B.Sc. in Computer Science and Engineering Thesis

Implementation of Genome Transposition in GRIMM

Submitted by

Rafi Imran Noor
201014055

Nabila Binte Alam
201014039

Saroar Ahmed
201014046

Supervised by

Dr. M. Sohel Rahman
Professor, Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

Department of Computer Science and Engineering
Military Institute of Science and Technology

CERTIFICATION

This thesis paper titled **"Implementation of Genome Transposition in GRIMM"**, submitted by the group as mentioned below has been accepted as satisfactory in partial fulfillment of the requirements for the degree B.Sc. in Computer Science and Engineering on December 2013.

Group Members:

Rafi Imran Noor
Nabila Binte Alam
Saroar Ahmed

Supervisor:

(Signature of the Supervisor)
Prof. Dr. M. Sohel Rahman
Department of CSE
Bangladesh University of Engineering and Technology

CANDIDATES' DECLARATION

This is to certify that the work presented in this paper is the outcome of the investigation and research carried out by the following students under the supervision of Prof. Dr. M. Sohel Rahman, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh.

It is also declared that neither this thesis paper nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.

_____

Rafi Imran Noor
201014055

_____

Nabila Binte Alam
201014039

_____

Saroar Ahmed
201014046

ACKNOWLEDGEMENT

We are thankful to Almighty Allah for his blessings for the successful completion of our thesis. Our heartiest gratitude, profound indebtedness and deep respect go to our supervisor Prof. Dr. M. Sohel Rahman, Department of CSE, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh, for his constant supervision, affectionate guidance and great encouragement and motivation. His keen interest on the topic and valuable advices throughout the study was of great help in completing thesis.

We are especially grateful to the Department of Computer Science and Engineering (CSE) of Military Institute of Science and Technology (MIST) for providing their all out support during the thesis work.

Finally, we would like to thank our families and our course mates for their appreciable assistance, patience and suggestions during the course of our thesis.

Dhaka
December 2013

Rafi Imran Noor
Nabila Binte Alam
Saroar Ahmed

ABSTRACT

In Computational biology, genome rearrangement by transposition is an important problem. Genome evolution analysis by transpositions leads to a combinatorial optimization problem of sorting by transpositions. Though the status of the transposition distance problem was open, a non-trivial proof for the NP-completeness of the transposition median problem is given [2]. In this paper, we utilize a 1.375-approximation algorithm for sorting by transpositions [16]. GRIMM is a tool for analyzing rearrangements in pairs of genomes, including unichromosomal and multichromosomal genomes, and signed and unsigned data. In GRIMM, reversal is the only operation for unichromosomal genome rearrangement. In this paper, we implement transposition in GRIMM as another unichromosomal operation. We also compute the transpositional distance between two genomes and compute it with the corresponding reversal distance. At the end of the paper, we also show that, we merge, two algorithms [16] and [11], we reduced the algorithm [16] in our implementation of transposition.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF ABBREVIATION

GRAPPA : Genome Rearrangement Analysis under Parsimony and other Phylogenetic Algorithms

MGR : Multiple Genome Rearrangement

TE : Transposition Element

TD : Transposition Diameter

TDS : Transposition Diameter of Simple Permutation

CPM : Combinatorial Pattern Matching

CHAPTER 1

INTRODUCTION

In recent years, the genomes of more and more species have been sequenced, to provide data for genome rearrangement measures, where the most important distance measures are the reversal distance and the transposition distance.In genome rearrangement, the two compared genomes are represented by permutations, where each element stands for a gene, and the goal is to find a shortest sequence of rearrangement operations that sorts one permutation into the other. Previous work focused mainly on the sorting of a permutation by reversal operations. This problem was shown to be NP-hard [3]. Hannenhalli and Pevzner shows that for signed permutations (every element of the permutation has a sign, which represents the direction of the corresponding gene; a reversal reverses the order of the elements it operates on and flips their signs), the problem becomes polynomial [13]. The algorithm is based on representing a permutation using a breakpoint graph (we defer a formal definition in the next chapter) which decomposes uniquely into disjoint cycles, and studying the effect of a reversal on its cycle decomposition.

Transposition is a genome rearrangement operation where a segment of genes moves from one place to another in the chromosome. A transposition is said to happen when two consecutive markers of a genome exchange their positions. It is always possible to produce the same result as a transposition with a sequence of three reversals. Thus a sequence of m transpositions can always be transformed in a sequence of $3m$ reversals. Therefore we can say that transposition is an intrachromosomal operation. Though in recent years the status of the transposition distance problem was still open, now it is proved that, transposition distance problem is NP-Complete [2].

Though there has been less progress on the problem of sorting by transpositions. Several

1.5-approximation algorithms are known for it [4] [14] [20]. But in this paper, we are going to implement a 1.375 approximation algorithm by Ilias and Hertmen [16].

Hannenhalli and Pevzner [12] give a polynomial time algorithm genomic sort for computing the distance between two multichromosomal genomes, where the distance is the minimum number of reversals, translocations, fissions, and fusions required to transform one genome to the other. Glenn Tesler have implemented this algorithm in full in a program GRIMM [8] available on the web [5].

GRIMM is a tool for analyzing rearrangements in pairs of genomes, including unichromosomal and multichromosomal genomes, and signed and unsigned data. GRIMM concatenates all the chromosomes together in an order determined by its algorithms. This lets you see how translocations, fissions, and fusions are emulated by reversals that cross chromosome boundaries. But there is no scope for performing operation like transposition or block transfer operations.

1.1 Contribution of this Thesis

In this paper we are going to integrate the 1.375 approximation algorithm for transposition operation with the previous algorithms used in GRIMM tool. We will also try to reduce the algorithm and used it in GRIMM for better performance. This is described in detail in Chapter 4. We will also compare our results with the previous result on unichromosomal genomes in GRIMM using graphical representation.

CHAPTER 2

PRELIMINARIES

Transposition is a well known topic in genome rearrangement. In this paper we only concentrate on Transposition. For this purpose we have to study some related topics before. Those topics are mainly discussed in this chapter

2.1 Genome

A genome is an organism's complete set of DNA, including all of its genes. Each genome contains all of the information needed to build and maintain that organism. In humans, a copy of the entire genome-more than 3 billion DNA base pairs-is contained in all cells that have a nucleus. In other words we can say that the genome is the entire hereditary information of an organism. Genomes are partitioned into chromosomes. A chromosome can be linear (eukaryotes), or circular (prokaryotes).

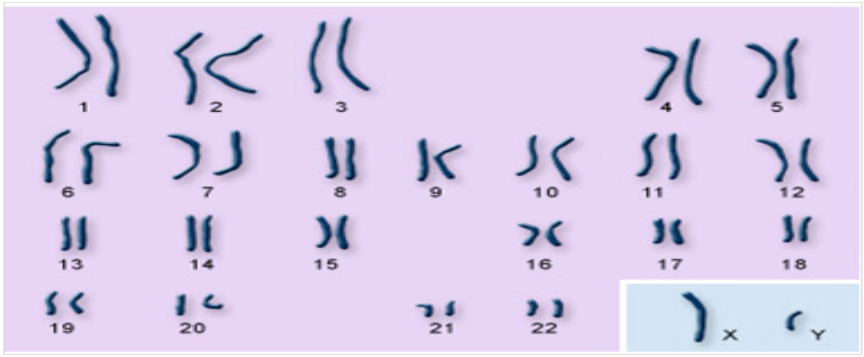


Figure 2.1: Human Chromosomes

2.2 Genome Rearrangement

Watterson et al. first proposed the minimum number of chromosomal reversals necessary to transform one ordering into other. The arrangement distance between single chromosome

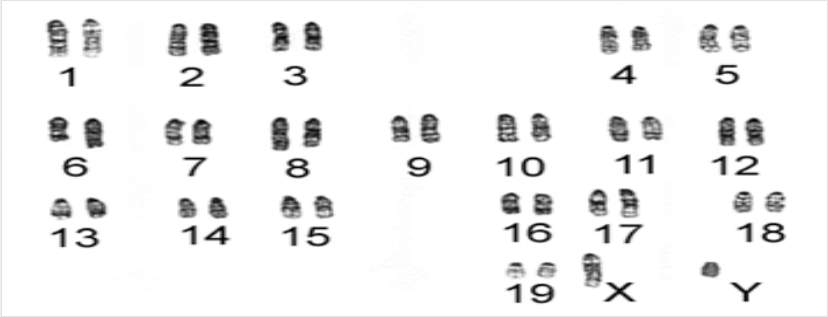


Figure 2.2: Mouse Chromosomes

genomes can be estimated as the minimum number of reversals required to transform the gene ordering observed in one into that observed in the other. This measure, known as "reversals distance" .It can be computed as the reversal distance between signed permutations (π).Where multiple nucleotides are modified. Some operations are such like-

- Reversal
- Translocation
- Transposition
- Inversion
- Deletion
- Fission
- Fusion
- Duplication

2.3 Transposition

A transposition is said to happen when two consecutive markers of a genome exchange their positions. It is always possible to produce the same result as a transposition with a sequence of three reversals. Thus a sequence of m transpositions can always be transformed in a sequence of 3m reversals.

2.4 How Can Transposons Move Through the Genome?

Several mechanisms of transposition are found in prokaryotes as well as eukaryotes. In the E. coli bacteria there are replicative and conservative methods of transposition. In the replicative way, the new copy of the transposable element appears at a new site. The original element stays at the old location. This is duplication. In the conservative way there does not exists a copy of the original element. The element moves from one site in the chromosome to another. This means the transposable element is jumping around in the genome of the host. By these movements through the genome of the host, transposons can generate a large amount of deletions and inversions in the genome. Special kind of transposable elements are retroviruses. Retroviruses are viruses that can integrate their genome in the genome of the host, by copying RNA into DNA by the enzyme reverse transcriptase. It can stay there for a long time, until the integrated genome of the virus gets some kind of signal that makes it copying itself out of the host genome.

2.5 Genome Representation

- A genome is grouped into chromosomes (linear/circular).
- A gene g on the forward strand is represented by [-g,+g]
- Telomeres are represented by the special symbol ‘o’.
- A gene g on the reverse strand is represented by [+g,-g]
- An adjacency (intergenic region) is encoded by the unordered pair of neighboring gene/telomere ends.

