Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Decyphering the DNA replication dynamics of *Trypanosoma cruzi* through computational modeling

Victor Seiji Hariki

Capstone Project Report

mac 499 — Trabalho de Formatura Supervisionado

Advisor: Dr. Marcelo da Silva Reis

This work was supported by a scholarship from CNPq

São Paulo
January 30, 2021

Abstract

Trypanosomatids are endoparasitic protozoan whose genes are organized into polycistrons and have constitutive transcription along the whole cell cycle. Those facts suggest that conflicts between DNA replication and transcription machineries yield increased replication origin firing in the cell cycle S phase. To investigate that hypothesis, our group developed a DNA replication model for *Trypanosoma brucei* that was calibrated with MFA-seq data and used it to predict and validate that increasing constitutive transcription levels indeed increase the number of fired origins. More recently, it was concluded the DNA origins mapping for *T. cruzi* through MFA-seq assays, which unveiled that several origins are located in coding regions of the dispersed gene family 1 (DFG-1), a family of genes that is relevant for the parasite life cycle. Once DFG-1 genes have high genetic variability, one possibility is that conflicts between DNA replication and transcription machineries are responsible for such variability, which implies that the origin firing distribution in *T. cruzi* is conditioned by the genomic organization of that parasite. In this work, we proposed to investigate such question through computational modeling of the DNA replication dynamics of *T. cruzi*. We adapted for *T. cruzi* the dynamic model of DNA replication programming originally developed for *T. brucei*, in this process promoting several improvements on the original simulator, for instance, a more suitable data compressor. With the improved simulator, we calibrated a *T. cruzi* model with MFA-seq data and carried out a number of computational experiments, assessing the correlation between conflict regions and DFG-1 locations. Finally, we developed a novel, genetic algorithm-based simulator for studies of genomic organization evolution in *T. cruzi*. With all those accomplishments, we expect to be able to assess the interplay between replication-transcription conflicts and evolution of genomic organization, thus helping to understand one of the underlying mechanisms that guarantees the successful invasion and survival of the parasite in its host.

Contents

Chapter 1

Introduction

The family of trypanosomatids is composed by obligate endoparasitic protozoa, some with biomedical relevance, as with *Trypanosoma cruzi* and *brucei* (Etiological agents for the Chagas disease and sleeping sickness, respectively). As the sicknesses caused by this family of protozoa are considered by the World Health Organization as neglected diseases (Andrade *et al.*, 2014), the biology of these organisms is strongly studied aiming the discovery of molecular targets, seeking pharmacological interventions for prevention and treatment of infections. Those studies have revealed that trypanosomatids have some quite peculiar characteristics, such as having their genes organized into polycistrons (i.e., linear gene sequences that are transcribed into a single RNA molecule) and the presence of constitutive transcription (i.e., that occurs without transcription factor regulation in its promoter sequence) during the full cell cycle. Because of this, those organisms have a DNA replication dynamic that displays some very specific properties.

Eukaryotic DNA replication, such as in protozoa, is started from genomic sites named replication origins. On each of these sites, a pair of replication machineries named replisomes are attached and head to opposite directions. We call this phenomenon a firing of the replication origin. Transcription is done through the action of an the RNA polymerase enzyme (RNAP). The genomic organization in trypanosomatids leads to the ocurrence of conflicts between DNA replication and transcription machineries. There are two types of these conflicts: head-to-tail and head-to-head collisions (García-Muse and Aguilera, 2016). Head-to-tail collisions are easily resolved by derailing the RNAP molecule from the DNA strand. On the other hand, head-to-head collisions cannot be solved in a simple manner, and may cause a collapse of the replisome or even a break of the DNA strand.

Aiming to study the impact of the conflicts between DNA replication and transcription conflicts on the DNA replication dynamics of *T. brucei*, a stochastic computational model was developed to simulate the S-Phase processes of this organism (M. d. Silva *et al.*, 2019) (Figure 1.1). This model was based on another DNA replication dynamic model developed previously (Gindin *et al.*, 2014), and callibrated with experimental data available in the literature, such as the distribution of putative sites of replication origins that was acquired via MFA-seq assays (Tiengwe *et al.*, 2012). Other properties of the process, like the expected replisome speed and S-phase duration were also available (Calderano *et al.*, 2015; M. S. d. Silva *et al.*, 2017). Using a simulator called ReyDyMo, which originally was coded in

Python and latter ported to C++ for better computational performance (G. R. C. Silva, 2017; Scholl BB, 2018), simulations using this model with many different parameters showed that higher constitutive transcription levels result in higher conflict counts, that causes higher replication origin firing rates. This increase in firing rates does not, though, produces significant differences in the time required to replicate the entire genome of this parasite, a prediction that was experimentally confirmed (M. d. Silva *et al.*, 2019).

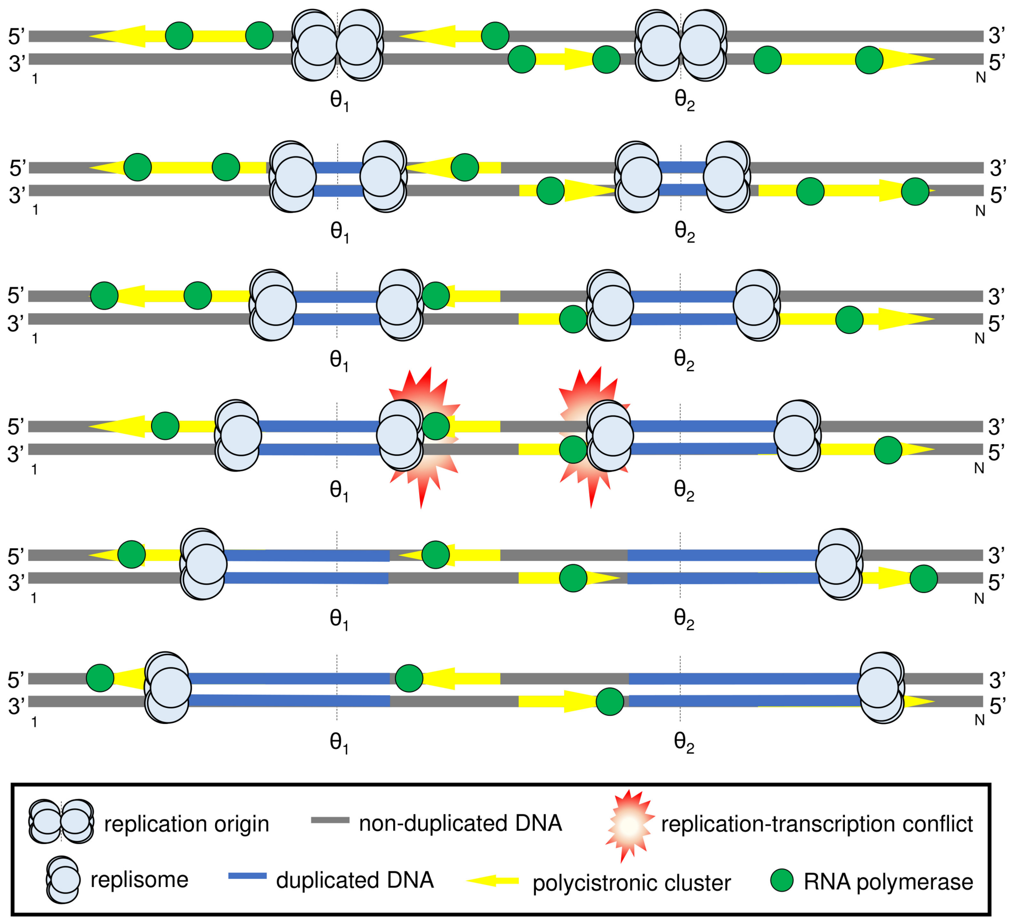

Figure 1.1: ***T. brucei* DNA replication dynamic model.** *In this example, we have a chromosome segment with two replication origins (θ_1 and θ_2). When we simulate DNA replication and transcription at the same time, if there is a head-to-head collision between DNA replication (replisome) and transcription (RNAP) machineries, then both are released from DNA, leaving an unfinished replication. In the case of this example, a third replication origin, located between θ_1 and θ_2, would be required to be fired in order to finish the interrupted replication. Image extracted from M. d. Silva et al., 2019, with the authorization of a corresponding author.*

More recently, in a study lead by Dr. Maria Carolina Elias (Cell Cycle Laboratory, Butantan Institute), MFA-seq assays were concluded for the *T. cruzi*. When comparing the chromosome DNA replication profile acquired by these assays with polycistron locations in this parasite, a large number of putative origins were found in coding regions for genes in the *dispersed gene family* (DGF-1) (Araujo *et al.*, 2020). DGF-1 genes are fundamental for the lifecycle of the *T. cruzi*, as they encode cell surface proteins that are important for successful infection and survival of the parasite in the host. Considering the high genetic variability of the DGF-1 genes, a possible cause of it is due to replication-transcription conflicts. This is an open question that could be investigated with computational models similar to the ones we successfully employed with the *T. brucei*.

1.1 Objectives

The main objective for this work was the study of the DNA replication mechanisms for the *Trypanosoma cruzi* using computational modeling and statistical analysis. To this end, we needed to adapt the stochastic model previously used for analysis regarding *T. brucei* and also to improve its implementation, the ReDyMo simulator.

A more specific objective was the assessment of the hypothesis that the genetic variability of the DGF-1 gene family is influenced by collisions between replication and transcription machineries, and that the genomic topology of the organism evolved to maximize such type of collisions. This goal was divided into two tasks:

- First, to carry out simulations for each input parameter set, using real experimental data from public biologic data repositories, and also to perform statistical analysis on the results.
- Second, to implement and run a population evolution simulation with random starting topologies and apply selective pressures to see which topologies eventually evolve from it.

1.2 Outline of this work

The remainder of this capstone project report is organized as follows:

- In Chapter 2, we will introduce fundamental biologic concepts important for the understanding of this work.
- In Chapter 3, we will discuss the model and its implementation, as well as provide data analysis methods that were used throughout this work.
- In Chapter 4, we will present and analyze the obtained results in a concise and descriptive manner.
- Finally, in Chapter 5, we will recall the main contributions of this work, and also add some extra thoughts about future activities in this research line.

Chapter 2

Fundamental Concepts

This work requires understanding of biological concepts that may not be clear for readers that are not familiar with biological sciences in general. In this chapter, we will present some of such concepts, and some of the nomenclature used in the remainder of this capstone project report. More details about most of the concepts covered here can be obtained in textbooks such as Voet *et al.*, 2008.

2.1 The Genome

The genome of a organism is the entirety of its genetic code. For the trypanosomatids covered here, the genome is composed of many chromosomes, which in turn are made of DNA (abbreviation of **d**eoxyribo**n**ucleic **a**cid). A DNA strand is composed of a sequence of nucleotides, which are what effectively encodes the actual data of the genome. These nucleotides may be A (adenine), C (cytosine), T (thymine) and G (guanine). All nucleotides are chemical components, with a common sugar ring called pentose and four different nitrogenous bases. Nucleotides are typically found in strand pairs, coiled as shown in Figure 2.1.

