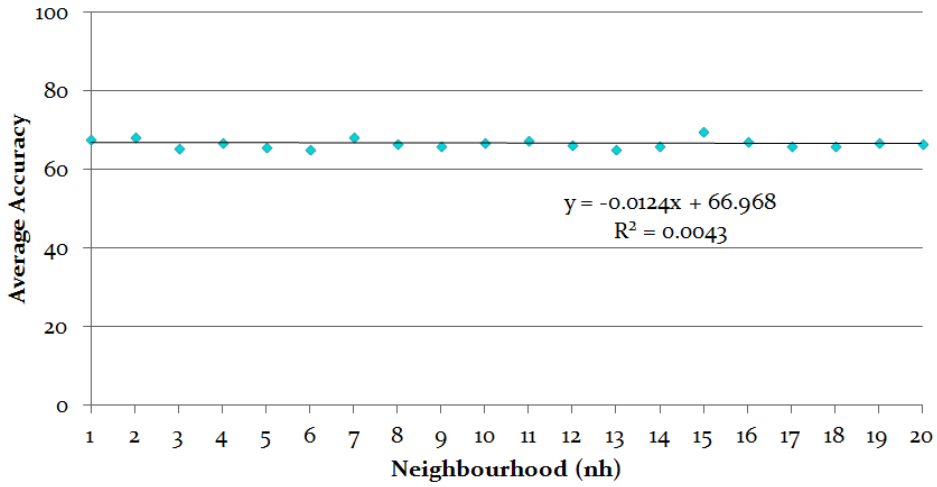


Figure 5.7: Relationship between neighbourhood (*nh*) and average accuracy (after the *rearrangement* function for all 13 input pairs). Here, *threshold_MDS* and *threshold_IES* are fixed with the optimal values. This figure is generated by using Table 5.4.

Table 5.6: An example for gene pair number 1, highlighting the relationship between neighbourhood (nh) and accuracy, when there is a high value of *threshold_MDS*. The fourth and fifth columns show that better values of *acc2* are found when the value of nh is very small. It also indicates that when more scnRNAs are considered as similar to IES specific sequences for the high *threshold_MDS* value, then at the time of finding repeats, if the size of neighbourhood (nh) is increased it may select a cryptic pointer instead of a real pointer and cause the erroneous deletion of the MDSs from the micronucleus. This can generate poor accuracy.

threshold_MDS	threshold_IES	nh	acc1	acc2	acc3
26	12	1	55.25	64.48	99.91
26	12	2	39.03	67.59	97.34
26	12	3	20.28	24.32	99.13
26	12	4	13.33	11.11	98.36
26	12	5	10.00	12.91	97.12
26	12	6	11.24	13.33	99.21
26	12	7	11.61	14.21	99.94
26	12	8	11.52	13.33	99.23
26	12	9	11.33	13.03	99.01
26	12	10	11.24	13.05	99.05
26	12	11	11.13	10.22	99.91
26	12	12	11.13	12.56	99.14
26	12	13	11.04	11.73	99.28
26	12	14	11.26	16.16	99.09
26	12	15	11.05	12.22	99.92
26	12	16	12.22	13.53	99.99
26	12	17	12.32	16.01	99.95
26	12	18	12.21	16.28	99.93
26	12	19	13.04	13.39	99.92
26	12	20	13.13	22.00	99.02

Chapter 6

Conclusions, Discussions and Future Work

This chapter describes conclusions, discussions and future work of this research work.

6.1 Conclusions

Both computer scientists and biologists are trying to understand the gene assembly process of ciliates. The scnRNA model partially revealed the mechanism of deleting IESs from the new micronucleus. The template guided model helps to know the order of MDSs in the new macronucleus with the IESs. Moreover, templates provide the necessary information for proof reading at the final stage. The template guided recombination model helps pointer identification and the method of MDS rearrangement at the developing stage of the new macronucleus. All of these models seem to be correct for some certain part of the gene assembly process. As these models alone are not able to completely describe the gene assembly process, the 2JLP model was created [10] to combine both the scnRNA and the template guided models.

This research describes a new formal model called the *Computational 2JLP* model which is based on the 2JLP model. A simulator is developed to check the feasibility of the computational model. The simulator has been run by using real data. In the simulator, three parameters (*threshold_MDS*, *threshold_IES*, and neighbourhood) are used to test edge cases of the computational model. The parameter *threshold_MDS* represents the minimum score of the semi-global alignment needed for sufficient similarity between scnRNAs and the old macronucleus. The parameter *threshold_IES* represents the minimum score of the semi-global alignment needed for sufficient similarity between filtered scnRNAs and the new micronucleus. The parameter neighbourhood (nh) represents the range of possible cryptic or real pointers in proximity to the marked IES specific sequences in the new micronucleus (which is likely close to the MDS-IES junction). There are some MDSs and IESs that are less than 28 bp long, and also there are scnRNAs that contain both a part of an MDS and part of an IES. In both cases, it is interesting to see the simulation results as the parameters vary. Indeed, *threshold_MDS* and *threshold_IES* are the parameters which gives that information.