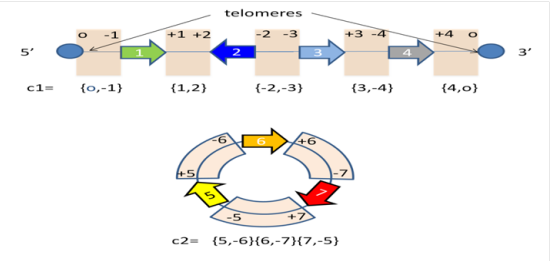


Figure 2.3: Genome Representation

Example:

- linear c1=(o 1 -2 3 4 o)
- circular c2=(5 6 7)

2.6 Genomic Distance

Every genome rearrangement study involves solving a combinatorial puzzle to find a series of genome rearrangements to transform one genome into another. Palmer and co-authors [18] pioneered studies of the shortest (most parsimonious) rearrangement scenarios and applied this approach to plant mtDNA and cpDNA. Since then, the analysis of the most parsimonious scenarios has become the dominant approach in genome rearrangement studies. For unichromosomal genomes, it usually amounts to analysis of inversions (also known as reversals), which are the most common rearrangement events. The problem of finding the minimum number of reversals to transform one unichromosomal genome into another is known as the reversal distance problem. For multichromosomal genomes, the most common rearrangements are reversals, translocations, fusions, and fissions, and the number of such rearrangements in a most parsimonious scenario is known as the genomic distance between multichromosomal genomes [13].

2.7 Synteny Blocks and Segments

Synteny: describes how genomic segments are located on the same chromosome or close to each other such as Genes, markers (any sequence).
Shared synteny between two species: genes are located close to each other in both of the species.
Synteny block (or syntenic block): A set of genes or markers that co-occur together in two species.
Synteny segment (or syntenic segment): Syntenic block where the order of genes or markers is preserved.

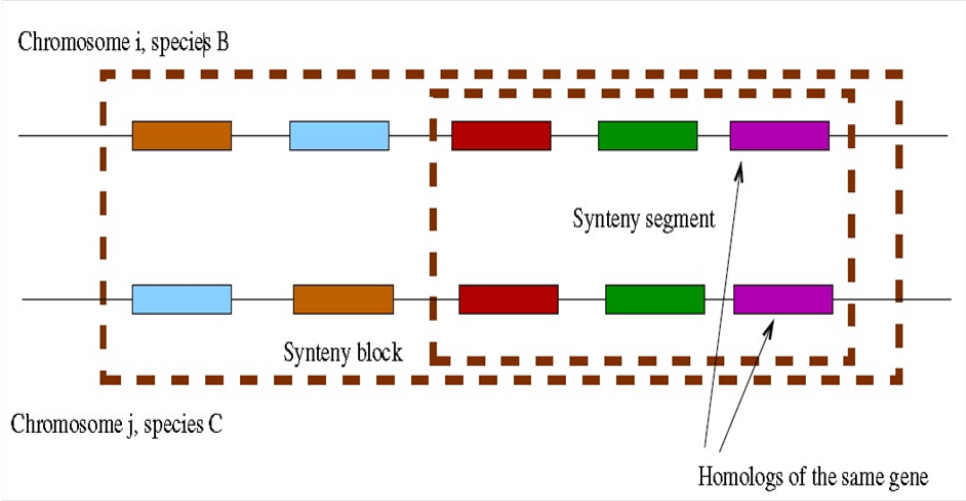


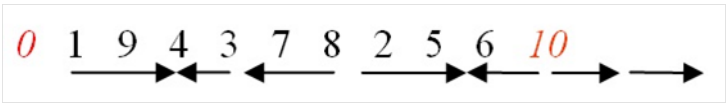
Figure 2.4: Synteny Segment and Block

2.8 Strip

An interval between two consecutive breakpoints in a permutation is called a strip. This is 2 types:

Decreasing strip: a strip of elements in decreasing order (e.g. 6 5 4 and 3 2).

Increasing strip: a strip of elements in increasing order (e.g. 6 7 8 9).



A single-element strip can be declared either increasing or decreasing. We will choose to declare them as decreasing with exception of the strips 0 and n + 1.

2.9 Linear and Circular Permutations

A transposition cuts the permutation at 3 points. Circular transpositions can be represented by exchanging any 2 of the 3 segments.

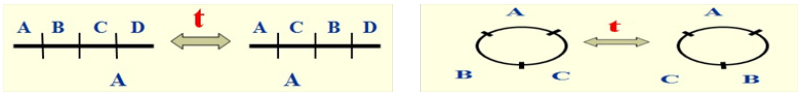


Figure 2.5: Linear & Circular Transposition

Linear and Circular Equivalence:

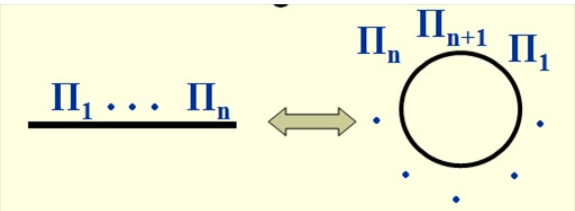


Figure 2.6: Linear and Circular Equivalence

2.10 Adjacency and Breakpoint

Let $\pi = \pi_1\pi_2\pi_3.....\pi_{n-1}\pi_n$ be a permutation. A pair of elements π_I and π_{i+1} is called an adjacency if

$$\pi_{i+1} = \pi_i + 1$$

The remaining pairs are called breakpoints.

Example:

π = 1 9 3 4 7 8 2 6 5

- (3, 4), (7, 8) and (6,5) are adjacent pairs.
- (1,9), (9,3), (4,7), (8,2) and (2,5) are breakpoints.

2.11 Breakpoint Graph

Let a permutation is 1 6 5 4 7 3 2. Here we replace each element j by 2j-1,2j. So, the permutation becomes π = 1 2 11 12 9 10 7 8 13 14 5 6 3 4.

The black edges we get by joining (π_{2i}, π_{2i+1}) and the gray edges we get by joining (2i, 2i+1).This is the unique decomposition into cycles

$c_{odd}(\pi)$ is the number of odd cycles in G(π). Define $\Delta\, c_{odd}(\pi,t) = c_{odd}(t,\pi) - c_{odd}(\pi)$. For all

t and π, Δ c_{odd}(π,t) = 0, 2, -2 [Lemma [BP98]].The circular Breakpoint graph G(π) is shown in Figure-2.7

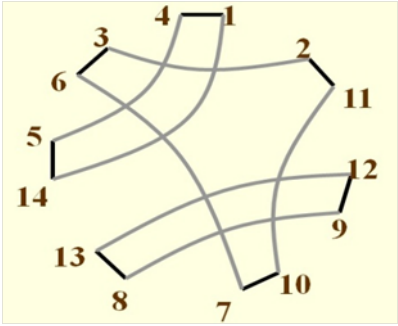


Figure 2.7: Breakpoint Graph 1

Effect of Graph Example 1: Let another permutation that is 1 3 2. After extension the permutation becomes 1 2 5 6 3 4. Here the number of cycles is increased by 2. The breakpoint graph is shown in Figure-2.8

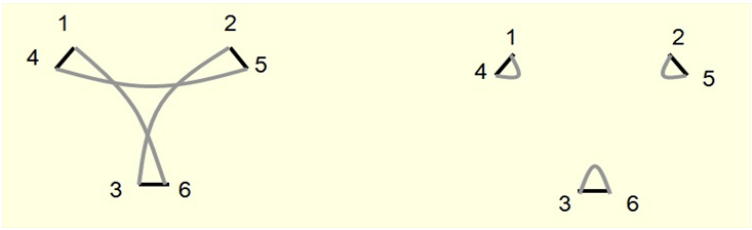


Figure 2.8: Breakpoint Graph 2

Effect of Graph Example 2: Let another one permutation is 6 5 4 3 2 1. After extension of the permutation we get the the permutation becomes 11 12 9 10 7 8 5 6 3 4 1 2. The number of cycle remains 2. The breakpoint graph is shown in figure-2.9

Max number of odd cycles n, is in the id permutation, thus-

- Lower bound [BP98]: For all π, d(π)=[n-c_{odd}(π)]/2.
- Goal: increase number of odd cycles in G.
- t is a k-transposition if Δc_{odd}(π,t) = k.
- A cycle that admits a 2-transposition is oriented.

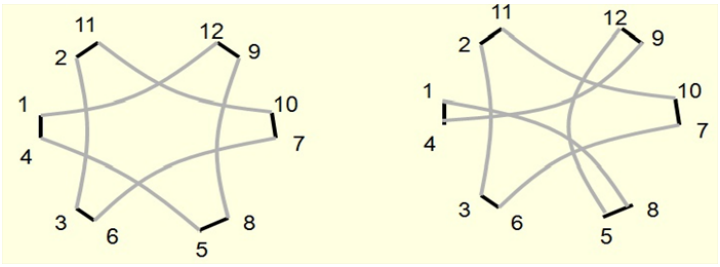


Figure 2.9: Breakpoint Graph 3

2.12 Cycles

Since the degree of each vertex is exactly 2, the graph uniquely decomposes into cycles. Denote the number of cycles in G(π) by c(π). The length of a cycle is the number of black edges it contains. A k-cycle is a cycle of length k and it is odd if k is odd. The number of odd cycles is denoted by $c_{odd}(\pi)$ and let $c_{odd}(\pi, \tau) = \text{codd}(\tau, \pi) - c_{odd}(\pi)$ [16].

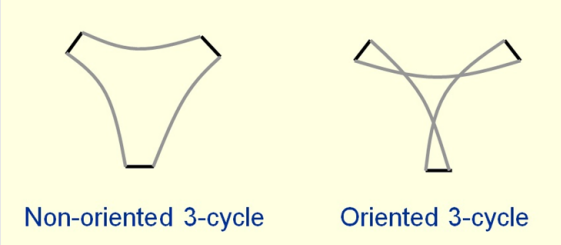


Figure 2.10: 2 possible configurations of 3-cycles

Throughout the paper, we also use the term permutation when referring to the breakpoint graph of the permutaion. For example, when we say that π contains an oriented cycle, we mean that G(π) contains an oriented cycle.

2.13 Simple Permutation

A k-cycle in the breakpoint graph is called short if $k \leq 3$; otherwise, it is called long. A breakpoint graph is simple if it contains only short cycles. A permutation π is simple if G (π) is simple and it is a 2-permutation (respectively, 3-permutation) if G (π) contains only 2-cycles (3-cycles). A common technique in the genome rearrangement literature is to transform permutations with long cycles into simple permutations. This transformation consists of inserting new elements into the permutations and thereby splitting the long cycles. The

reader is referred to [19] for a thorough description. If $\hat{\pi}$ is the permutation attained by inserting elements into π, then $d(\pi) \leq d(\hat{\pi})\rho$ since inserting new elements only can result in a permutation that requires more moves to be sorted. Such a transformation is called safe if it maintains the lower bound of Theorem 2, i.e., if $n(\pi) - c_{odd}(\pi) = n(\hat{\pi}) - c_{odd}(\hat{\pi})$ [16].