The nucleotides are only capable of correctly associating in one of the following pairs: Adenine and Thymine or Guanine and Cytosine, allowing us to describe the strand pair using only one of the strands. In this regard, there is a convention for encoding a DNA strand, which is from 3' to 5', which are carbon indices of a given pentose. 3' and 5' are the start and end extremities of a DNA strand, respectively, determined by the carbon atoms present on the pentose at each end of the strand.

A chromosome is a pair of complementing DNA strands, usually coiled around packaging proteins called histones, giving it the characteristic shape shown in Figure 2.2. trypanosomatids usually have many chromosomes, that in conjunction constitutes the genome of the organism. The set of all the chromosomes of the organism in question is called genome. The genome of *T. cruzi*, its chromosomes can be seen in Figure 2.3.

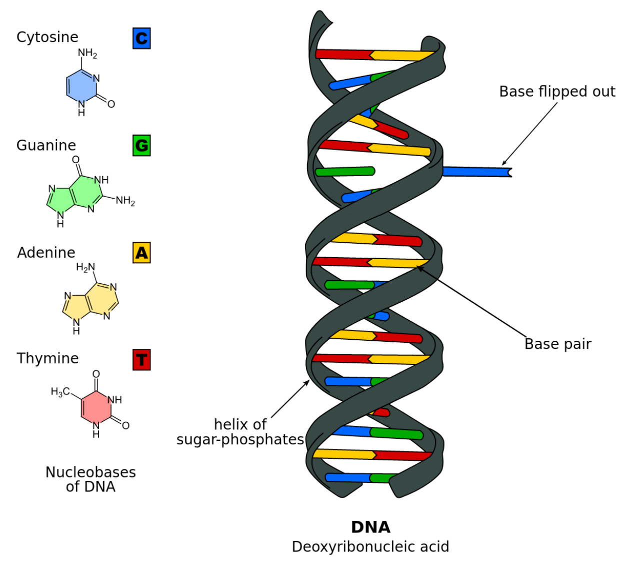

Figure 2.1: Pair of associated DNA strands. *Left: molecular composition of the four nucleotides. Right: an example of an DNA molecule. Figure extracted https://en.wikipedia.org/wiki/DNA_base_flipping*

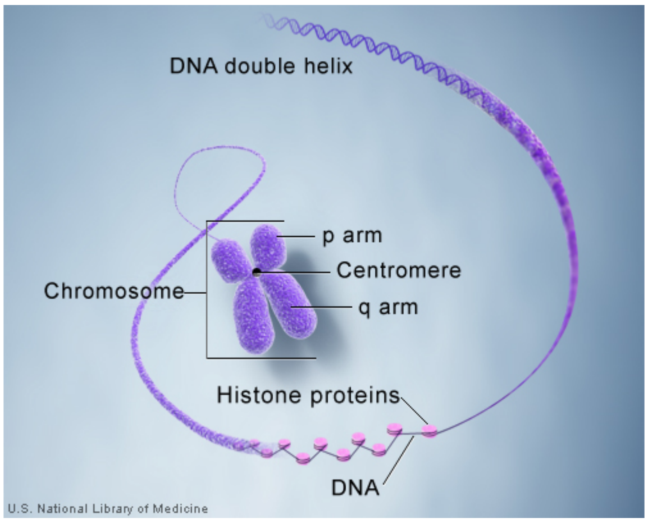

Figure 2.2: ***A chromosome anatomy.*** *The double-strand DNA is packed by histone proteins, and then further packaged in a condensed form that gives the characteristic 'X' shape.*

2.2 Replication and Transcription

Replication and transcription are two fundamental genetic processes that we will be simulating and analyzing here. We will now describe some basics of the workings of both processes.

2.2.1 DNA replication

We call DNA replication (or simply replication) the process of taking a DNA strand pair and cloning it. That is, starting with a double helix, we get two identical helices. On

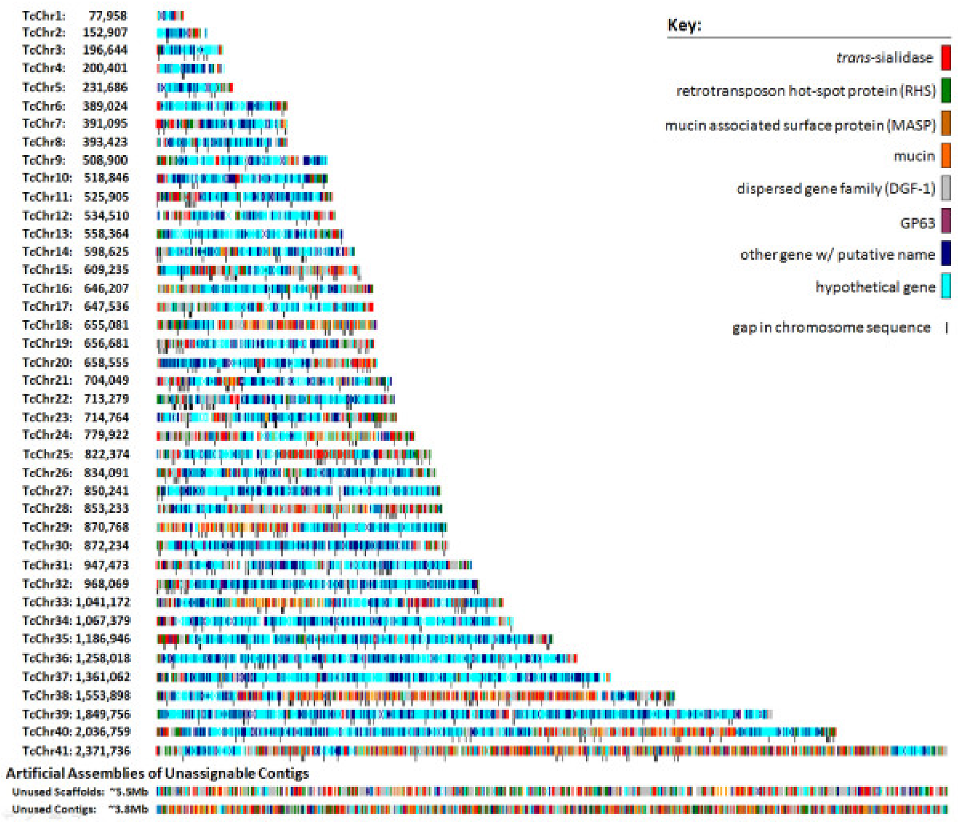

Figure 2.3: ***Trypanosoma cruzi* chromosomes, colored by decoded ranges.** *The chromosome sizes are shown on the left, along with the respective chromosome names.*

replication, the original strand is divided into two individual strands, each of which is used as a template for building its complementary strand, as seen in Figure 2.4. The machinery for this process is named replisome, and this is how it will be referred from here on. It is possible that errors may occur in replication, whose correction might not be perfect, thus causing mutations. This will be explored with more detail in Section 2.2.3.

2.2.2 Transcription

The DNA hosts sequences for many reasons, but one of its most important roles is to store the code for RNA sequences that perform essential functions in the cell. Transcription is the process of making a mRNA (short for messenger **R**ibo**N**ucleic **A**cid) sequence from a section of a double DNA strand. The RNAP (RNA polymerase - transcription machinery) attaches at the start of the gene to be transcribed and detaches at the end. The mRNA is then used for protein production at ribosomes (the cell translation machineries) or other functions (e.g., for the assembly of the ribosomes themselves).

2.2.3 Collisions

There are many possible sources of mutations, such as radiation destroying part of a DNA strand or replication machinery problems. But one of the most prevalent source of replication errors is collisions with other cellular machinery. The collisions that may

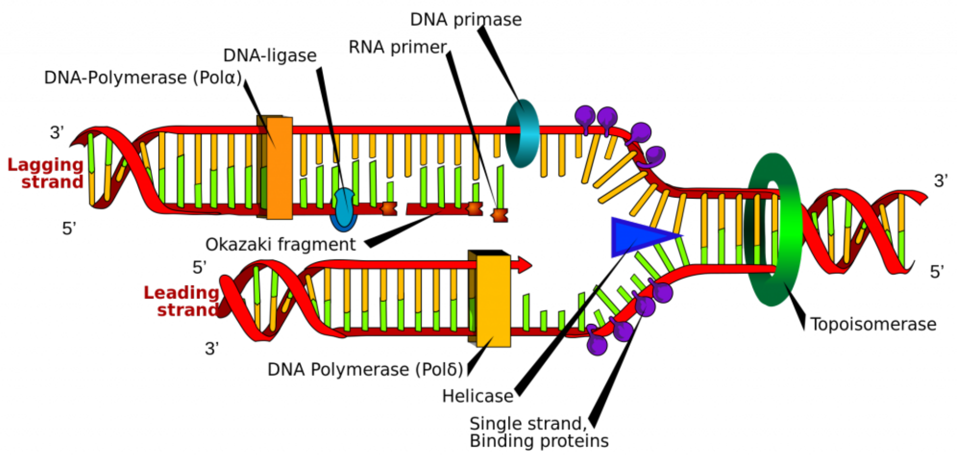

Figure 2.4: ***DNA replication process****. On the right is the original dual helix, that is divided, and each strand becomes a dual helix, presumably being an exact copy of the original.*

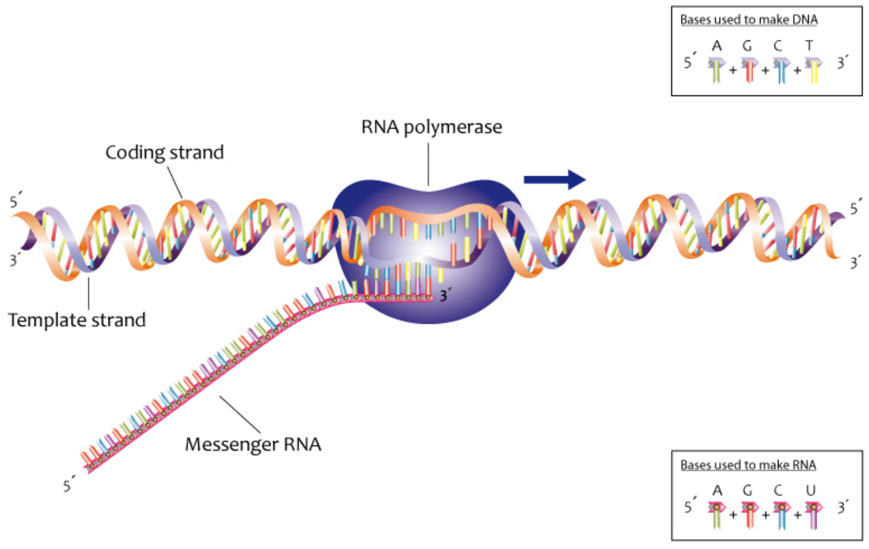

Figure 2.5: ***Transcription process****. As we can see, the RNA polymerase unwinds then rewinds the helix, using one of the strands as a template for the mRNA transcription.*

happen with replication and transcription machineries are:

- Replication-replication head-to-head collision: When replisomes approach from opposing sides of the chromosome. This type of collision is not a problem because replication machinery evolved in such a way that collisions of this type are handled gracefully.
- Replication-transcription head-to-tail collision, also called a co-directional collision: When a replisome approaches a RNAP from behind (replisomes are commonly faster than RNAPs). Worrisome but still handled quite well. Usually the RNAP is bumped

off (Figure 2.6, left).