In an analysis of the simulation results, three consecutive stages have been considered for calculating the accuracy between the developing macronucleus and the old macronucleus. These stages are after IESs deletion, after the rearrangement of MDSs, and after the proof reading of the developing macronucleus. The results of the simulator have showed that for almost all cases higher accuracy is found from one stage to

another (with the exception of one gene pair that had incomplete data). More interestingly, for all cases after proof reading, an accuracy between 92 and 100 is determined. This high accuracy may indicate that the gene assembly process is correct. Indeed, it is expected to not reach 100 due to regular sequence variation. From the result analysis, the optimal values for *threshold_MDS*, *threshold_IES*, and *nh* are determined to be 5, 9 and 15 respectively. A negative correlation is shown between the value of *threshold_MDS* and accuracy after the rearrangement of MDSs. That means that if a scnRNA consists of an MDS and IES, then from the perspective of maximizing the accuracy of the simulation, it is desirable to filter out this scnRNA. Indeed, if it does not get filtered out then the simulation may discard the matching portion of that scnRNA from the new micronucleus by considering it as similar to an IES, which may cause an erroneous deletion of MDSs from the micronucleus. This ultimately generates poor accuracy after the *IES_deletion* function. An interesting relationship among the variables *threshold_MDS*, *nh*, and accuracy is found. If the value of *threshold_MDS* is increased then the simulation sends more scnRNAs to the macronucleus development stage by considering them as similar to the IES specific sequences. After the marking of IES specific sequences in the micronucleus based on those scnRNAs, if a large value is set for *nh* then the possibility of making erroneous deletions is higher. This is because some of the marked sequences may contain MDSs which may influence the selection of a cryptic pointer instead of a real pointer.

6.2 Discussions

This section describes the importance of the Computational 2JLP model by considering both the biological and computational perspectives.

6.2.1 Biological importance

In the new micronucleus, MDSs can occur in the correct order or in some scrambled order. After gene assembly, ciliates produce a macronucleus which has the MDSs in the correct order. In the simulation, when a new micronuclear gene is taken with scrambled MDSs and both the *IES_deletion* and *rearrangement* functions are performed over the gene then the simulation unscrambles almost all MDSs.

It is known that the gene assembly process could lead to new combinations of MDSs by having a mistake during the process of cutting, splicing, excision, and unscrambling [10]. Especially, as a consequence of imprecise IES elimination at the borders of neighbouring MDSs and correction by removing extra or inserting missing information based on the template, this may lead to new genetic combinations of chromosomes in the macronucleus. From the simulation output, this phenomenon is often observed. Also, the variation of the parameters give preferred values (in particular, low values of *threshold_MDS*) indicating what is more likely to occur especially when scnRNAs contain part of an IES and part of an MDS.

6.2.2 Computational importance

Gene assembly is done by performing extensive IES elimination and MDS rearrangement. Research shows that a cell has to process its micronucleus into about 2×10^8 chromosomes in only a few hours implying that there are over one million DNA rearrangements per second at the time of the gene assembly [10]. This represents a large-scale method of computation in nature with a huge amount of parallelism. Indeed, the alignment and determination of relevant MDSs, IESs and pointers between a micronuclear and macronuclear gene pair is a known NP-Complete problem [11]. As ciliates "solve" (or at least are able to successfully descramble) this problem easily then it is an interesting use of "natural computation" for computer science. Thus, potentially other NP-Complete problems could benefit by applying the techniques of the gene assembly process in ciliates as a heuristic.

In Section 6.1, it is mentioned that the simulation is able to perform the gene assembly process in ciliates quite well. Indeed, all steps of the biological 2JLP model are used by the newly designed Computational 2JLP model. This is a helpful step in understanding the nature of the gene assembly process more systematically.

6.3 Future Work

Some potential future extensions of this work are listed as follows:

- In the simulator, many parameters have been used in an attempt to determine new biological facts. When selecting values for these parameters, due to a lack of evidence, values for some parameters like *threshold_MDS*, *threshold_IES*, neighbourhood (nh), *window_size*, *diff*, etc are assumed. Among these assumed parameters only *threshold_MDS*, *threshold_IES*, and neighbourhood (nh) have been systematically varied in the simulator. An important future work is to analyse all of these assumed parameters and find new biological facts.

- The simulator has been tested only for thirteen pairs of real genes. After the availability of more micronuclear data, more conclusive analysis will be possible.

- One of the important future works is to perform a detailed complexity analysis of the simulation. The simulation was run on an Intel Core i5-2430M with 6GB RAM running Windows 7 Home Premium 64-bit. The execution time for running the simulation once for every input pairs by using optimal parameters is 29.97 minutes.

- Currently, no online simulator is available to perform the gene assembly process in ciliates. One of the future works is to design an online simulator where biologists can easily perform the gene assembly process by giving inputs and selecting parameters.