2.14 Interactions Between Cycles

Two pairs of black edges, (a, b) and (α, β), are said to intersect if their edges occur in alternated order in the breakpoint graph, i.e., in order a, α, b, β Cycles C and D intersect if there is a pair of black edges in C that intersects with a pair of black edges in D (see Figure-2.11 (c)). Similarly, two triplets of black edges (a, b, c) and (α, β, γ) are interleaving if their edges occur in alternated order, i.e., in order a, α, b, β, c, γ. Two 3-cycles are interleaving if their edges interleave (see Figure-2.11 (e)). A 3-cycle C is shattered if each pair of black edges in C intersects with a pair of black edges from some other 3-cycle [16].

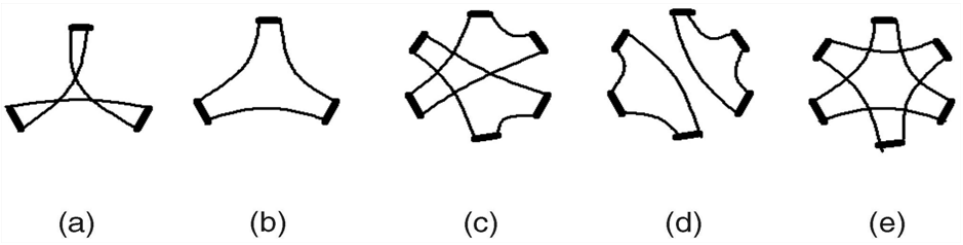


Figure 2.11: Interactions between Cycles

2.15 Sequence of Transpositions

An (x, y)-sequence of transpositions on a simple permutation (for $x \geq y$) is a sequence of x transpositions such that, at least y of them are 2-moves and that leaves a simple permutation at the end. For example, a 0-move followed by two consecutive 2-moves (which is called a (0, 2, 2)-sequence in the papers [4], [15]) is a (3, 2)-sequence. An a/b-sequence is an (x, y)-sequence such that $x \leq a$ and $x/y \leq a/b$. A configuration (or component or permutation) has an (x; y) (or a/b) sequence if it is possible to apply such a sequence on its cycles [16].

(0, 2, 2)-Sequence:

- A (0, 2, 2)-sequence is a sequence of 3 transpositions: the 1^{st} is a 0-transposition and the next two are 2-transpositions.
- A series of (0, 2, 2)-sequences preserves a 1.5 approximation ratio.
- Throughout the algorithm we show that there is always a 2-transposition or a (0, 2, 2)-sequence.

2.16 Transposition Diameter

The transposition diameter, TD (n), of the symmetric group is the maximum value of d(π) taken over all permutations of n elements, i.e., TD(n) = $\max_{\pi} : {}_{n(\pi)-n}$d($\pi$). Similarly, the transposition diameter of simple permutations (denoted by TDS), 2-permutations (TD2), and 3-permutations (TD3) is the longest distance for any such permutation to the identity [16].

2.17 Necessary Lemmas for 1.5 Approximation Algorithm

Necessary Lemmas for 1.5 Approximation Algorithms are as follows [14]

Lemma 4: Every simple permutation π can be transformed into a 3-permutation $\hat{\pi}$ by safe paddings. Moreover, every sorting of $\hat{\pi}$ mimics a sorting of π with the same number of operations.

Lemma 7: Let π be a permutation that contains two unoriented, interleaving cycles C and D that do not form a 1-twisted pair. Then π admits a (0, 2, 2)-sequence.

Lemma 10: Let π be a permutation that contains a closed configuration in which there are two intersecting 0-twisted cycles C and D. Then π admits a (0, 2, 2)-sequence.

Lemma 12: Let π be a permutation that contains a closed configuration with two intersecting, 1-twisted cycles. Then π admits a (0, 2, 2)-sequence.

Lemma 13: Let π be a permutation that contains a 0-twisted cycle, which intersects with the coupled edges of a 1-twisted cycle. Then π admits a (0, 2, 2)-sequence.

Lemma 14: Let π be a permutation that contains $k \geq 2$ mutually interleaving 1-twisted cycles, such that all their twists are consecutive on the circle and k is maximal with this property. Then π admits a (0, 2, 2)-sequence.

2.18 1.5 Approximation Algorithms

It is described below and necessary elements to understand the algorithm is described earlier. The algorithm is [14]-

Algorithm 1 1.5 Approximation Algorithms

1: Transform π into a 3-permutation $\hat{\pi}$ (Lemma 4)

2: **while** $G(\hat{\pi})$ contains a 3-cycle C do:

If C is oriented, apply a 2-operation.

Otherwise, find a cycle D that intersects with a coupled pair of C.

If C and D interleave, apply a (0, 2, 2)-sequence (Lemmas 7, 14).

Else if C or D are 1-twisted, apply a (0, 2, 2)-sequence (Lemmas 12, 13).

Otherwise, apply a (0, 2, 2)-sequence (Lemma 10).

3: Mimic the sorting of π using the sorting of $\hat{\pi}$ (Lemma 4).

CHAPTER 3

GRIMM

There are many genome rearrangement tool in the present world. Some of them are more efficient. GRIMM is one of them. It gives a better result comparing to others. In this chapter we discribed all about the GRIMM tool.

3.1 What is GRIMM (Genome Rearrangements in Man and Mouse)?

A tool for analyzing rearrangements in pairs of genomes, including unichromosomal and multichromosomal genomes, and signed and unsigned data [9].

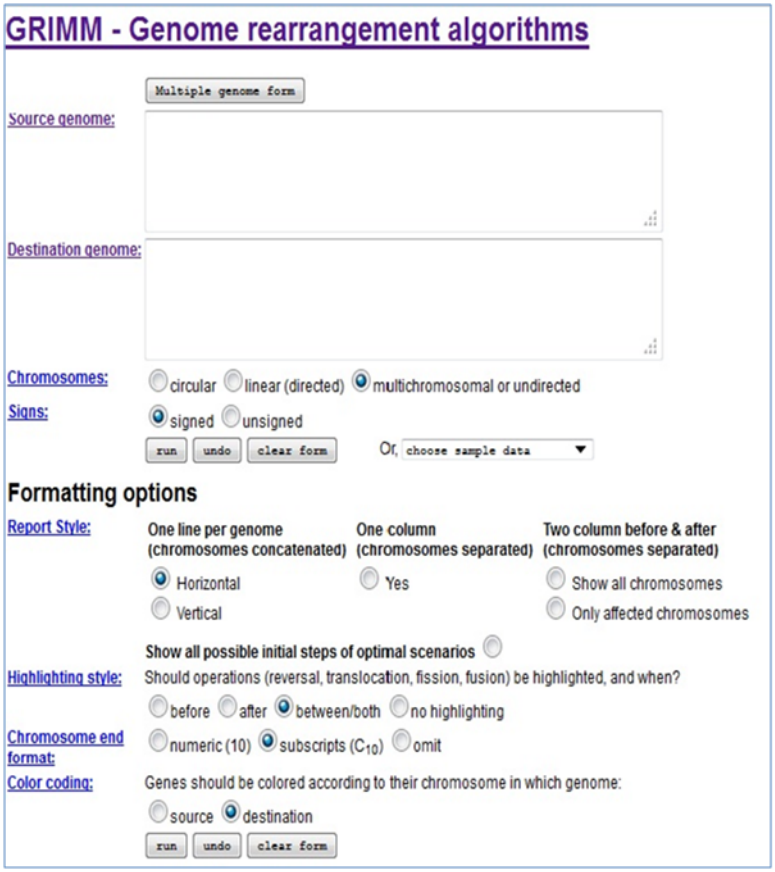


Figure 3.1: GRIMM Tool (Genome Rearrangements in Man and Mouse Tool)

Two genomes may have many genes in common, but the genes may be arranged in a different sequence or be moved between chromosomes. Such differences in gene orders are the results of rearrangement events that are common in molecular evolution. For example, in unichromosomal genomes, the most common rearrangement events are reversals, in which a contiguous interval of genes is put into the reverse order. For multichromosomal genomes, the most common rearrangement events are reversals, translocations, fissions, and fusions, which are described later.

The pairwise genome rearrangement problem is to find an optimal scenario transforming one genome to another via these rearrangement events. We provide a C program and a web tool combining rearrangement algorithms for unichromosomal and multichromosomal genomes, with either signed or unsigned gene data. In each case, it computes the minimum possible number of rearrangement steps, and determines a possible scenario taking this number of steps.

GRIMM implements the Hannenhalli-Pevzner algorithms for computing unichromosomal and multichromosomal genomic distances, making extensive use of code that was adapted from GRAPPA for the unichromosomal problem. GRIMM also implements the Hannenhalli-Pevzner algorithm for computing the reversal distance between two unsigned unichromosomal genomes, and Tesler's algorithm for computing the distance between two unsigned multichromosomal genomes.

3.2 Source and Destination Genomes

We consider a unichromosomal genome [10] to be of a sequence of n genes. The genes are represented by numbers 1, 2, ..., n. The two orientations of gene i are represented by i and -i. A genome is represented as a signed permutation of the numbers 1, 2, ..., n. For example, one unichromosomal genome with n=5 genes is

5 -3 4 2 -1

A multichromosomal genome consists of n genes spread over m chromosomes. We represent it as a signed permutation of 1, 2, ..., n with delimiters "$" inserted between the

chromosomes.

7 -2 8 3 $
5 9 -6 -1 12 $
11 4 10 $

We have written each chromosome on a separate line, and have terminated the last chromosome with the delimeter, but neither of these are necessary. Any whitespace, including line breaks, simply separates the genes and the chromosome delimeters; only the "$" actually separates chromosomes. Also, the order of the chromosomes and the direction of the chromosomes do not matter in our algorithms.

11 4 10 $
-3 -8 2 -7 $
5 9 -6 -1 12 $

If you enter only one genome, GRIMM assumes you want to do a pair wise comparison of that genome with the identity permutation. For neatness, we have written each chromosome on a separate line, and have terminated the last chromosome with the delimeter, but neither of these are necessary. Any whitespace, including line breaks, simply separates the genes and the chromosome delimeters; only the "$" actually separates chromosomes. This could also have been written in any of the following alternative ways:

- 7 -2 8 3 $ 5 9 -6 -1 12 $ 11 4 10 $
- 7 -2 8 3 $ 5 9 -6 -1 12 $ 11 4 10
- 7 -2
  8 3$ 5 9 -6
  -1 12 $ 11 4 10

This format is valid, but very sloppy; it is provided for illustrative purposes only.