- Replication-transcription head-to-head collision, also called a head-on collision: When a replisome and RNAP collide front to front. This can result in a fork collapse, which interrupt DNA replication in that location and might increase the probability of mutations (Figure 2.6, right).

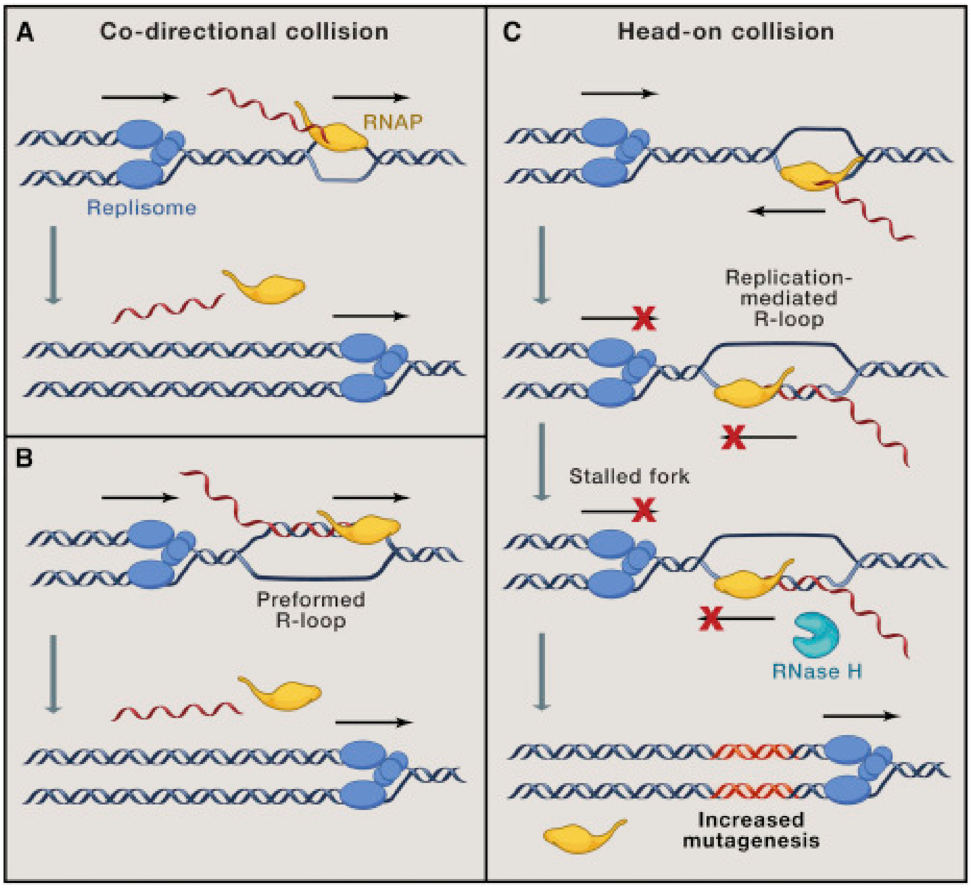

Figure 2.6: ***Types of replication-transcription collisions.*** *Figure extracted from* *Yea-Lih Lin, 2017.*

2.3 Replication origins

Replication actually can only start in specific origin locations on the DNA. These are called replication origins and come in three types:

- Constitutive origins: They are fired frequently and consistently in the start of the S phase, and more easily detectable.
- Flexible origins: Must be fired during the S phase, but firings occur randomly throughout the process.
- Dormant origins: Origins that are not fired unless something goes wrong during the replication.

2.4 The Cell Cycle

The cell cycle is a sequence of states a cell goes through before finally undergoing mitosis (duplicating). For eukaryotic organisms, such as the ones we are studying, the following are the four states it goes through:

1. G1 or Gap 1: The cell is active and growing, preparing required resources for eventual genome duplication
2. S or Synthesis: The cell duplicates its genome so this is the phase where the replisomes attach to the chromosomes.
3. G2 or Gap 2: The cell is active and growing, now with two copies of the genome. It prepares resources for eventual cell splitting.
4. M or Mitosis: The cell splits into two cells, each with a copy of the original genetic material.

The S Phase is the most important for us, as it is when the actual genetic duplication process occurs.

2.5 Evolution and Selective pressure

Selective pressure is a characteristic of the environment that has more probability of letting some organisms reproduce than others according to its traits. On the other hand, evolution is a process that happens when there are organisms with traits that reproduce non-perfectly in an environment with selective pressure. The population in this type of scenario slowly evolves, optimizing the traits for a phenotype with the best fitness in their environment.

Chapter 3

Methodology

In this chapter, we will present the main methodological activities that were accomplished to tackle the objectives proposed in this capstone project. We start describing, in Section 3.1, the optimization that was carried out in the C++ version of the ReDyMo simulator. In the sequence, in Section 3.2, we present some further improvements in that simulator that were performed through software patching. Next, in Section 3.3, we introduce a new evolution simulator, which enables us to study evolutionary aspects of genomic organization in *T. cruzi*. Finally, in Section 3.4, we describe the scripts that were coded for analyses of results generated by all those simulators.

3.1 Simulator Optimization

The speed we can run simulations at is one of the limiting factors for quality of the final results. Moreover, a large amount of data is generated by the ReDyMo simulator, which can be further explored using data mining and machine learning methods. However, a critical point in the ReDyMo computational performance is in the storage of such simulation data into the hard disk; therefore, we needed to develop a solution for such bottleneck.

3.1.1 Semantic Data Compressor

One of the main things that can be improved for a higher speed when running the simulations is in saving the data to the disk. Preliminary tests run for *Trypanosoma cruzi CL Brener Non-Esmeraldo-like* on a HDD, saving uncompressed data, show that more than 75% of the time was spent writing raw data to the disk.

While the newest version of the simulator had implemented Facebook's *zstandard* compression, allowing the time to be closer to 40% of total run time, the data was also rendered illegible to humans, and requires external libraries to decode. Fortunately, the data format is very predictable, and by utilizing this fact, it is possible to do better. Using this to our advantage, it is possible to write an algorithm that compresses the data more efficiently, while keeping it readable for humans and easy to parse. This section describes the implementation of a compressor that uses the data semantics to its advantage.

Properties of the compression algorithm

The compression algorithm we want should use the data semantics as a foothold and have the following properties:

1. **Must be efficient**: The algorithm should run in $\mathcal{O}(n)$, allowing for no significant loss of speed while using it.
2. **Must allow data streaming**: We are dealing with a large amount of raw output data. Together with the above item, it should allow data to be streamed in, not requiring the full data to be loaded into working memory.
3. **Must be easy to parse**: The format must not be difficult to decompress using any available language, as it is a custom algorithm. There should be clear delimiters, and a consistent formatting.
4. **Should be readable by humans**: The format should be readable by humans without decompression. This allows people to know how the data is expected to look like even while compressed

Raw data and compressed format

Now we will discuss about the format for the raw data that would have been written to the disk, and the representation of the data in compressed format.

To clarify some terminology, we define A streak as an arrangement of repeated numbers:

```
1 # Start of streak 1
1
1 # End of streak 1
2 # Start of streak 2
2
2
2 # End of streak 2
3 # Start of streak 3
3
3
3 # End of streak 3
```

Program 3.1: *Streak example.*

And a streak sequence is a range with several back to back streaks of same size, with values being incremented or decremented by one for each streak:

```
2 # Start of negative sequence
2
2
2
1
1
1
1 # End of negative sequence
2 # Start of positive sequence
2
```

```
2
3
3
3
4
4
4 # End of positive sequence
7 # Start of non-sequence streak
7
7 # End of non-sequence streak
5 # Start of negative sequence
5
5
4
4
4 # End of negative sequence
```

Program 3.2: *Sequence example.*

The uncompressed data saved to disk are mostly sequences of equally sized number streaks, except on replication machinery collisions and the ends of chromosomes. It is not uncommon for more than half of the data to be a single sequence of large streaks. (with streak length > 50).

As rebuilding a sequence from start value, end value and streak length is trivial in linear time and allows streaming, we can represent each sequence or lone streak in a generic and easy to read format:

`<sequence_start_value>[-<sequence_end_value>][x<streak_length>]`

A compressed file is to be comprised of lines in this format, with the sequence end value of the previous line different from the sequence start value of the current line. This restriction avoids confusing number streaks where half of its definition is in one line and the other half in in the next line. It also simplifies compression and removes ambiguity, as seen in the **next section**.

```
1275x23
1274-257x65
256
257-436x65
437x64
```

Program 3.3: *Example of compressed file.*

Above is an example of a real compressed file for chromosome TcChr1-S of the *Trypanosoma cruzi CL Brener Non-Esmeraldo-like*, with 77958 lines in the original data.

Compressor Implementation Details

The final algorithm is pretty simple in structure. There are many possible sequence break types. The last three number streaks are stored, in order to be able to find peaks and valleys. A compressed line is printed when a break is detected, for a previous sequence or single value:

```
# Not seq value detection example
s - 1 # Before previous (1, 2)
s - 1
s - 2 # Previous (2, 2)
s - 2
n - 4 # Current (4, 2)
n - 4
n - 5
n - 5

# Change size detection example
s - 1 # Before previous (1, 2)
s - 1
s - 2 # Previous (2, 2)
s - 2
n - 3 # Current (3, 3)
n - 3
n - 3
n - 4
n - 4

# Peak detection example
s - 1
s - 2 # Before previous
s - 3 # Previous > Before Previous
n - 1 # Current < Previous
n - 0

# Valley detection example
s - 1
s - 2 # Before previous
s - 3 # Previous > Before Previous
n - 1 # Current < Previous
n - 0

# Single value detection example
n - 1
n - 2 # Before previous
s - 4 # Previous > Before Previous
n - 1 # Current < Previous
n - 0
```

Program 3.4: *Example sequence breaks.* **s** *marks values included in the sequence that is currently being processed.*

Some details were changed from previous iterations of the algorithm, such as adding a requirement for each compressed line to end in a different value than the next line starts. This helped removing ambiguity, such as:

```
# Ambiguous case example - raw
1
1
2
2
2
3
```

```
3
4
4

# Possibility 1
1-2x2
2
3-4x2

# Possibility 2
1x2
2
2-4x2

# Possibility 3 - Now the only correct option
1x2
2x2
3-4x2
```

Program 3.5: *Example of possible ambiguity before change.*

It also helped simplifying the algorithm, as this fact made a technically correct, but very inefficient compression that required ugly workarounds to amend:

```
# Inefficient case example - raw
1
1
2
2
2
3
3
3
4
4
4
5
5
5

# Before
1-2x2
2-3
3-4x2
4-5
5x2

# Current - Now the correct option
1x2
2-5x3
```

Program 3.6: *Example of a technically correct but inefficient compression.*

Implementations

The compressor was implemented twice during this project. Once in the ReDyMo-CPP source code (https://github.com/msreis/ReDyMo-CPP), and in a python module, that may also be used as a command line script, available in the git repository (https://github.com/seijihariki/redymo-tcruzi-analysis).