References

[1] Dieter Ammermann, Günther Steinbrück, Ludwig Berger, and Wolfgang Hennig. The development of the macronucleus in the ciliated protozoan *Stylonychia mytilus*. *Chromosoma*, 45(4):401–429, 1974.

[2] Angela Angeleska, Nataŝa Jonoska, Masahico Saito, and Laura F. Landweber. RNA-guided DNA assembly. *Journal of Theoretical Biology*, 248(4):706–720, 2007.

[3] John R. Bracht, Wenwen Fang, Aaron David Goldman, Egor Dolzhenko, Elizabeth M. Stein, and Laura F. Landweber. Genomes on the edge: Programmed genome instability in ciliates. *Cell*, 152(3):406–416, 2013.

[4] Michael Brudno, Sanket Malde, Alexander Poliakov, Chuong B. Do, Olivier Couronne, Inna Dubchak, and Serafim Batzoglou. Glocal alignment: finding rearrangements during alignment. *Bioinformatics*, 19(suppl 1):i54–i62, 2003.

[5] Andre Cavalcanti. Ciliate nuclear dimorphism pages. http://oxytricha.princeton.edu/dimorphism/, July 2004.

[6] Mark Daley and Lila Kari. Some properties of ciliate bio-operations. In *Proceedings of the 6th international conference on Developments in language theory*, DLT'02, pages 116–127, Berlin, Heidelberg, 2003. Springer-Verlag.

[7] Mark Daley and Ian McQuillan. Template-guided DNA recombination. *Theoretical Computer Science*, 330(2):237–250, feb 2005.

[8] Andrzej Ehrenfeucht, Tero Harju, Ion Petre, David M. Prescott, and Grzegorz Rozenberg. *Computation in Living Cells – Gene Assembly in Ciliates*. Springer Verlag, 2004.

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

[10] Franziska Jönsson, Jan Postberg, and Hans J Lipps. The unusual way to make a genetically active nucleus. *DNA Cell Biol.*, 28(2):71–8, 2009.

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

[12] Laura F. Landweber and Lila Kari. The evolution of cellular computing: natures solution to a computational problem. *Biosystems*, 52:3–13, 1999.

[13] Laura F. Landweber, T. C. Kuo, and E. A. Curtis. Evolution and assembly of an extremely scrambled gene. *Proc Natl Acad Sci USA*, 97(7):3298–3303, mar 2000.

[14] Kazufumi Mochizuki. DNA rearrangements directed by non-coding RNAs in ciliates. *Wiley Interdisciplinary Reviews: RNA*, 1(3):376–387, 2010.

[15] Kazufumi Mochizuki, Noah A. Fine, Toshitaka Fujisawa, and Martin A. Gorovsky. Analysis of a piwi-related gene implicates small RNAs in genome rearrangement in Tetrahymena. *Cell*, 110(6):689–699, 2002.

[16] Matthias Möllenbeck, Yi Zhou, Andre R. O. Cavalcanti, Franziska Jönsson, Brian P. Higgins, Wei-Jen Chang, Stefan Juranek, Thomas G. Doak, Grzegorz Rozenberg, Hans J. Lipps, and Laura F. Landweber. The pathway to detangle a scrambled gene. *PLoS ONE*, 3(6):e2330, 06 2008.

[17] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, 1970.

[18] Mariusz Nowacki, Vikram Vijayan, Yi Zhou, Klaas Schotanus, Thomas G. Doak, and Laura F. Landweber. RNA-mediated epigenetic programming of a genome-rearrangement pathway. *Nature*, 451:153–158, 01 2008.

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

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

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

[22] David M. Prescott and M.L. DuBois. Internal eliminated segments (IESs) of Oxytrichidae. *J. Euk. Microbiol.*, 43(6):432–441, 1996.

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

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

[25] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 03 1981.

[26] Estienne C. Swart, John R. Bracht, Vincent Magrini, Patrick Minx, Xiao Chen, Yi Zhou, Jaspreet S. Khurana, Aaron D. Goldman, Mariusz Nowacki, Klaas Schotanus, Seolkyoung Jung, Robert S. Fulton, Amy Ly, Sean McGrath, Kevin Haub, Jessica L. Wiggins, Donna Storton, John C. Matese, Lance Parsons, Wei-Jen Chang, Michael S. Bowen, Nicholas A. Stover, Thomas A. Jones, Sean R. Eddy, Glenn A. Herrick, Thomas G. Doak, Richard K. Wilson, Elaine R. Mardis, and Laura F. Landweber. The *Oxytricha trifallax* macronuclear genome: A complex eukaryotic genome with 16,000 tiny chromosomes. *PLoS Biol*, 11(1):e1001473, 01 2013.

[27] Sergey Verlan, Artiom Alhazov, and Ion Petre. A sequence-based analysis of the pointer distribution of stichotrichous ciliates. *Biosystems*, 101(2):109–116, 2010.