Also, the order of the chromosomes and the direction of the chromosomes do not matter in our algorithms. Thus, we could represent this same genome by flipping the first chromosome (reverse the order of its entries and negate them) and then moving the last chromosome to the beginning:

11 4 10 $

-3 -8 2 -7 $

5 9 -6 -1 12 $

Using this or the original representation of the genome makes no difference in computing the rearrangement distance, or in the possible rearrangement scenarios that can theoretically occur; however, it may affect some of the arbitrary choices GRIMM makes, such as cap numbers and which rearrangement scenario is chosen for display. There can be similar subtle effects when MGR has to make arbitrary choices.

3.3 Naming Genomes and Making Comments

A genome may be named by preceding it with a line ">name", as shown below. This name will be used in the report. Comments are given as "# comment" [10]. They are ignored.

```
# Comments are indicated with a "#"
# and last till the end of the line.
>Sample name
11 4 10 $
-3 -8 2 -7 $     # another comment
5 9 -6 -1 12 $
```

3.4 Default Genome

If you enter only one genome, GRIMM assumes you want to do a pairwise comparison of that genome with the identity permutation

1 2 3 ... n

Although this makes sense for unichromosomal genomes, it does not make much sense for multichromosomal genomes. However, as there really isn't any meaningful default to use in the multichromosomal case, this default is as good as any [10].

Tip: If your genomes are long or you will be using them extensively, we suggest that you create them in a single file in an editor (or otherwise) on your own computer, and cut and paste them into the genome windows. This is an extension of the file format used by GRAPPA:

```
# useful comment about first genome
# another useful comment about it
>name of first genome
1 -4 2 $   # chromosome 1
-3 5 6     # chromosome 2
>name of second genome
5 -3 $
6 $
2 -4 1 $
```

For multiple genomes, continue this for as many genomes as required.

Instead of doing numerous cut and paste operations to manually separate the genomes into their own individual genome windows, you may cut and paste the entire file into one genome window. The number of genome windows does not have to match the number of genomes. For example, in the multiple genome form with a default view of 3 genome windows, you could still paste a dozen genomes into a single window.

3.5 Chromosome Types

The types of chromosome is described below [10]-

3.5.1 Circular

A circular chromosome has no physical start or end, or preferred direction, so the choice of which gene to read first is arbitrary. Distance=0 between any two of them in circular mode:

1 2 3, 2 3 1, 3 1 2, -3 -2 -1
-1 -3 -2, -2 -1 -3

3.5.2 Linear (Directed)

All 6 signed permutations above represent different chromosomes.

3.5.3 Linear (Undirected)

Chromosomes are not regarded as having a direction; flipping a chromosome gives equivalent genome. In multichromosomal genomes, all chromosomes are of this type, and an error message will be issued if you check off one of the other two types.

3.5.4 Signed and Unsigned Genomes

Most comparative mapping techniques determine the physical locations and relative order of genes in each chromosome, but do not determine which of two orientations each gene has. Current sequencing methods do provide the orientations. It turns out that the genome rearrangement problem (uni and multichromosomal) for unsigned permutations is NP-hard, but the same problems for signed data can be done in polynomial time. Fortunately, with many genomes currently being sequenced, it is likely that many comparative maps (corresponding to unsigned permutations) will soon be replaced by sequencing data (corresponding to signed permutations).

Existing data for which signs are not known may be entered into the programs without specifying the signs. The programs will find an optimal assignment of signs, if the data is not too complex, or will approximate it otherwise.

For example, to turn the unsigned genome

1 2 3 4 5

into the unsigned genome

1 4 3 2 5

requires one unsigned reversal. The program determines an assignment of signs in the source and destination genomes that give a signed reversal scenario requiring this same number of steps. Here, we get

1 **2 3 4** 5

$\rightarrow$

1 **-4 -3 -2** 5

which also takes one step. Note that there may be other sign assignments taking this minimum number of steps. It is possible that correctly signed data would have increased the number of steps:

1 **2 3 4** 5

$\rightarrow$

1 **-4 -3 -2** 5

$\rightarrow$

1 **-4 3 -2** 5

If the data collection method did not determine signs, it is impossible to know mathematically whether the one step or two step scenario is more biologically accurate; the mathematical problem the program solves is to find the signs giving the minimum possible distance.

3.6 Formatting Options for Pair-wise Scenarios

The formatting options for Pair-wise Scenarious are described below [8]-

3.6.1 One Line per Genome, Displayed Horizontally or Vertically

A good choice for small genomes and for seeing how the algorithm works. GRIMM concatenates all the chromosomes together in an order determined by its algorithms. This lets you see how translocations, fissions, and fusions are emulated by reversals that cross chromosome boundaries. However, since each rearrangement event involves just one or two chromosomes, this format is unwieldy for large genomes. The steps are shown in Figure-3.2.

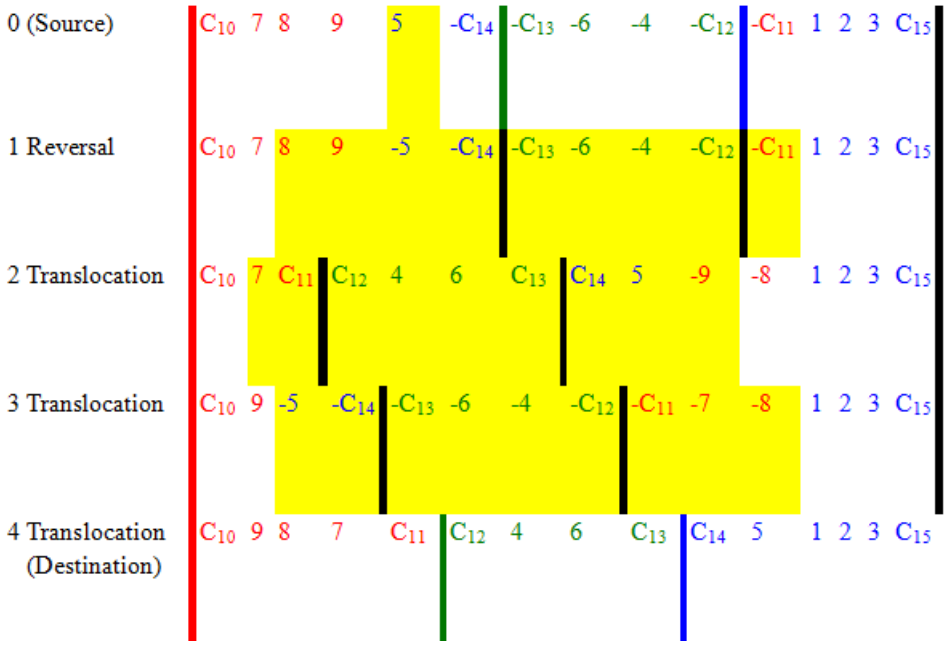


Figure 3.2: One Line Per Genome, Displayed Horizontally or Vertically

3.6.2 One Column

This is also suited to small genomes. The chromosomes are shown separately. If a chromosome is too large to fit in the width of the screen, its genes will be shown on multiple lines. Table borders (not line breaks) delineate the separate chromosomes. The steps are shown in Figure-3.3.

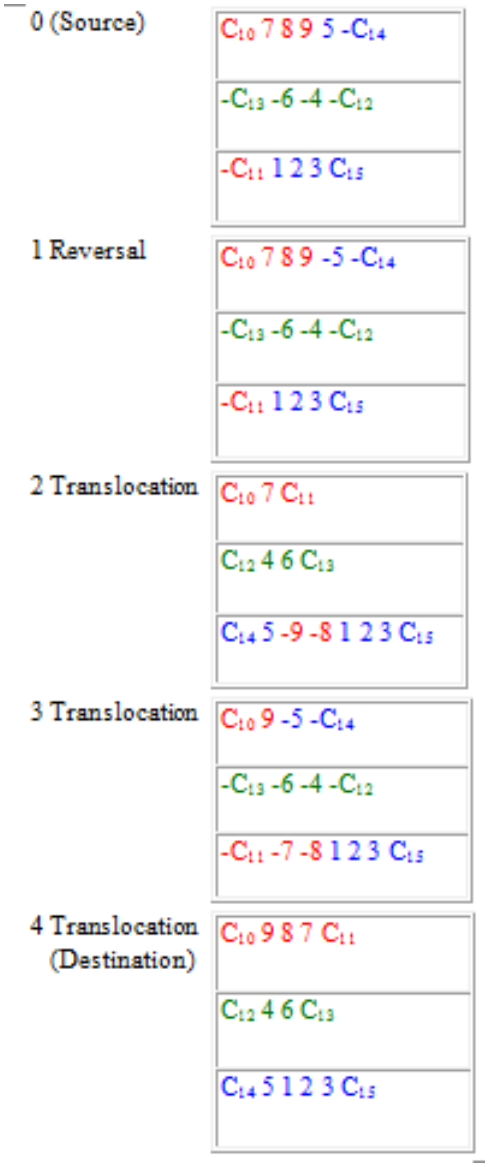


Figure 3.3: One Column

3.6.3 Two Column Before and After

This shows the chromosomes in two side-by-side tables similar to the one column format. The option "Only affected chromosomes" is the best choice for large genomes, as it shows only the one or two chromosomes affected by each rearrangement event. The steps are shown in Figure-3.4.

1 (Source)	C_{10} 7 8 9 5 $-C_{14}$	C_{10} 7 8 9 -5 $-C_{14}$
Reversal	$-C_{13}$ -6 -4 $-C_{12}$	$-C_{13}$ -6 -4 $-C_{12}$
	$-C_{11}$ 1 2 3 C_{15}	$-C_{11}$ 1 2 3 C_{15}
2 Translocation	C_{10} 7 8 9 -5 $-C_{14}$	C_{10} 7 C_{11}
	$-C_{13}$ -6 -4 $-C_{12}$	C_{12} 4 6 C_{13}
	$-C_{11}$ 1 2 3 C_{15}	C_{14} 5 -9 -8 1 2 3 C_{15}
3 Translocation	C_{10} 7 C_{11}	C_{10} 9 -5 $-C_{14}$
	C_{12} 4 6 C_{13}	$-C_{13}$ -6 -4 $-C_{12}$
	C_{14} 5 -9 -8 1 2 3 C_{15}	$-C_{11}$ -7 -8 1 2 3 C_{15}
4 Translocation (Destination)	C_{10} 9 -5 $-C_{14}$	C_{10} 9 8 7 C_{11}
	$-C_{13}$ -6 -4 $-C_{12}$	C_{12} 4 6 C_{13}
	$-C_{11}$ -7 -8 1 2 3 C_{15}	C_{14} 5 1 2 3 C_{15}

Figure 3.4: Two Columns Before and After

3.6.4 Show all Possible Initial Steps of Optimal Scenarios

Every reversal, translocation, fission, and fusion that does not create a new breakpoint is applied to the source genome. The events that would reduce the distance by 1 are displayed graphically. A summary of how many events changed the distance by -1, 0, or +1 is displayed, and the number of events in each category that were attempted until the first success (reducing distance by 1) is shown.