3.1.2 GPU Processing

Another way of optimizing the speed of the simulator is running simulations on GPGPU. While the original code uses a separate thread for each simulated cell, the GPGPU simulation was implemented in OpenCL and made so to take advantage of graphic processor strengths. Although results may not be completely replicable by utilizing GPU processing, simple simulations can run faster that usual, accelerating sampling runs.

Implementation Details

The genome is stored contiguously, for faster and easier result retrieval using OpenCL array manipulation functions (Figure 3.1).

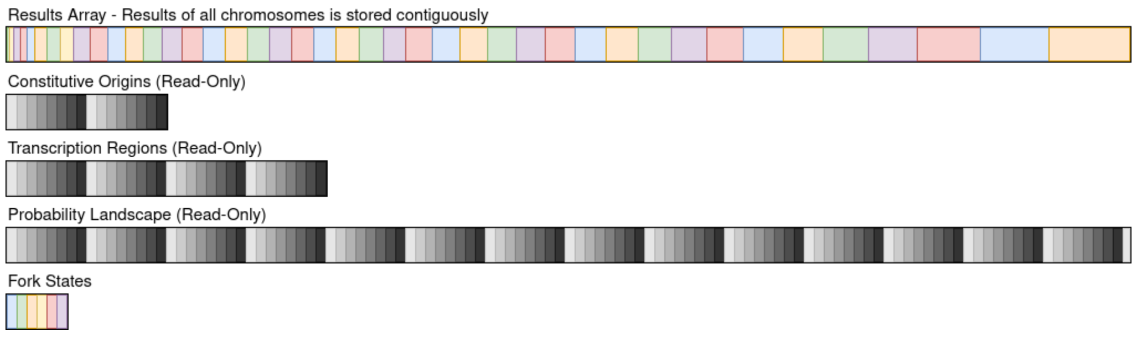
Results Array - Results of all chromosomes is stored contiguously
Constitutive Origins (Read-Only)
Transcription Regions (Read-Only)
Probability Landscape (Read-Only)
Fork States

Figure 3.1: *Data Arrays used in the simulation.*

Each fork is an individual thread, and the thread with ID 0 is responsible for managing fork. All threads are synchronized every 100 timesteps to avoid too large of a desynchronization. All relevant data is uploaded to the VRAM in the beginning of the simulation and retrieved in the end. In Figure 3.2, we provide the complete flowchat for GPGPU simulation.

3.2 Simulator Patching

As all software, the simulator had some shortcomings; some of them are small (e.g., lack of argument for output folder selection, which made scripting and automation difficult), whereas other are rather relevant (e.g., non-uniform probability for each base pair to be selected, that could possibly affect results in a significant way). In this section, we describe in details the functional changes made to the simulator to deal with those issues.

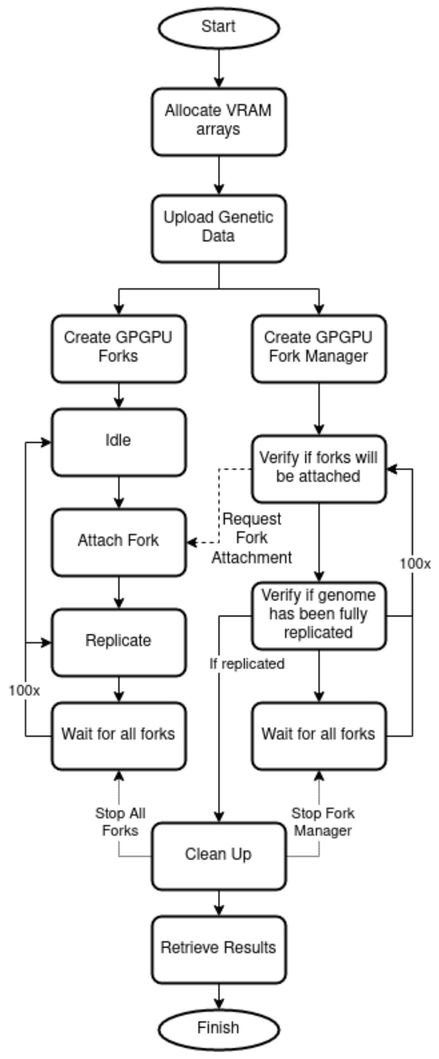

Figure 3.2: *Flowchart for GPGPU simulation.*

3.2.1 Migration to GNU getopt

A quite small, but relevant change is the migration to GNU getopt for command line argument parsing. Not only it is more stable, it is also more readable than the previous hands-on implementation. Many previously hard coded values have also been made into parameters, making the simulator more flexible, as well as a help command and menu. The added options are:

- -h/−help - Opens the help menu;
- -p/−probability - Sets an uniform probability for attaching replication machinery instead of loading MFA-Seq data;
- -O/−output - Sets output folder path, previously hardcoded to ./output/;
- -t/−threads - Allows to configure a number of concurrent threads, for running in different machines without recompilation;
- -g/−gpu - Enables GPGPU processing mode;

- -x/−seed - The seed to be used for the simulations. It will not guarantee consistent results for GPU simulations;
- -C/−config - Simulation configuration file to use for simulation.

All the options now have short equivalents, and are parsed via a dedicated Configuration class.

3.2.2 Use of Simulation Configuration File

With the added options for simulation, the length of the simulation command is getting way too long and unreadable. With the added possibility of simulating cell evolution, many more options would be needed. Therefore, we can share simulation configurations more easily and a growing number of parameters, a YAML configuration file reader was implemented. The configuration should be written in the following format:

```yaml
simulation: basic
parameters:
  cells: 50
  organism: TcruziCLBrenerEsmeraldo−like
  resources: 50
  speed: 65
  period: 100000
  timeout: 100000
  dormant: true
```

Program 3.7: *Configuration file template - Basic Simulation.*

```yaml
simulation: evolution
parameters:
  # Basic simulation data
  cells: 10
  organism: TcruziCLBrenerEsmeraldo-like
  resources: 50
  speed: 65
  period: 1000
  timeout: 100000
  dormant: true

  evolution: # Evolution simulator data
    population: 50
    generations: 50
    survivors: 30    # 30 out of 50 individuals will survive for the next
    generation
    mutations: # Allowed mutation types and their parameters
      probability_landscape:
        add: 0.15         # 15% chance of adding a new source
        del: 0.1          # 10% chance of deleting an existing source
        change_mean:
          prob: 0.05 # 5% chance of changing mean for each source
          std: 2000       # Standard deviation for the change
        change_std:
          prob: 0.05  # 5% chance of changing standard variation for each
    source
```

```
      std: 50         # Standard deviation for the change
      max: 1000       # Maximum standard deviation may ever be
  genes:
    move:
      prob: 0.02   # 2% chance of moving for each gene
      std: 5000    # Standard deviation of the change
    swap:
      prob: 0.005  # 0.5% chance of swapping with another gene
fitness: # Fitness calculation for evolution
  min_sphase: 3     # Optimizes for minimum s_phase duration with weight 3
  match_mfaseq: 5   # Optimizes for minimum difference with the original MFASEQ proabilities with weight 5
  max_coll:         # Optimizes for maximum collisions with gene with weight 2
    weight: 2
    gene: TcCLB.511557.29   # Will maximize collisions for this gene
```

Program 3.8: *Configuration file template - Evolution Simulation.*

3.2.3 Fixing Chromosome Probabilities

During development and data analysis, we noticed that when running simulations with uniform probability landscapes, random base probabilities were not uniform. This resulted from the process of selecting a random base. First, a random chromosome was selected, and then a random base in that chromosome was selected. That way, bases in shorter chromosomes had a higher probability of being selected than bases in longer chromosomes.

As an example, imagine we have three chromosomes named c_1, c_2, and c_3, with lengths of 1 base, 5 bases and 10 bases respectively. First, we choose a random chromosome, thus we have the following probabilities ($P(x)$ is the probability of a certain chromosome being chosen, and $Pb(x)$ is the probability of any specific base of that chromosome being chosen):

$$\begin{aligned}
P(x) &= 0.33, \\
Pb(c_1) &= \frac{1}{1} \cdot P(c_1) = 0.33, \\
Pb(c_2) &= \frac{1}{5} \cdot P(c_2) = 0.066, \\
Pb(c_3) &= \frac{1}{10} \cdot P(c_3) = 0.033.
\end{aligned}$$

Therefore, the probability of any base in c_3 being chosen is 10 times smaller than the base in c_1. As each base should have the same probability of being chosen, we need to adjust $P(x)$ probabilities to make all $Pb(x)$ equal. This was done by using a weighted discrete distribution, thus weighting the probability of each chromosome to its size.

3.3 Evolution Simulator

Considering we have a simulator for the process of DNA replication, we can use it to simulate a simplistic view of DNA structure evolution. Some of the parameters we can try to mutate are gene locations and the MFA-Seq replication fork attachment probability landscape. Working DNA sequences have some structural particularities that will be of concern when simulating evolution. Still, we will have to use some assumptions about the internal workings of DNA machinery that are not clear at the moment of the writing of this capstone project report. Some changes were made to the main simulator code for usage by the evolution simulator, and a separate memory management system was implemented.

3.3.1 Evolution

There is a classic artificial evolution architecture, with a fixed population size and fitness proportionate or roulette wheel selection, in which each being has a probability to survive proportional to its fitness (Figure 3.3). The fitness is calculated by taking an average of a number of simulations executed with that organism for certain functions that may be executed on each simulated organism (Figure 3.4). In the following, we will present the currently implemented functions in the evolution model simulator.