3.7 Operations and Representation in GRIMM

The operations are discribed below [10]-

3.7.1 Unichromosomal Operations

Symbols:

‘\’ denotes a fission reducing distance by 1.

‘:’ denotes a breakpoint where fission does not reduce distance by 1.

‘-’ denotes segments on which reversals reduce the distance by 1.

Reversal: A reversal in a signed permutation is an operation that takes an interval in a permutation, reverses the order of the numbers, and changes all their signs. For example,

5 1 3 2 -9 7 -4 6 8

$\rightarrow$

5 1 -7 9 -2 -3 -4 6 8

The reversal distance between two genomes is the minimum number of reversals it takes to get from one genome to the other. For a given pair of genomes, the reversal distance is unique, but there are usually many possible reversal scenarios with this distance. However, it is possible that this mathematical notion of reversal distance can underestimate the actual number of steps that occurred biologically.

3.7.2 Multichromosomal Operations

Symbols:

‘+’ denotes a fusion reducing distance by 1. If chromosomes A, B can be combined as A+B and B+A, it is denoted A+B+. (The other two possible fusions, in which one chromosome is flipped but not the other, are shown separately when relevant)

‘|’ separates chromosomes when fusion does not reduce the distance by 1.

We treat four elementary rearrangement events in multichromosomal genomes: reversals, translocations, fusions, and fissions.

Reversal: An interval within a single chromosome may be reversed in the same fashion as a reversal acts in the unichromosomal case:

7 -2 8 3 $
5 9 -6 -1 12 $
11 4 10 $
→
7 -2 8 3 $
5 9 -12 1 6 $
11 4 10 $

Note: When the programs are run in unichromosomal mode, the genomes 3 1 2 and -2 -1 -1 are considered different (one reversal apart, distance=1), while in multichromosomal mode, those same genomes are considered equivalent (distance=0) because we have simply flipped an entire chromosome, which gives an equivalent genome in the multichromosomal mode.

Translocation: Two chromosomes "A B" and "C D" may be rearranged into "A D" and "C B" (The letters A, B, C, D stand for sequences of genes). Because flipping chromosomes does not alter a genome (only its representation is altered), "A -C" and "-B D" is another possible translocation, and is the one actually done by our algorithm. (-B means to reverse the order of the genes in sequence B and negate each one.) For example, a translocation on chromosomes 1 and 3 is

7 -2 8 3 $
5 9 -6 -1 12 $
11 4 10 $
→
7 -2 8 -4 -11 $
5 9 -6 -1 12 $
-3 10 $

Fusion: Two chromosomes may be fused together into a single chromosome. Due to chromosome flipping, there are four distinct fusions between each pair of chromosomes. Here is one of the fusions between chromosomes 1 and 3:

7 -2 8 3 $

5 9 -6 -1 12 $

11 4 10 $

→

7 -2 8 3 -10 -4 -11 $

5 9 -6 -1 12 $

Fission: A chromosome may be broken into two chromosomes between any pair of genes.

7 -2 8 3 $

5 9 -6 -1 12 $

11 4 10 $

→

7 -2 8 3 $

5 9 $

-6 -1 12 $

11 4 10 $

3.8 Highlighting Style

The genes involved in a rearrangement event can be highlighted in a variety of ways, depending on the report style chosen: before they are rearranged; after they are rearranged; both before and after (in the two column formats); by a yellow line drawn between the lines (in the one line formats); or no highlighting. If your browser permits it, the options that do not make sense for the chosen report style will be disabled [10].

3.9 Caps (Chromosome end Markers)

Chromosome delimeters "$" or ";" are not displayed in the reports; the chromosome boundaries are rendered graphically instead as colored lines or table borders. However, most report formats do, by default, display caps. These are artificial markers created by the multichromosomal genomic distance algorithm to delimit the start and end of each chromosome. This is a necessary part of the mathematical algorithms that compute the distance and the rear-

rangement scenarios, but it is not necessary for you to see them if you do not need them [10]. If you enter the 12 gene genome

7 -2 8 3 $
5 9 -6 -1 12 $
11 4 10 $

It will initially add caps 13 and 14 to the first chromosome, 15 and 16 to the second, and 17 and 18 to the third:

13 7 -2 8 3 14
15 5 9 -6 -1 12 16
17 11 4 10 18

In the course of computing the distance and of computing rearrangement scenarios, the caps will be rearranged as well. Throughout a scenario with this genome, the numbers 1,...,12 will represent genes, and the numbers 13,...,18 will represent caps, but the numbers 13 and 14 will not necessarily continue to delimit the first chromosome, or even the same chromosome.
You may display the caps as numbers 13,...,18; highlight them as C_{13},...,C_{18} (default); or omit them all together.

3.10 Color Coding

The genes are assigned a color based on their chromosome in the source or destination genome. There are only a limited number of distinguishable web-safe colors that also contrast well with the normal and highlighting backgrounds, so if there are a lot of chromosomes, it cycles through the colors and then reuses them [10].

3.11 Pairwise or Multiple Genome Form

In a pairwise genome comparison, the rearrangement distance between two genomes is given, and an example of a specific sequence of steps achieving that distance is shown.

In a multiple genome comparison, a matrix of the pairwise distances is displayed, and optionally, a phylogenetic tree is computed by MGR.
A button at the top of the form lets you switch to the other form: “Multiple genome form” or “Pairwise genome form”.
On the multiple genome form, there is a box to adjust the number of genome windows. Enter a new number of windows and press “enter” or “update” depending on your browser. Technically this is not the number of genomes, because you may leave windows blank, and because you may enter multiple genomes in one window [10].

3.12 Multiple Genome Options

The multiple genome options are [10]-

3.12.1 Distance Matrix Only

This displays a matrix of the pairwise distances between the genomes. Clicking on an entry in the matrix will show a pairwise scenario achieving that distance. If you have updated the pairwise scenario formatting options, they will be respected, but if you have also changed the genomes (which would be inconsistent with the matrix), those changes will be ignored. This requires JavaScript be enabled.

3.12.2 Phylogenetic Tree (MGR)

This produces:

- A phylogenetic tree with the given genomes as the leaves. Clicking on an edge runs GRIMM to produce a pair wise scenario between the two genomes.
- The “Newick Standard” string representation of the tree, for input to other tree drawing software.
- The distance matrix for the input genomes.

Since MGR does not produce instant results, this web interface to MGR only permits small inputs: small numbers of genes and small numbers of genomes. It aborts after a time limit of 1 minute.

3.12.3 Tree Size

- Edge length proportional to distance and Total width of tree: The edges will be stretched out in proportion to the distances they represent, to fit horizontally within the specified number of characters
- Not proportional: The input genomes are aligned on the right.
- Reformat tree: This only appears when a tree is displayed. You may change the tree size options and hit this button to reformat the tree. It does not recompute it, so any alterations you have made to the genomes will be ignored.

3.13 Run, Undo, Clear, and Sample Data

- Run - runs the program.
- Undo - only undoes changes since your last submission. The behavior may depend on your browser. Use your browser's Back button to back up to previous inputs.
- Clear - clears the form.
- Sample data - shows a menu of demonstration data. Selecting data will automatically run the program with it, unless you have an older browser or have disabled JavaScript, in which case you will have to hit the run button.

3.14 Necessary Algorithms for GRIMM

3.14.1 MGR Algorithm

The MGR algorithm is as follows [7]-

Algorithm 2 MGR Algorithm

1: MGR Multiple Genome Rearrangement algorithm deals with more than two genomes. GRIMM can only deal with two genomes at a time.

2: Connect vertices in the anchor graph by an edge if the distance between them is smaller than the gap size G.

3: MGR creates a phylogenetic tree and can create the most likely ancestor of a set of genomes.

4: Given a set of m signed permutations (existing genomes), find a tree T with the m permutations as leaf nodes and assign permutations (ancestral genomes) to internal nodes such that D(T) is minimized, where D(T) is the sum of rearrangement distances over all edges of the tree, where d(π,γ) is the rearrangement distance between two permutations π and γ.

D(T)=$\sum_{(\pi,\gamma)\in T} d(\pi,\gamma)$

5: Consider 3 genomes: G1, G2, G3.

6: Evaluate all possible rearrangements for each genome, identifying good rearrangements.

7: A rearrangement is good in G1 if it brings it closer to the ancestral genome (closer to both G2 and G3).

8: Iterate finding good rearrangements until G1, G2, and G3 are transformed into the same genome (ancestor A).

3.14.2 GRIMM-Synteny Algorithm

The algorithm is as follows [17]

Algorithm 3 GRIMM-Synteny Algorithm

1: Form an anchor graph whose vertex set is the set of anchors.
2: Connect vertices in the anchor graph by an edge if the distance between them is smaller than the gap size G.
3: Determine the connected components of the anchor graph. Each connected component is called a cluster.
4: Delete "small" clusters (shorter than the minimum cluster size C in length).
5: Determine the cluster order and signs for each genome.
6: Output the strips in the resulting cluster order as synteny blocks.

3.14.3 Hannenhalli-Pevzner Algorithm

Algorithm 1: As long as π has an oriented pair, choose the oriented reversal that has maximal score.

Algorithm 2: If a permutation has 2k hurdles, $k \geq 2$, merge any two non-consecutive hurdles. If a permutation has 2k + 1 hurdles, $k \geq 1$, then if it has one simple hurdle, cut it; If it has none, merge two non-consecutive hurdles, or consecutive ones if k = 1.

Together with Algorithm 1, Algorithm 2 can be used to optimally sort any signed permutation. Permutations that are not already reduced can always be reduced by merging consecutive integers, and by renumbering all the elements.

Given the vectors p and s, selecting the oriented reversal with maximal score is elementary. In the above example, vertex 2 would be the selected candidate. The interesting part is how a reversal affects the structure. These effects are summarized in the following algorithm [1], which recalculates the bit-matrix v, the parity vector p, and the score vector s, following the reversal associated to vertex i, whose set of adjacent vertices is denoted by v_i.

Algorithm 4 Hannenhalli-Pevzner Algorithm

1: $s \leftarrow s + v_i$

2: $v_{ii} \leftarrow 1$

3: **for** each vertex j adjacent to i do

  if j is oriented

    $s \leftarrow s + v_j$

    $v_{jj} \leftarrow 1$

    $v_j \leftarrow v_j \oplus v_i$

    $s \leftarrow s + v_j$

  else

    $s \leftarrow s - v_j$

    $v_{jj} \leftarrow 1$

    $v_j \leftarrow v_j \oplus v_i$

    $s \leftarrow s - v_j$

4: $p \leftarrow p \oplus v_i$

The logic behind the algorithm is the following. Since vertex i will become unoriented and isolated, each vertex adjacent to I will automatically gain a point of score in step 1. Next, if j is a vertex adjacent to i, vertices adjacent to j after the reversal are either existing vertices that were adjacent to j and not adjacent to i, or vertices that were adjacent to i but not to j. This is the definition of the exclusive-or operator $\oplus$.

3.15 GRIMM Synteny Block

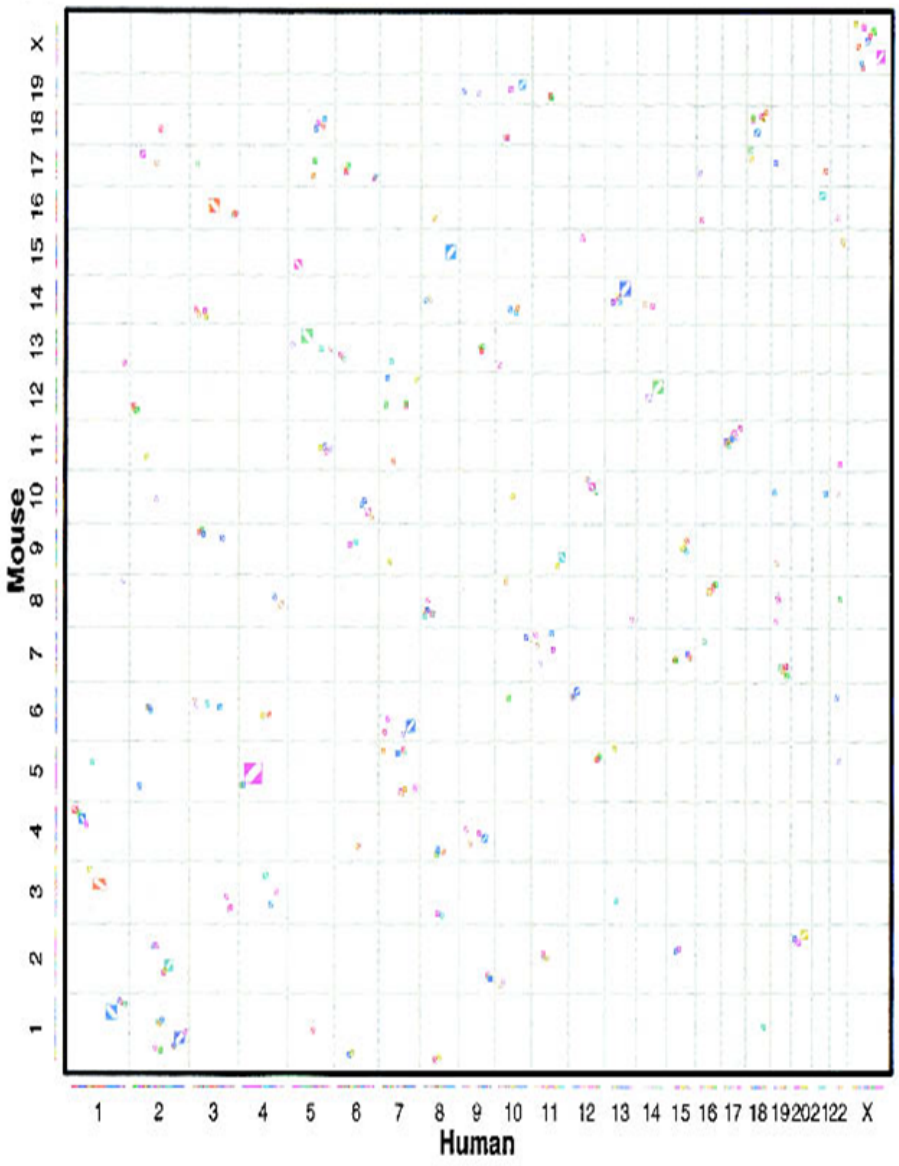


Figure 3.5: GRIMM Synteny Block

3.16 GRIMM After Clustering

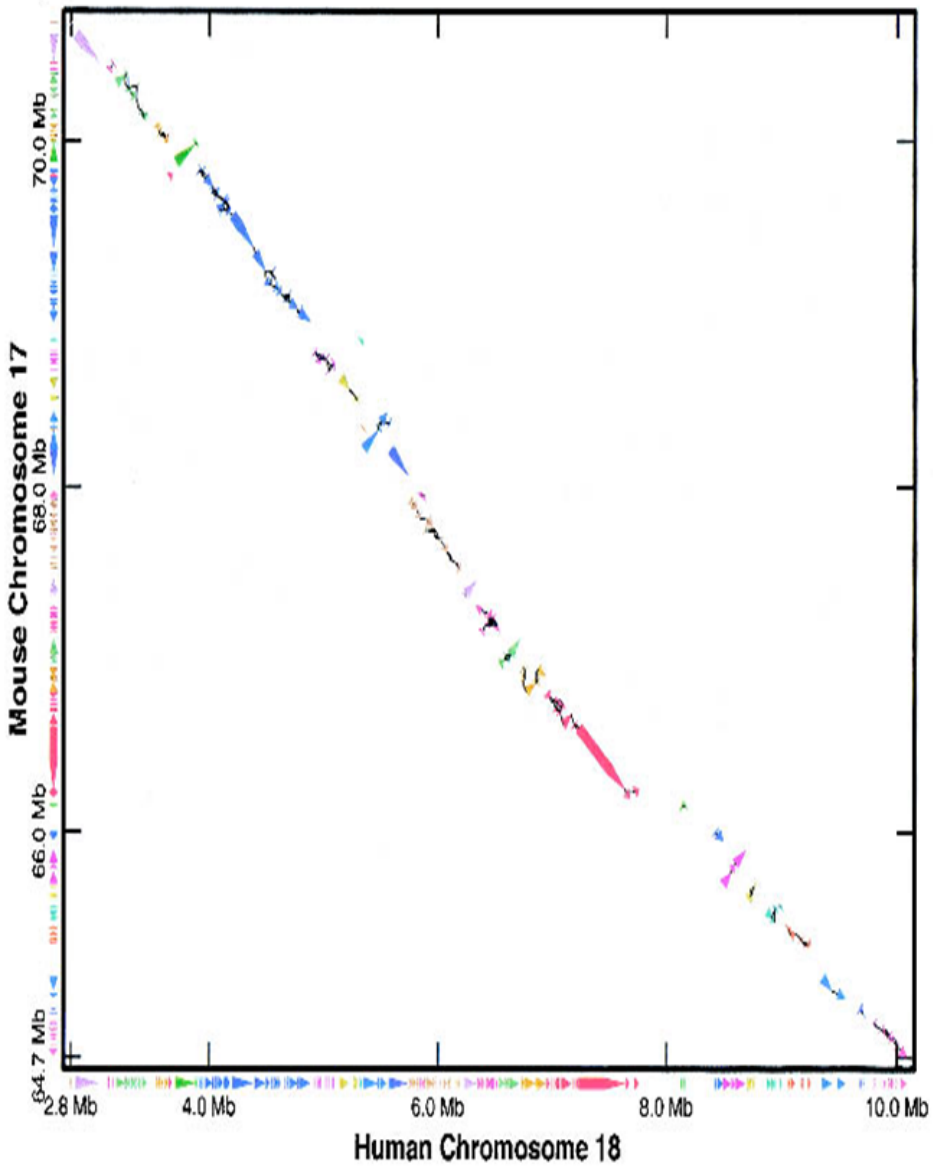


Figure 3.6: GRIMM After Clustering

CHAPTER 4

OUR RESEARCH ON GRIMM

We treat four elementary rearrangement events in multichromosomal genomes: reversals, translocations, fusions, and fissions. In GRIMM, there is mainly one unichromosomal operation which is reversal. In our research, we are going to add transposition additionally in GRIMM for unsigned unichromosomal genomes. There has been significantly less progress on the problem of sorting by transpositions. The complexity of sorting by transpositions was open. But it is proved that, it is np-hard [1]. In our research, we use the 1.375-Approximation Algorithm for Sorting by Transpositions by Elias and Hartman [16].

4.1 The1.375-Approximation Algorithm by Elias and Hartman

The algorithm is based on a new upper bound on the diameter of 3-permutations. In addition, there are some new results regarding the transposition diameter: the lower bound for the transposition diameter of the symmetric group is improved and determine the exact transposition diameter of simple permutations. A sequence of transpositions sorts an interval separately if they sort the interval and are applied only on black edges from the interval. Permutations can be sorted by sorting each interval separately or by applying transpositions that mix the intervals, i.e., moves that are applied on black edges from different intervals. Intuitively, the algorithm sorts the permutation by repeatedly applying (11, 8)-sequences and, since 11/8 = 1:375, we get the desired approximation ratio. The algorithm is as follows [16]-

Algorithm 5 Elias and Hartman: Algorithmic Sort(π)

1: Transform permutation π into a simple permutatiom $\hat{\pi}$

2: Check if there is a a (2,2)-sequence. If so, apply it.

3: **While** $G(\hat{\pi})$ contains a 2-cycle, apply a 2-move.

4: $\hat{\pi}$ is a 3-permutatiom. Mark all 3-cycle in $G(\hat{\pi})$

5: **While** $G(\hat{\pi})$ contains a 3-cycle C, do

    If C is oriented, apply a 2-move on it.

    Otherwise, try to sufficiently extend C eight times.

        If a big configuration is reached (whick is a sufficient configuration by definition), apply a 11/8-sequence (Lemma 16).

        Otherwise, it is a small component. If an 11/8-sequence is possible, apply it. Otherwise, unmask all cycle of the component (this is a bad small component).