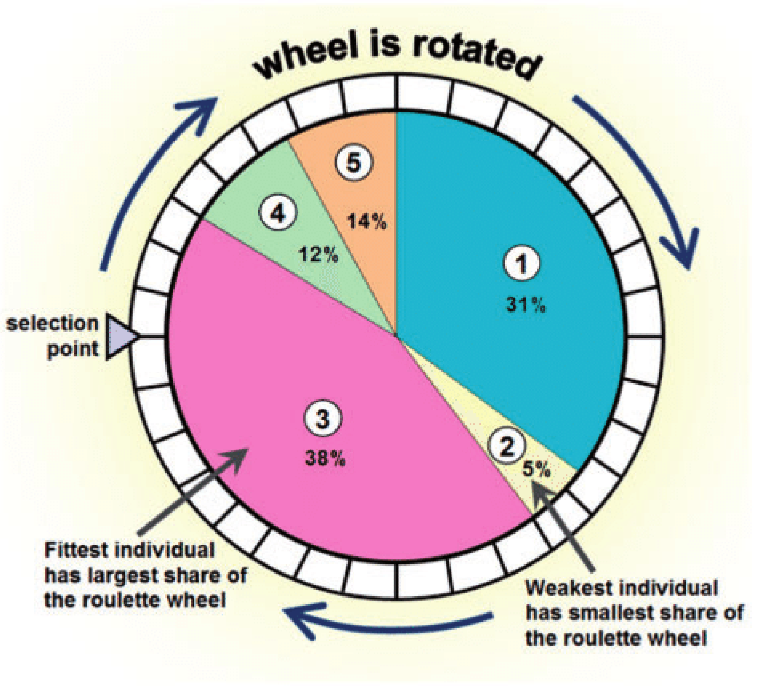
wheel is rotated
5
14%
4
12%
1
31%
selection point
3
38%
2
5%
Fittest individual has largest share of the roulette wheel
Weakest individual has smallest share of the roulette wheel

Figure 3.3: *Description of the roulette wheel selection.*

Implemented functions

The currently implemented functions are:

- min_sphase: Minimizing S-Phase duration;

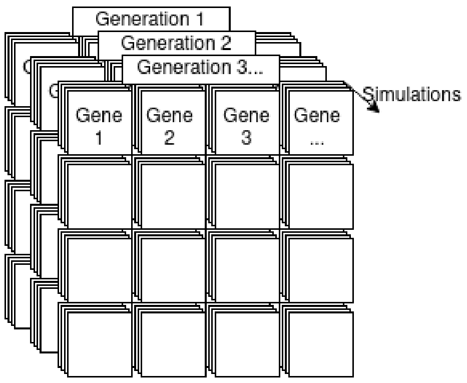

Figure 3.4: *This is the format of the population simulated in each generation. Each cell has its own genes and probability landscapes, and each cell is run for a number of simulations. The simulation statistics are then averaged for final fitness calculation.*

- max_coll: Maximizing collisions in a certain gene;
- min_coll: Minimizing collisions in a certain gene;
- max_coll_all: Maximizing collisions in genes;
- min_coll_all: Minimizing collisions in genes;
- match_mfaseq: Match the MFA-Seq probability Landscape.

If more than one method is specified to be used in the configuration, they are composed by a weighted sum. Fitness for each function is normalized to the range from 0 to 1 before applying the weighted sum. Genomes, replication times and statistics for the entire population are written to the disk every n generations, read from the configuration file. Replication is handled in a way that the worst are killed via the roulette wheel, and the newly created empty population slots are filled with identical copies of the survivors. The surviving ones to multiply are also chosen proportionally from their fitness.

3.3.2 Memory Manager

The memory manager was implemented in order to avoid memory allocations, as they are quite slow. It mainly manages strand duplication time vector allocation, but it is generic enough to support other vector data types. It functions by allocating a new vector only if necessary, keeping previously used but now unused vectors allocated for future use. This works fairly well due to consistent vector lengths for all simulations. Also, it makes some vectors, such as the probability landscape vector (at least for common simulations) global. Because these vectors are usually read-only, we can use the same array for all instances without worry. That way, we consume less memory and don't spend time allocating memory we already have. It is a singleton so instances would not be passed everywhere.

The Memory Manager's functionality is pretty simple, but must be thread-safe. It works by using categories and allocation ids. It is assumed that any memory allocation in a category may be reused by another request in the same category. These are the operations one can run for memory management:

- `vector<T> &getMemorySpace<T>(length)` - Allocates a new CPP vector of type T and size length;
- `void freeMemorySpace(v)` - Frees a previously allocated CPP vector.

The memory manager should thus be pretty easy to use. Some example usage of the manager might be:

```cpp
#include "memory_manager.hpp"

int main () {
    // Allocation will ocurr during first call
    auto vec = MemoryManager::getMemorySpace<int>(10);

    // Do things with the vector

    // Free used memory
    MemoryManager::freeMemorySpace<int>(vec);

    // Allocation should not be required in this call
    auto vec2 = MemoryManager::getMemorySpace<int>(10);

    // Do things with vector
    // Obs. Data should not be used uninitialized

    // Free used memory
    MemoryManager::freeMemorySpace<int>(vec2);
}
```

Program 3.9: *Memory manager usage example code.*

The memory manager was removed after some tests that indicated a lower performance when using it in comparison to simulations without it. More time is used during simulation when using the memory manager, possibly because of cache related issues. The memory manager will not be used for actual simulations while this problem is not resolved.

3.3.3 Gene Location Evolution

Genes are something we cannot actually alter too much. They have their own functions in the organism, so we cannot resize, add or delete them on a whim. The only change we can make without interfering massively on cell functions is moving genes, assuming we are ignoring epigenetics on this simulator.

Gene moving can happen in two different ways in this simulation: The gene is moved in the empty space surrounding it, or its position can be swapped with another random gene in the same chromosome (Figure 3.5).

These two methods of mutation can result in any combination of gene orders and positions if applied repeatedly to a genome.

3.3.4 Probability Landscape Evolution

The probability landscape is defined by epigenetic characteristics, and other unknown biochemical mechanisms. Considering that, we decided to settle on a simplified representa-

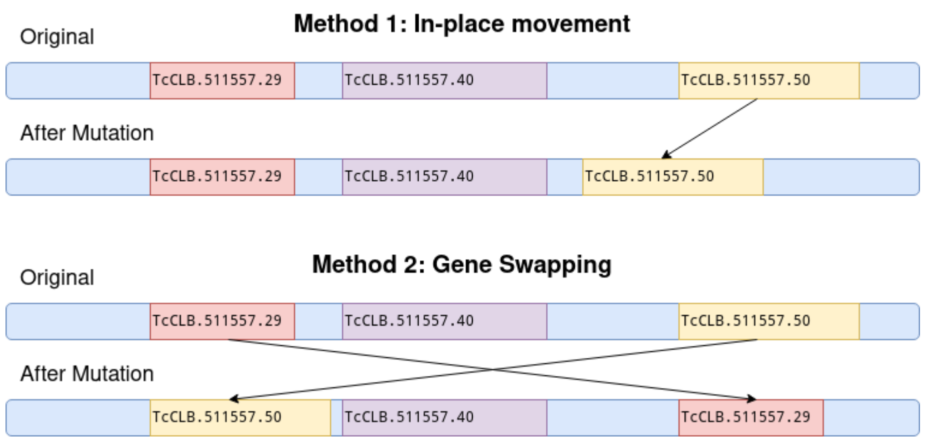

Figure 3.5: *Methods used in the simulator for moving.*

tion of the probability landscape that allows for easier mutation design and implementation. Our approach was to represent the probability landscape as a set of points centered on a base pair, with a bell curve that affects surrounding probabilities. All bell curve parameters can mutate: center, μ and σ. New curves can be added and existing curves may stop existing. From now on, these points will be referred to as "Probability Sources", or only "Sources", for simplicity. Random amounts use a normal distribution for generated values. Some mutations are run individually for each source, checking base mutation probability and then choosing one of the possible source mutations. Possible source mutations are:

- Changes parameter μ of the bell curve, which is also moving the source by a random amount;
- Changes parameter σ of the bell curve of the source by a random amount.

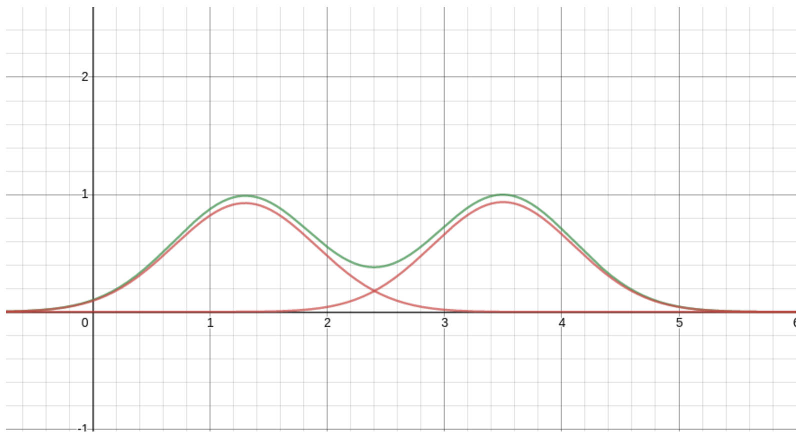

Figure 3.6: *Original probability landscape. Green is normalized sum, red are original sources.*

Some mutations are run externally to a specific source. These are run for each individual chromosome. In this case there is a base probability for non-source changes. Possible non-source mutations are:

- Creates a new source at a random location (uniform distribution) with a random μ and σ;

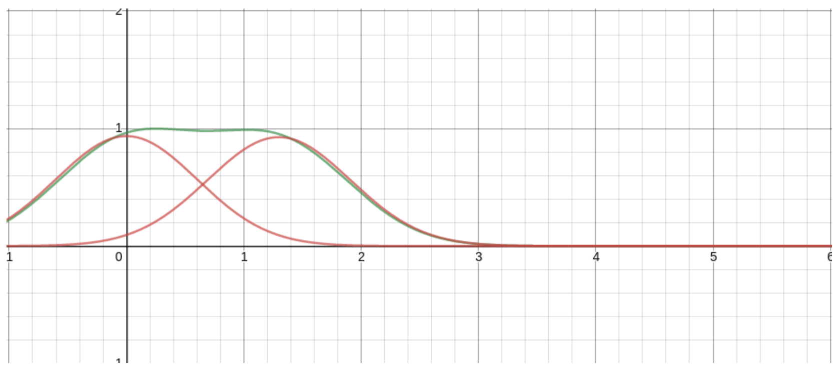

Figure 3.7: *Moving source on probability landscape. Green is normalized sum, red are original sources.*

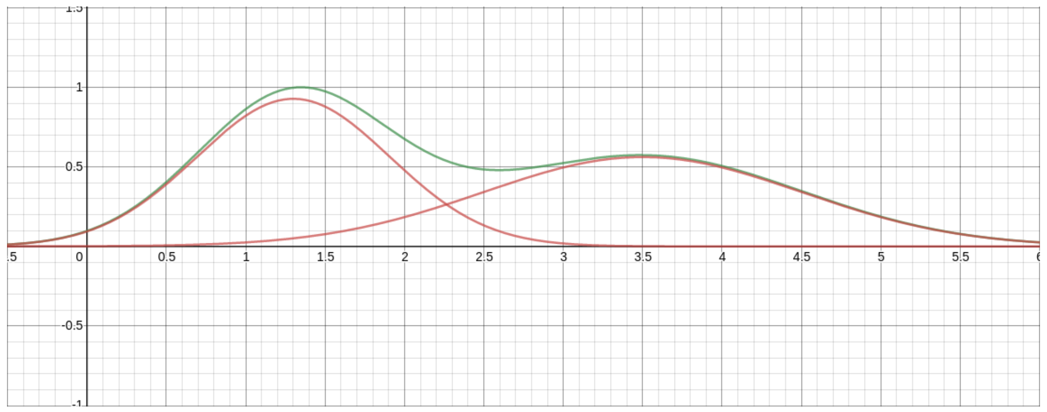

Figure 3.8: *Changing σ on a source. Green is normalized sum, red are original sources.*

- Deletes a random source from the chromosome.

For each base, probabilities of the bell curves are calculated as the normalized sum of the curves.

3.4 Data Analysis

Many scripts were made for utility reasons, such as compression/decompression and FASTA file parsing. Others were made with the purpose of data analysis, processing data and plotting it. Here we describe the usage and purposes of scripts developed external to the ReDyMo-CPP code. The scripts were separated into two categories:

- Base Scripts, which are mostly standalone scripts, or with dependencies only to utility scripts;
- Meta Scripts, which are processes that use one or more base scripts.

Utility Scripts

Some of these scripts were made specifically for importing into other scripts, and not to be run from the terminal. They are mostly helper classes and functions that parse certain files or provide useful functions. These scripts include:

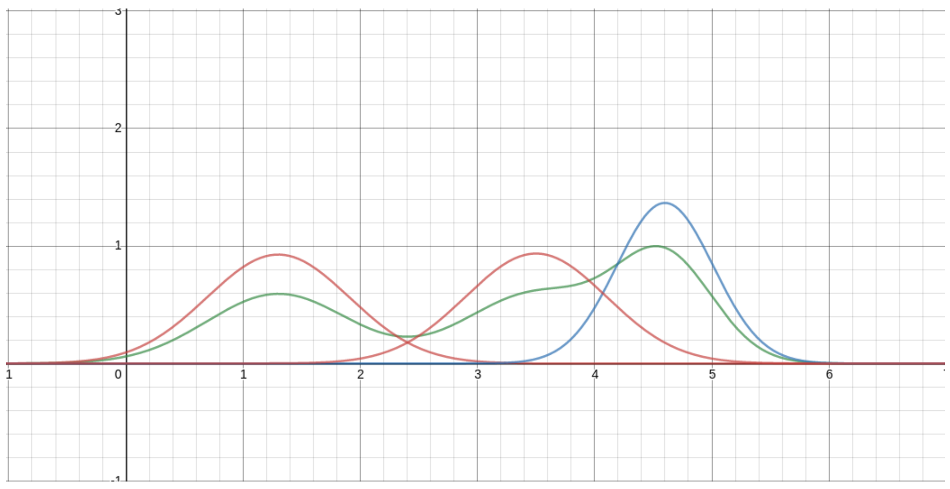

Figure 3.9: *Adding source to probability landscape. Green is normalized sum, red are original sources. Blue is the new source.*

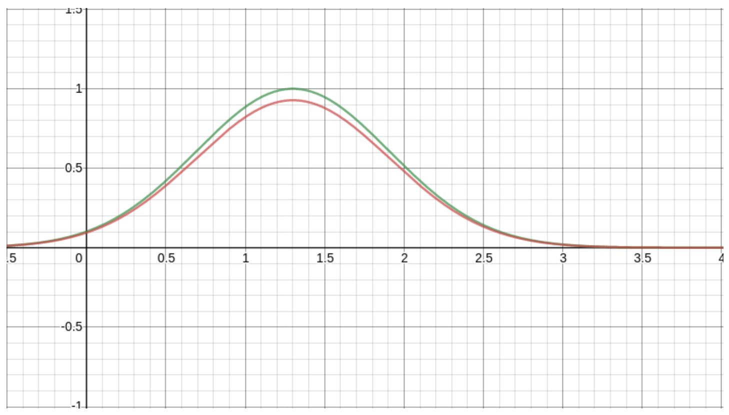

Figure 3.10: *Deleting source from probability landscape. Green is normalized sum, red are original sources.*

- **compressor.py**: Semantic Compressor algorithm implemented in Python 3. It provides a function to compress and decompress an input string stream according to the compression algorithm described in this document. This file is a executable script that can actually be used from the command line;

- **fasta.py**: A module containing the FASTA class, a simple parser for the *.fasta* files acquired from TriTrypDB (Aslett *et al.*, 2010). Not very powerful but enough for the analyses done in this project;

- **data_processing.py**: A module containing useful data processing functions, such as a moving average calculator, genome concatenation and a collision statistics calculator;

- **plotting.py**: A module containing some of the more common plotting procedures used by the other scripts. Plotting of gene regions and moving average are included in this file.

Data Scripts

Some scripts were made for facilitating data analysis and plotting. These can be used to compare and visualize data from simulations:

- **load_genome.py**: Loads FASTA file information into the sqlite3 database used by the simulator, making it easy for future analysis of other organisms;
- **plot_mfaseq.py**: Plots MFA-seq information for quick visualization;
- **plot_results.py**: Plots replication time information of a simulation for easy visualization;
- **compare_results.py**: Plots replication time information of two simulations for result comparison.

3.4.1 DGF-1 to Collision relationship Analysis

This analysis was done to test the hypothesis that the genome has evolved so to maximize collisions (thus maximizing mutations) to the Dispersed gene family protein 1, or DGF-1. The first hurdle was to find a way to detect specifically replication to transcription machinery collisions, ignoring other types. We could modify the simulator so it logs collision locations and types, but many classes would have to be reworked to allow for this. We then noticed that it was actually possible to detect and separate the collisions from the replication times of each base. This resulted in a quite simple external python script that calculates collisions on a given protein. In Figure 3.11, we show the replication times of a chromosome put on a chart.

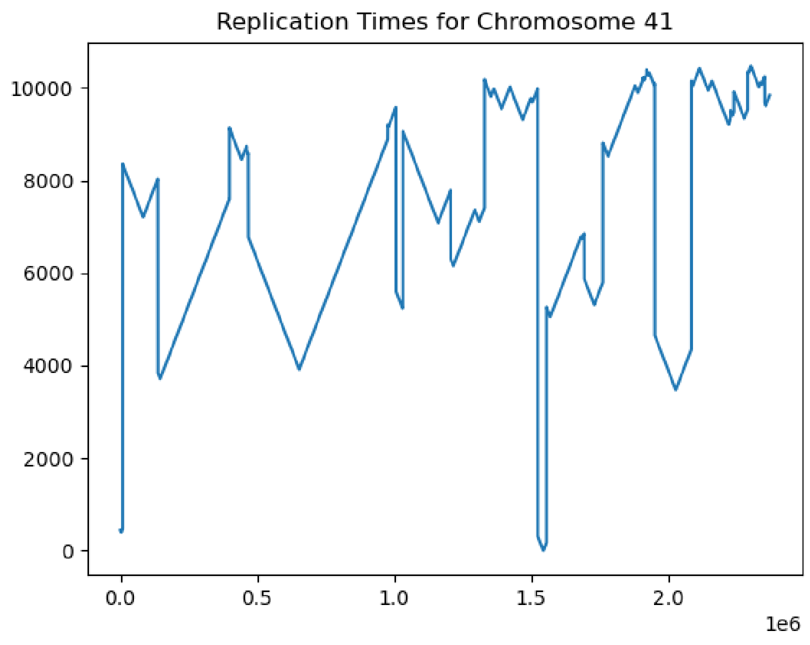

Figure 3.11: *Replication times for chromosome 41 after one simulation.*

Any collision between two instances of collision machinery results in the upside down V-shapes we see in the chart. That is because both sides are replicated and both reach the collision point at the same time. However, collisions between replication and transcription machinery can be visualized on the graph as vertical lines, because on collision, replication stopped. After that, another replication machinery replicates the other side of the collision. This results in a jump of replication times, and in a vertical line in the graph.

Chapter 4

Results and Discussion

In this chapter, we summarize the main results obtained so far with the simulators and programs described previously. Firstly, in Section 4.1, we provide performance results obtained with the GPGPU processing; we do the same in Section 4.2 for the implemented data compressor. Next, in Section 4.3, we compare the *T. brucei* model original behaviour with the one yielded by the correction of the origin firing probability. Finally, in Section 4.5, we provide some results for *T. cruzi* with the new evolution model simulator.

4.1 GPGPU Processing

With the addition of GPGPU processing, it was possible to achieve quite a speed increase in execution times. Initial setup is a bit slower, for data upload to the graphics card, but is overcome by the performance gains on the tested machine. The average time saves due to the addition of GPGPU processing can be seen in Table 4.1

Run Type	Average Simulation Time	Average Simulation Time to Run Time Ratio
CPU parallel execution	1746.06 ms	71.71 %
GPU parallel execution	1354.23 ms	62.44 %

Table 4.1: *Time for running the simulation itself for CPU and GPU*

4.2 Replication Time Data Compression

The simulator was already quite optimized on the start of this work, and zstandard compression had been recently implemented. But writing output to disk was still one of the most time-consuming processes. For faster and more legible save files, a compressor algorithm and format was developed that takes into account the expected output format.

Implementing the compressor has shown a sizeable improvement in execution times:

Compression Type	Average Save Time	Average Save Time to Run Time Ratio
no compression	13375.9 ms	83.47 %
zstandard	1637.4 ms	38.21 %
semantic	300.08 ms	10.18 %

Table 4.2: *Time for writing output to disk for each compression type*

As can be seen on the table, the new compression algorithm showed an increase of more than 5 times in speed for saving simulation data, when compared to zstandard compression.

Because semantic compression uses the shape of the data for its advantage, we can see a smaller file size too, even without considering improvements to file readability and ease of decompression for unorthodox languages that may not have implementations for zstandard compression:

Compression Type	Average Simulation Output Disk Size
no compression	153 Mb
zstandard	1.1 Mb
semantic	172 Kb

Table 4.3: *Space in disk used to store one simulation for each compression type*

The compression and decompression algorithms can also be easily implemented for data streams, not requiring a large amount of RAM.

4.3 Homogeneous vs Non-Homogeneous base attachment

It was found that the random base selection used was not taking into account the lengths of the chromosomes, skewing the probabilities of the bases in different chromosomes. These probabilities may affect final simulated replication times, and lengthen replication periods, as seen in Figure 4.1.

As we can see, we have a sloped graph caused by this method of choosing a random base. As it contains all chromosomes concatenated, we can see that the bases in the first chromosomes, that are smaller, have a higher probability of being chosen, an thus, are most likely to be replicated first. As the chromosomes are sorted according to size in crescent order, this causes a clear tendency on the final replication times.