6: $G(\hat{\pi})$ contains only bad small components. While $G(\hat{\pi})$ contains at least 8 cycle, apply an 11/8-sequence (Lemma 17).

7: While $G(\hat{\pi})$ contains a 3-cycle, apply a (3,2)-sequence (Lemma 7).

8: Mimic the sorting of π using the sorting of $\hat{\pi}$

4.2 Double Cut and Joint (DCJ)

We implement this algorithm in GRIMM using Double Cut and Joint method. The double cut and join (DCJ) is an abstract rearrangement operation, which allows to represent most large scale mutation events, such as inversions, translocations, fusions and fissions, which can occur in genomes. If no restriction on the genome structure considering linear and/or circular chromosomes is imposed, using a simple graph data structure, the adjacency graph, this leads to considerable algorithmic simplifications. For example, the inversion distance problem can be tackled via the DCJ model in linear time.

4.3 GRIMM Results

In GRIMM, reversal is the only unichromosomal operation. We consider a permutation π = 1 5 6 2 3 4 7 as source, the identity permutation= 1 2 3 4 5 6 7 as the destination. The output is shown below:

```
raien@ubuntu:~/Desktop/th/GRIMM-2.01$ ./grimm -f data/gollan12.txt -L -d
Running:                        ./grimm
Input:                          data/gollan12.txt
Genome:                         Linear
Signs:                          Signed
Number of Genomes:              2
Number of Genes:                7
Number of Chromosomes:          0 (unichromosomal)
Reversal Distance:              3
raien@ubuntu:~/Desktop/th/GRIMM-2.01$
```

Figure 4.1: The output of GRIMM for unichromosomal genome π

Here, we see that, the reversal distance between the source and destination genome is 3. We will going to show that, by transposition the distance will be same or minimized than the reversal distance.

4.4 Implementation of Transposition

According to the 1.375 approximation transposition algorithm, we have to make the source permutation into an equivalent simple permutation. For this purpose we have to make the source permutation (π) into an extended π and draw a breakpoint graph of π.

π = 1 5 6 2 3 4 7

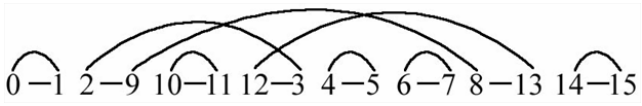


Figure 4.2: The source permutation π and it's equivalent breakpoint graph.

We represent the reality-desire diagram as a linked list of 2n + 2 nodes. The data structure node for each node v consists of the three pointers reality (pointing to the node connected

with v by a reality edge), desire (pointing to the node connected with v by a desire edge), and co element (pointing to the co-element of v), and the two variables position (the position w.r.t. the leftmost node in the diagram), and cycle (the index j of cycle Cj the node belongs to). We can initialize this data structure for every permutation in linear time. First, the initialization of reality, co element, and position can be done with a scan through the permutation. Second, for the initialization of desire we need the inverse permutation (mapping the nodes ordered by their label to their position) which can also be generated in linear time. Finally, we can initialize cycle by following the reality and desire edges which also takes linear time.

Step 1

We now tackle the problem of transforming a permutation into an equiva- lent simple permutation in linear time. The algorithm has two processing phases.

Phase 1:

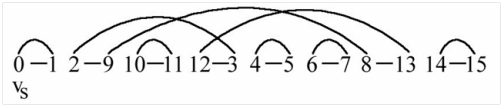
Our goal in the first phase is to create short cycles or cycles that have no interleaving desire edges. We achieve this goal with a scanline algorithm. The algorithm requires two additional arrays: left[j] stores the leftmost node of each cycle C_j and next[j] stores the right node of the desire edge we are currently checking for interleaving. In both arrays, all variables are initialized with UNDEF. In the following, vs denotes the current position of the scanline. Before we describe the algorithm, we will first provide an invariant for the scanline.

Invariant: If g_{ij} is a desire edge of the long cycle C_j with $i < l_j$, and both nodes of g_i lie to the left of vs, then g_i does not intersect with any other desire edge of C_j .

It is clear that a cycle C_j has no interleaving edges if the invariant holds and the scanline passed the rightmost node of C_j : g_{lj} does also not interleave with a desire edge of C_j because the interleaving relation is symmetric. As v_s is initialized with the leftmost node of RD(π), the invariant holds in the beginning. While the scanline has not reached the right end of the diagram, we repeat to analyze the following cases:

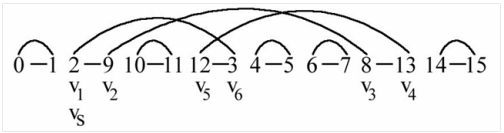
- **Case 1.1:** v_s is part of a short cycle. We move the scanline to the left node of the next reality edge. As the invariant only considers long cycles, the invariant is certainly preserved.

  Initially, v_s is at the leftmost node of the graph which is part of a short cycle (size=1).



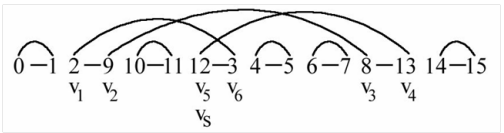
Therefore, we move vs to the left node of the next reality edge.

- **Case 1.2:** v_s is part of a long cycle C_j and next[j] = UNDEF. That is, v_s is the leftmost node of cycle C_j . So we set left[j]= v_s. To check whether $g_1 = (v_2, v_3)$ interleaves with another desire edge, we store the right node of g_1 in next[j] and move v_s to the left node of the next reality edge. Both nodes passed by the scanline (i.e. v_1 and v_2) are the left nodes of a desire edge, so the set of desire edges that lie completely to the left of v_s is not changed and the invariant is preserved.



Here, v_s is at v_1 which is part of a long cycle C_j where j=1. Initially, next[1] =UNDEF. Therefore, left[1] = $v_s = v_1 = 2$, $g_1 = (v_2, v_3)$ and next[1] = $v_3 = 8$.

- **Case 1.3:** v_s is part of a long cycle C_j and next[j] $\neq v_s$. Let next[j] be the node v_{2k+1}, i.e. we check for a desire edge that interleaves with gk (going from node v_{2k} to node v_{2k+1}). As pos(v_1) < pos(v_{2k}) < pos(v_s) < pos(v_{2k+1}), there must be a desire edge gm belonging to C_j that interleaves with g_k.



Here, $v_s = v_5$ which is part of a long cycle C1 and next[1] $\neq v_s = v_5$.
We now distinguish three cases:

  - g_k is not g_1. We perform a (b, g)-split with b = b_{k+1} and g = g_{k-1}. That is, we split the 2-cycle (v_{2k}, v_{2k+1}, x, v_{2k-1}) from C_j. This split is save since gk now lies in the 2-cycle that still interleaves with gm, which belongs to C_j . The right node of the new g_{k-1} in C_j is y, so we adjust next[j] to y.

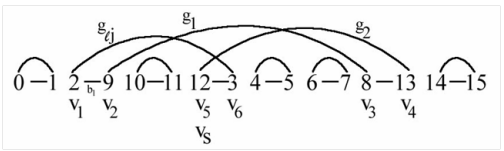
– g_k is g_1 and g_k interleaves with g_{lj}. We perform a (b, g)-split with b = b1 and g = g_2. That is, we split the 2-cycle (v_2, v_3, v_4, y) from C_j. This split is save since g_1 now lies in the 2-cycle that still interleaves with g_{lj}, which belongs to C_j . Now, g_1 = (x, v_5), so we set next[j]=v_5. Note that v5 cannot be to the left of v_s, as v_s is the leftmost node that belongs to C_j and has an index ≥ 4.

– g_k is g_1 and g_k does not interleave with g_{lj}. It follows that gm $\neq g_{lj}$. We perform a (b, g)-split with b = b_2 and g = g_{lj} . That is, we split the 2-cycle (v_2, v_3, x, v_1) from C_j. This split is save since g_1 now lies in the 2-cycle that still interleaves with g_m. As the old leftmost node and reality edge of C_j lie in the 2-cycle we set next[j] = UNDEF which forces the re-initialization of left[j] with v_s and next[j].

In all of these cases, we do not create a desire edge that lies completely to the left of v_s, so the invariant is preserved.

- **Case 1.4:** v_s is part of a long cycle C_j and next[j] = v_s. That is, we reach the right node of a desire edge g_k. It follows that g_k does not interleave which any other desire edge of C_j since we have not detected a node of C_j between the left and right node of g_k. Thus moving v_s to the right preserves the invariant. The next desire edge to check is g_{k+1} = ($v_{2(k+1)}$, $v_{2(k+1)+1}$), so we set next[j] to the right node of g_{k+1} and move vg_s to the left node of the next reality edge.

Here, g_k is g_1 and it interleaves with g_{lj}. Therefore we have to do a (b, g)-split where b=b1 and g = g_2.

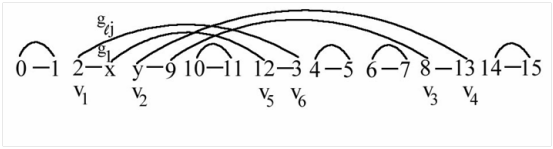


Given a reality edge b = (v_{b1}, v_{b2}) and a desire edge g = (v_{g1}, v_{g2}), a (b, g)-split can be performed in constant time, if we disregard the problem that we have to update the position variables of the new nodes and all the nodes that lie to the right of b. Fortunately, we need position only to determine if two edges of the same cycle interleave, thus it is sufficient if the relative positions of the nodes of each cycle are correct. This information can be maintained if we set the positions of the new nodes x and y to the positions of the old nodes of b which are now non- incident to x or y. After performing all splits, the reality-desire diagram can easily be transformed into the simple permutation by following desire edges and co-element pointers.

Algorithm 6 (b,g)-split

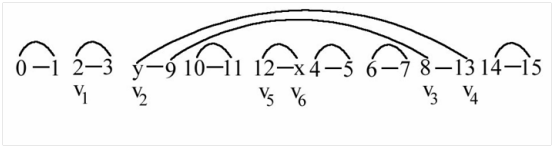
1: function bg-split(b = (vb_1, vb_2), g = (vg_1, vg_2)).
2: create new nodes x, y.
3: vb_1.reality = x; vb_2.reality = y adjust reality and desire edges.
4: x.reality = vb_1; y.reality = vb_2.
5: vg_1.desire = x; vg_2.desire = y.
6: x.desire = vg_1; y.desire = vg_2.
7: x.position = vb_2.position; y.position = vb_1.
8: return(x, y).