It is interesting to note that though the values become heavily skewed, it still has a very similar shape to the corrected one, with peaks and valleys almost perfectly synchronized. This shows the simulator is quite robust against general errors and mistakes.

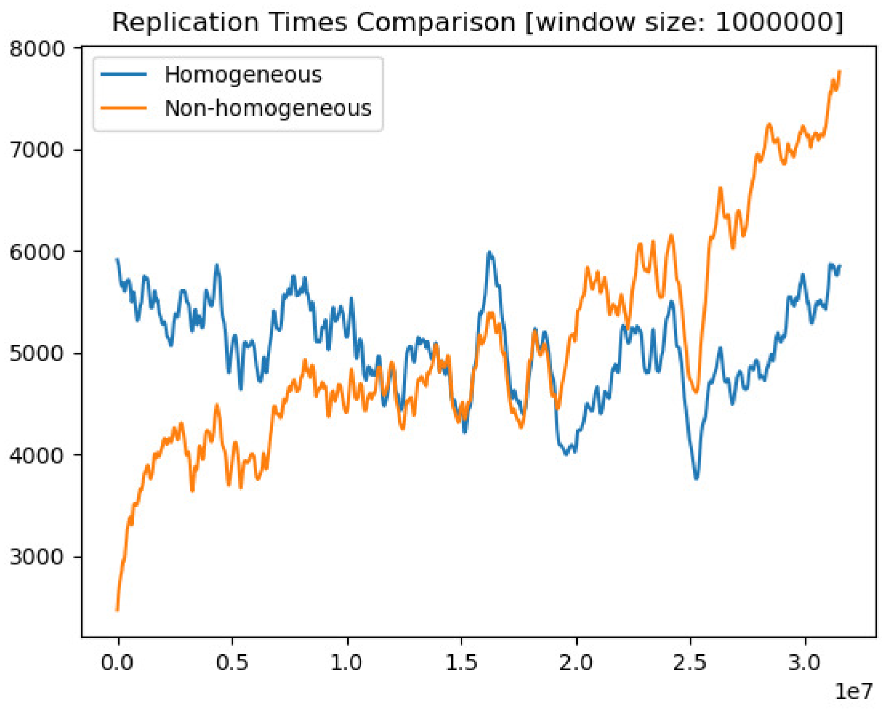

Figure 4.1: *Comparison of effect of homogeneous and non-homogeneous random base selection on replication times*

4.4 DGF-1 region collision analysis

This was an analysis done to verify if the genome has evolved in order to maximize collisions in the Dispersed gene family protein 1, or DGF-1. By utilizing some tricks, we were able to extract collision information from replication time files.

Using the FASTA files, we calculated total base pairs for DGF-1 and all other proteins. We then counted collisions inside DGF-1 proteins and inside other proteins, and normalized with the base pair count, resulting in the number of collisions per base pair.

Doing that, we got the following results:

```
Loading annotations:
Detecting collisions:
Calculating statistics:

Collisions:
TcChr1-S:       Base Pairs: 10394/32965       DGF/OTHER=0.19220571137084275
114385
TcChr2-S:       Base Pairs: 5057/79527        DGF/OTHER=0.8783366569780795
168344
TcChr3-S:       Base Pairs: 0/86326           DGF/OTHER=0.0
169572
TcChr4-S:       Base Pairs: 0/84860           DGF/OTHER=0.0
131173
TcChr5-S:       Base Pairs: 0/82812           DGF/OTHER=0.0
```

```
178372
TcChr6-S:      Base Pairs: 3106/232448      DGF/OTHER=1.0018333725107529
255568
TcChr7-S:      Base Pairs: 10382/221742     DGF/OTHER=1.1750927900363413
258874
TcChr8-S:      Base Pairs: 16683/197729     DGF/OTHER=0.7459664872563952
339370
TcChr9-S:      Base Pairs: 10388/253697     DGF/OTHER=0.3748779393648382
287805
TcChr10-S:     Base Pairs: 0/248717         DGF/OTHER=0.0
325312
TcChr11-S:     Base Pairs: 16707/285897     DGF/OTHER=0.8433824127677805
366511
TcChr12-S:     Base Pairs: 23068/256581     DGF/OTHER=0.8520016834131366
356770
TcChr13-S:     Base Pairs: 10397/263678     DGF/OTHER=0.3291812047060305
362898
TcChr14-S:     Base Pairs: 6762/334057      DGF/OTHER=0.537053093904176
375390
TcChr15-S:     Base Pairs: 0/187488         DGF/OTHER=0.0
260277
TcChr16-S:     Base Pairs: 43577/307780     DGF/OTHER=0.5756802524826716
420342
TcChr17-S:     Base Pairs: 33406/306694     DGF/OTHER=0.3480234756974697
288830
TcChr18-S:     Base Pairs: 21548/271302     DGF/OTHER=0.19805576925430846
531225
TcChr19-S:     Base Pairs: 75148/307653     DGF/OTHER=0.5268883040839344
403897
TcChr20-S:     Base Pairs: 13149/330604     DGF/OTHER=0.8472598659306112
453630
TcChr21-S:     Base Pairs: 23358/339526     DGF/OTHER=0.642520056161434
417348
TcChr22-S:     Base Pairs: 20962/308365     DGF/OTHER=0.3130578175229854
354600
TcChr23-S:     Base Pairs: 36751/310437     DGF/OTHER=0.978506591578264
472478
TcChr24-S:     Base Pairs: 61398/239911     DGF/OTHER=0.2503036364444952
372035
TcChr25-S:     Base Pairs: 55488/322621     DGF/OTHER=0.4555899373850986
413301
TcChr26-S:     Base Pairs: 43575/416927     DGF/OTHER=0.9562461598024145
496880
TcChr27-S:     Base Pairs: 1238/471911      DGF/OTHER=0.2800177008052627
344662
TcChr28-S:     Base Pairs: 92650/253791     DGF/OTHER=0.5306623933926979
540377
TcChr29-S:     Base Pairs: 0/296854         DGF/OTHER=0.0
353759
TcChr30-S:     Base Pairs: 24117/392050     DGF/OTHER=0.5665732815768618
539713
TcChr31-S:     Base Pairs: 40662/392555     DGF/OTHER=0.6527786687471524
477735
TcChr32-S:     Base Pairs: 3794/502101      DGF/OTHER=0.9342083237235888
429552
TcChr33-S:     Base Pairs: 13567/431842     DGF/OTHER=0.3568292273174148
```

```
400773
TcChr34-S:     Base Pairs: 20884/462045      DGF/OTHER=0.8236002741699766
394484
TcChr35-S:     Base Pairs: 26975/558125      DGF/OTHER=0.46591946722892663
504753
TcChr36-S:     Base Pairs: 1556/595399       DGF/OTHER=0.31570595876113894
596814
TcChr37-S:     Base Pairs: 10376/672607      DGF/OTHER=0.8542102611083814
642928
TcChr38-S:     Base Pairs: 17505/383670      DGF/OTHER=0.5082010478614741
488189
TcChr39-S:     Base Pairs: 10454/1034455     DGF/OTHER=1.7093911834066873
901809
TcChr40-S:     Base Pairs: 112305/864465     DGF/OTHER=0.7438664733868388
1232782
TcChr41-S:     Base Pairs: 65815/970657      DGF/OTHER=0.5656396109474302
1951655
GLOBAL: DGF/OTHER=0.6545174524139905
```

Program 4.1: *Average after 10000 simulations with a period of 100 normalized by base pairs*

We can see that at least globally, there is actually a lower collision rate by base pairs for both DGF-1 and other proteins. Interestingly, when compared to other proteins, DGF-1 proteins were found to be about half as likely to incur transcription to replication collisions by base pair density.

```
Loading annotations:
Detecting collisions:
Calculating statistics:

Collisions:
TcChr1-S:      Gene Counts: 1/23             DGF/OTHER=1.393874769353448
114385
TcChr2-S:      Gene Counts: 1/63             DGF/OTHER=3.518681125696975
168344
TcChr3-S:      Gene Counts: 0/56             DGF/OTHER=0.0
169572
TcChr4-S:      Gene Counts: 0/55             DGF/OTHER=0.0
131173
TcChr5-S:      Gene Counts: 0/57             DGF/OTHER=0.0
178372
TcChr6-S:      Gene Counts: 2/139            DGF/OTHER=0.9303705113564269
255568
TcChr7-S:      Gene Counts: 1/129            DGF/OTHER=7.097328975360063
258874
TcChr8-S:      Gene Counts: 3/133            DGF/OTHER=2.7903166027197033
339370
TcChr9-S:      Gene Counts: 1/166            DGF/OTHER=2.5480889315373925
287805
TcChr10-S:     Gene Counts: 0/184            DGF/OTHER=0.0
325312
TcChr11-S:     Gene Counts: 3/183            DGF/OTHER=3.006375681370528
366511
TcChr12-S:     Gene Counts: 5/153            DGF/OTHER=2.3439445239086747
356770
TcChr13-S:     Gene Counts: 1/169            DGF/OTHER=2.1935921484558185
```

```
362898
TcChr14-S:     Gene Counts: 3/203            DGF/OTHER=0.7356082577314128
375390
TcChr15-S:     Gene Counts: 0/129            DGF/OTHER=0.0
260277
TcChr16-S:     Gene Counts: 10/218           DGF/OTHER=1.7768663340734776
420342
TcChr17-S:     Gene Counts: 5/176            DGF/OTHER=1.3343519679748168
288830
TcChr18-S:     Gene Counts: 4/246            DGF/OTHER=0.9674233935885032
531225
TcChr19-S:     Gene Counts: 14/169           DGF/OTHER=1.553579562625429
403897
TcChr20-S:     Gene Counts: 3/279            DGF/OTHER=3.133893291890931
453630
TcChr21-S:     Gene Counts: 3/203            DGF/OTHER=2.991052864659566
417348
TcChr22-S:     Gene Counts: 2/209            DGF/OTHER=2.223865315326991
354600
TcChr23-S:     Gene Counts: 8/210            DGF/OTHER=3.040806229158204
472478
TcChr24-S:     Gene Counts: 12/176           DGF/OTHER=0.9395126769210822
372035
TcChr25-S:     Gene Counts: 9/182            DGF/OTHER=1.5845627425588236
413301
TcChr26-S:     Gene Counts: 9/283            DGF/OTHER=3.142613741598097
496880
TcChr27-S:     Gene Counts: 1/295            DGF/OTHER=0.2167045576625466
344662
TcChr28-S:     Gene Counts: 20/171           DGF/OTHER=1.656355800221348
540377
TcChr29-S:     Gene Counts: 0/209            DGF/OTHER=0.0
353759
TcChr30-S:     Gene Counts: 3/296            DGF/OTHER=3.438811510614799
539713
TcChr31-S:     Gene Counts: 6/262            DGF/OTHER=2.952597297148314
477735
TcChr32-S:     Gene Counts: 1/375            DGF/OTHER=2.647166391976387
429552
TcChr33-S:     Gene Counts: 2/313            DGF/OTHER=1.754420558625388
400773
TcChr34-S:     Gene Counts: 2/315            DGF/OTHER=5.863088508279739
394484
TcChr35-S:     Gene Counts: 4/395            DGF/OTHER=2.223708919712258
504753
TcChr36-S:     Gene Counts: 1/412            DGF/OTHER=0.33992373248010305
596814
TcChr37-S:     Gene Counts: 1/470            DGF/OTHER=6.193429840237265
642928
TcChr38-S:     Gene Counts: 3/294            DGF/OTHER=2.2723012369898097
488189
TcChr39-S:     Gene Counts: 1/707            DGF/OTHER=12.213264598221084
901809
TcChr40-S:     Gene Counts: 12/603           DGF/OTHER=4.856045294787959
1232782
TcChr41-S:     Gene Counts: 17/854           DGF/OTHER=1.926672324203659
```

```
1951655
GLOBAL: DGF/OTHER=2.5759572385170917
AVERAGE GENE LENGTH: DGF/OTHER=5650.586206896552/1435.7409484454938
```

Program 4.2: *Average after 10000 simulations with a period of 100 normalized by gene count*

But, when normalized using gene counts. we get a much greater collision rate in DGF-1 than other genes. This may be caused by DGF-1 genes being in average much longer than other genes. From this data, we can conclude that DGF-1 genes each tend to have more collisions than any other genes, but due to them being larger, each base pair in any DGF-1 gene has a smaller chance of incurring in a collision if a collision does occur in a DGF-1 gene.

4.5 Evolution simulator

Simulations were ultimately not run on a particularly large number of cells or generations (3 instances, 10 cells at most for 100 generations) because of the large RAM capacity requirements for running them. 10 cells reached almost 15 GB of RAM usage, making it impractical for simulations on lower end hardware. This could definetely be a point of improvement in the future.

Because of low simulation count and population size, data acquired from evolution simulations was not reliable nor demonstrated clear tendencies. Some tendencies were seen though like of concentration of higher probabilities on areas with higher concentrations of DGF-1 when optimized for a higher collision rates on DGF-1. Other observed tendency was a concentration of higher probabilities on areas with less genes and areas with longer genes when optimized for lower replication times, as can be seen in Figure 4.2.

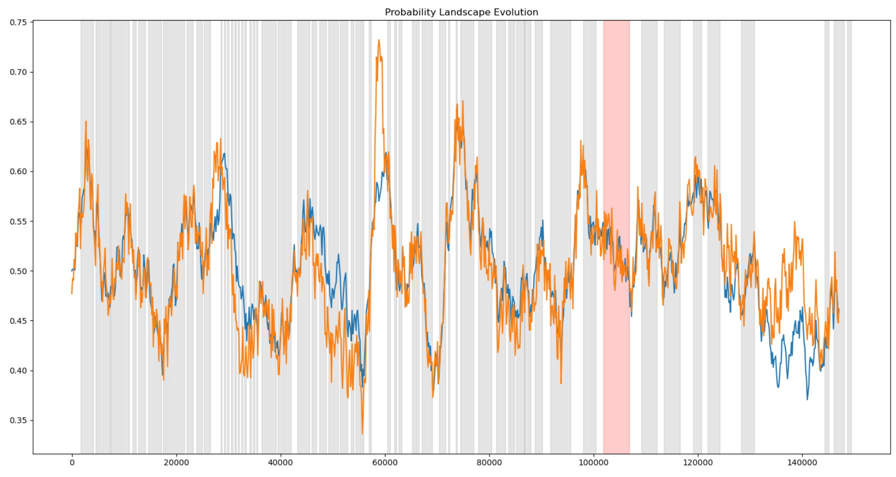

Figure 4.2: *The orange line is the probability landscape before and after evolution has taken place, optimizing for lower replication times after 100 generations, starting with the probability of the blue line. Red sections are the locations of DGF-1 genes, and gray sections other genes.*

Chapter 5

Conclusion

5.1 Summary of the project and its contributions

In this project, we approached several concepts related to the *Trypanosoma cruzi*, as well as some newer findings about the genomic structure of this organism, and its relations to the DGF-1 gene family.

We used, adapted, and improved the C++ version of the ReDyMo simulator, ReDyMo-CPP, for use on the genome of the *Trypanosoma Cruzi*. We simulated and analysed the genomic structure of this organism and the influence it has in collision locations and counts.

On the front for improvement of the simulator, we achieved a number of performance increases and some betterments in architecture and simulator usage:

- A result compressing algorithm that creates readable and smaller files that can be used for fast collision locator;
- Added GPGPU processing capability for basic simulations;
- Added possibility of using a simulation configuration file and seed;
- A fix to the probabilities for choosing a random site for replication fork attachment;
- Expanded the simulator, adding the capability of cell evolution simulation;
- General improvements to the architecture to facilitate future simulator expansions.

For data analysis, DGF-1 collision rates and preliminary evolution simulations were ran, that resulted in a few insights:

- DGF-1 genes have increased collision rates, if considering gene counts, but as DGF-1 genes are much longer than the other, they have lower collision rates by base pair;
- When evolution is optimized for a increase in DGF-1 collisions, probability of replication starting around DGF-1 genes is increased;

- More interestingly, when evolution is optimized for a reduced replication time, probability of replication starting at the beginning of long transcription areas and center of areas with few constitutive transcription regions is increased.

5.2 Future and Current Endeavors

This project has not been completely finished, and work will continue, mainly in the direction of optimization and scaling of evolution simulations, as well as checkpoints and centralization of genetic data in the sqlite3 database.

Future work may follow many paths, some of them may be as described below:

- A more rigorous statistical analysis of the results for DGF-1 collisions, and possibly run the same analysis on other gene families and groups to check for outliers;
- Analysis of data from the evolution simulator with bigger populations, and implementation of more evolutionary pressure aspects;
- Use of the evolution simulator on the *Trypanosoma brucei*, for which this mode of simulation didn't exist.

5.3 Student overview for the project

This project allowed me to dive into topics of molecular biology and genetics that I wouldn't have experienced otherwise, also being my biggest academic contribution. It teached me a lot about real-world application of multithreaded, GPGPU applications, and memory optimization techniques. This project was an all-round important and great experience.

For some self-reflection, I may have focused too much on optimization of the architecture and implementations, and a bit too little on data analysis and visualization. But am quite satisfied with the achieved results, and will continue working on this project for the near future.

References

[Andrade *et al.* 2014] Daniela V. Andrade, Kenneth J. Gollob, and Walderez O. Dutra. “Acute Chagas Disease: New Global Challenges for an Old Neglected Disease”. In: *PLOS Neglected Tropical Diseases* 8.7 (July 2014), pp. 1–10. doi: 10.1371/journal.pntd.0003010. url: https://doi.org/10.1371/journal.pntd.0003010 (cit. on p. 1).

[Araujo *et al.* 2020] Christiane Bezerra de Araujo *et al.* “Replication origin location might contribute to genetic variability in Trypanosoma cruzi”. In: *BMC Genomics* 21.1 (June 2020), p. 414. issn: 1471-2164. doi: 10.1186/s12864-020-06803-8. url: https://doi.org/10.1186/s12864-020-06803-8 (cit. on p. 2).

[Aslett *et al.* 2010] Martin Aslett *et al.* “TriTrypDB: a functional genomic resource for the *Trypanosomatidae*”. In: *Nucleic Acids Research* 38.suppl 1 (2010), pp. D457–D462. doi: 10.1093/nar/gkp851 (cit. on p. 25).

[Calderano *et al.* 2015] Simone G. Calderano *et al.* “Single molecule analysis of *Trypanosoma brucei* DNA replication dynamics”. In: *Nucleic Acids Research* (2015), gku1389. doi: 10.1093/nar/gku1389 (cit. on p. 1).

[García-Muse and Aguilera 2016] Tatiana García-Muse and Andrés Aguilera. “Transcription-replication conflicts: how they occur and how they are resolved”. In: *Nature Reviews Molecular Cell Biology* 17.9 (2016), pp. 553–563. issn: 1471-0080. doi: 10.1038/nrm.2016.88 (cit. on p. 1).

[Gindin *et al.* 2014] Yevgeniy Gindin, Manuel S Valenzuela, Mirit I Aladjem, Paul S Meltzer, and Sven Bilke. “A chromatin structure-based model accurately predicts DNA replication timing in human cells”. In: *Molecular Systems Biology* 10.3 (2014), p. 722. doi: https://doi.org/10.1002/msb.134859. eprint: https://www.embopress.org/doi/pdf/10.1002/msb.134859. url: https://www.embopress.org/doi/abs/10.1002/msb.134859 (cit. on p. 1).

[Scholl BB 2018] da Silva Scholl BB. *Multiscale models of cell cycle dynamics in trypanosomatids.* 2018. url: http://resumosrca.butantan.gov.br/Abstract2015/Details/1370?grid-page=109 (cit. on p. 2).

[G. R. C. Silva 2017] Gustavo Rodrigues Cayres Silva. *Desenho e simulação de modelos de computacionais da dinâmica de replicação de DNA em kinetoplastídeos.* 2017. url: https://linux.ime.usp.br/~gustavoc/tcc/proposta.html (cit. on p. 2).

[M. d. Silva *et al.* 2019] M.S. da Silva *et al.* “Transcription activity contributes to the firing of non-constitutive origins in African trypanosomes helping to maintain robustness in S-phase duration”. In: *Nature Scientific Reports* 9.1 (2019), p. 18512. issn: 2045-2322. doi: 10.1038/s41598-019-54366-w. url: https://doi.org/10.1038/s41598-019-54366-w (cit. on pp. 1, 2).

[M. S. d. Silva *et al.* 2017] Marcelo S. da Silva, Paula Andrea Marin Muñoz, Hugo A. Armelin, and Maria Carolina Elias. “Differences in the Detection of BrdU/EdU Incorporation Assays Alter the Calculation for G1, S, and G2 Phases of the Cell Cycle in Trypanosomatids”. In: *Journal of Eukaryotic Microbiology* (2017). doi: 10.1111/jeu.12408 (cit. on p. 1).

[Tiengwe *et al.* 2012] Calvin Tiengwe *et al.* “Genome-wide analysis reveals extensive functional interaction between DNA replication initiation and transcription in the genome of *Trypanosoma brucei*”. In: *Cell Reports* 2.1 (2012), pp. 185–197. doi: 10.1016/j.celrep.2012.06.007 (cit. on p. 1).

[Voet *et al.* 2008] Donald Voet, Judith G Voet, and Charlotte W Pratt. *Principles of biochemistry.* Vol. 4. Wiley New York, 2008 (cit. on p. 5).

[Yea-Lih Lin 2017] Philippe Pasero Yea-Lih Lin. *Transcription-Replication Conflicts: Orientation Matters.* 2017. url: https://doi.org/10.1016/j.cell.2017.07.040 (cit. on p. 9).