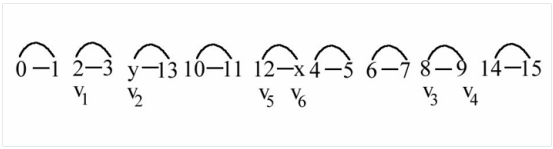
After applying (b, g) split, the breakpoint graph will be as follows:



Phase 2: After phase 1 we can assure that there remain only short cycles and long cycles with pairwise non-interleaving desire edges. These long cycles have a special structure. But, the breakpoint graph contains only short cycles. Therefore, we can skip the Phase 2.

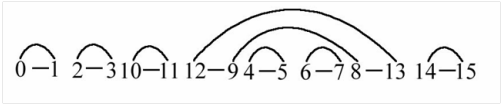
Step 2: Now, we have to find a (2, 2)-sequence and if there exists one, we have to apply it. As there are two intersecting 2-cycles, there exists a (2, 2)-sequence. After we apply a (2, 2)-sequence the breakpoint graph will be as follow:



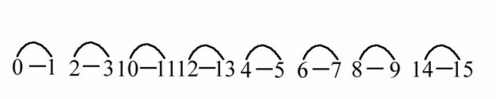


Here, after implementing the two 2-move the breakpoint graph only consists of 1-cycle. Therefore we can skip the other steps of Algorithm sort and apply the last step to mimic the sorting of simple permutation to sort the source permutation π.

Therefore, we have to apply one (2, 2)-sequence on π.



Now, if we just apply a 0-move then, Here in every 2-move, we have to break and join



2 edges (one black and one grey). Therefore, we have one DCJ at each 2-move. The 0-move doesn't break any edges. Therefore the overall distance for converting the source permutation to the identity permutation is 2 which is less than the corresponding reversal distance.

4.5 Our Contribution

- We have included our code in GRIMM and compared the reversal distance with the transpositional distance between a permutation and the identity permutation. The output of GRIMM after our modification is shown below.

```
raien@ubuntu:~$ cd Desktop/th/GRIMM-2-real.01
raien@ubuntu:~/Desktop/th/GRIMM-2-real.01$ ./grimm -f data/gollan12.txt -L -d
Running:                        ./grimm
Input:                          data/gollan12.txt
Genome:                         Linear
Signs:                          Signed
Number of Genomes:              2
Number of Genes:                7
Number of Chromosomes:          0 (unichromosomal)
Reversal Distance(GRIMM's):             3

Transpositional distance (our result):2
for permutation: 1 5 6 2 3 4 7
```

Figure 4.3: GRIMM output after our modification

- We compare the the reversal and transpositional distance for various permutations and provide a graphical presentation of the comparison where we consider no of permutations on the X-axis and equivalent distance on the Y-axis.

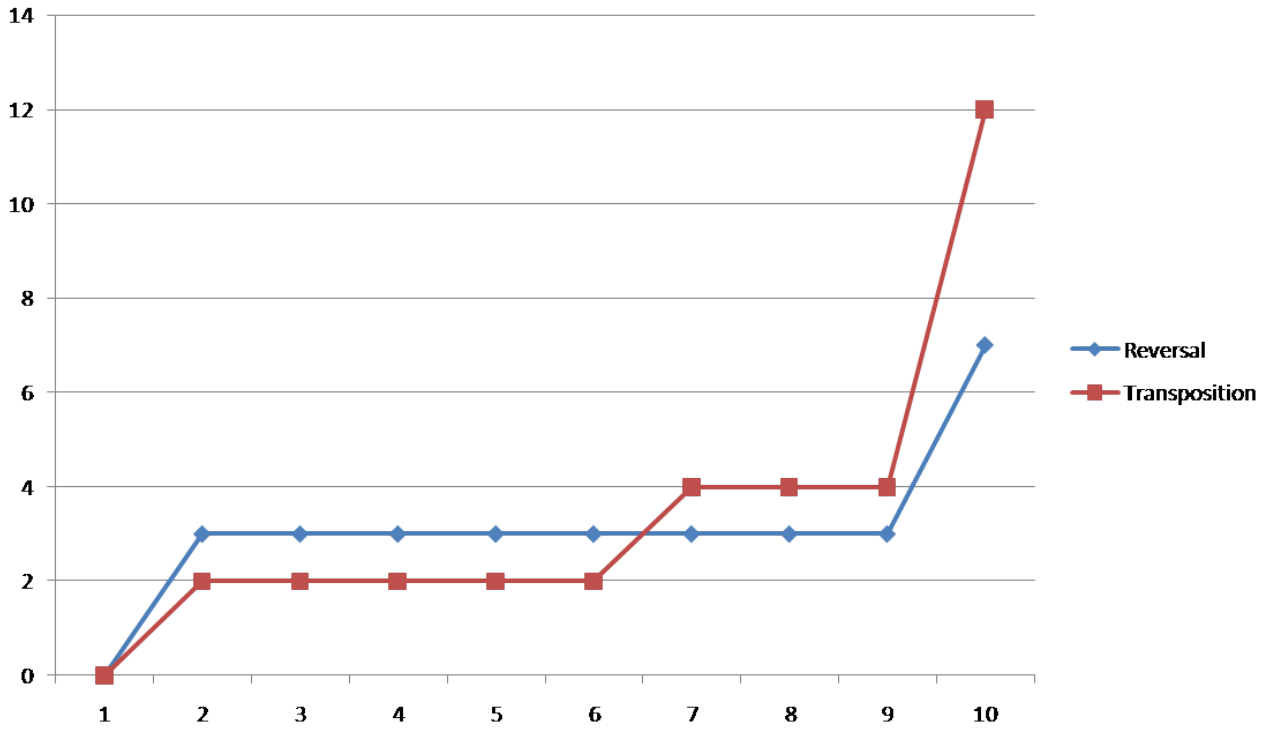


Figure 4.4: GRIMM output after our modification

From the graph we can see that, for many cases, the transpositional distance gives better result than the reversal distance. Therefore, if the right operation can be selected for the right types of permutations, the distance between two permutation will be either equal or less than the previous distance.

- In the 1.375 approximation algorithm [16], short cycle is defined by cycles of size k (k $\leq$ 3). But in the simple permutation algorithm [11], short cycle is defined by cycles of size k (k $\leq$ 2). Therefore when we convert a source permutation into a simple one, it is ensured that there is no cycle of size 3. Therefore, all the cases in the algorithm [16] for finding 3-cycles and eliminating them are not necessary in our implementation. Hence, merging these two algorithms reduces the no of steps to implement the 1.375 approximation algorithm [16].

CHAPTER 5

CONCLUSION

In this paper, we try to include sorting by transposition in GRIMM mainly for unsigned linear unichromosomal genomes. As transposition is an intrachromosomal sorting operation, it mainly is used in unichrosomal genome sorting. But the problem of sorting linear permutations by transpositions is linearly equivalent to the problem of sorting circular permutations by transpositions. Therefore, further modification can be made on GRIMM to improve the transposition algorithm for signed circular unichromosomal genomes. Though we give result regarding the transposition distance between two unsigned linear unichromosomal genomes, there is no comparison result regarding the performance of reversal AND transposition.

REFERENCES

[1] B. Anne. "A Very Elementary Presentation of the HannenhalliPevzner Theory". *Discrete Applied Mathematics 146*, page 134145, 2005.

[2] F. Guillaume B. Laurent and R. Irena. "Sorting by transpositions is difficult". *SIAM J. Discrete Math., 26(3)*, page 11481180.

[3] A. Caprara. "Sorting Permutations by Reversals and Eulerian Cycle Decompositions". *SIAM Journal on Discrete Mathematics 12*, page 91110, 1999.

[4] D.A. Christie. "Genome Rearrangement Problems". *PhD thesis, University of Glasgow*, 1999.

[5] T. Glenn. GRIMM.

[6] T. Glenn. GRIMM Download Link is . `http://grimm.ucsd.edu/DIST/`, 2001. [Online; accessed 04-December-2013].

[7] T. Glenn. "Efficient algorithms for multichromosomal genome rearrangements". *Journal of Computer and System Sciences, vol. 65, Issue 3*, pages 587–609, 2002.

[8] T. Glenn. "Genome Rearrangements Web Server". *Bioinformatics vol. 18, Issue 3*, pages 492–493, March 2002.

[9] T. Glenn and Y. Yang. GRIMM. `http://grimm.ucsd.edu/GRIMM/grimm_instr.html/`, 2001. [Online; accessed 04-December-2013].

[10] T. Glenn and Y. Yang. GRIMM. `http://grimm.ucsd.edu/GRIMM/index.html/`, 2001. [Online; accessed 04-December-2013].

[11] S. Gog and M. Bader. "How to achieve an equivalent simple permutation in linear time".

[12] S. Hannenhalli and P. A. Pevzner. "Transforming men into mice (polynomial algorithm for genomic distance problem)". *W.I. Milwaukee (Ed.), 36th Annual Symposium on Foundations of Computer Science, IEEE Computer Soc. Press, Los Alamitos, CA*, page 581592, 1995.

[13] S. Hannenhalli and P. A. Pevzner. “Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals”. *Journal of the ACM 46*, pages 1–27, 1999.

[14] T. Hartman. “A simpler 1.5-approximation algorithm for sorting by transpositions”. *14th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 156 – 169, 2003.

[15] T. Hartman and R. Shamir. “A Simpler and Faster 1.5-Approximation Algorithm for Sorting by Transpositions”. *Information and Computation, vol. 204, Issue 2*, pages 275–290, February 2006.

[16] E. Isaac and H. Tzvika. “1.375-Approximation Algorithm for Sorting by Transpositions”. *IEEE/ACM Transaction on Computational Biology and Bioinformatics, Vol. 3, Issue 4*, October - December 2006.

[17] T. Glenn P. Pavel. “Genome Rearrangements in Mammalian Evolution: Lessons from Human and Mouse Genomes”. pages 37–45, 2003.

[18] K. Pranav and G. Sahoo. “Survey On Bioinformatics And Computational Biology”. *International Journal of Engineering Research and Technology (IJERT), ISSN: 2278-0181, Vol. 2*, July 2013.

[19] B. Vineet and P. A. Pevzner. “Sorting by Reversals: Genome Rearrangements in Plant Organelles and Evolutionary History of X Chromosome”. *Molecular Biology and Evolution, vol. 12*, pages 239–246, 1995.

[20] B. Vineet and P. A. Pevzner. “Sorting by Transpositions”. *SIAM Journal on Discrete Mathematics 11*, page 224240, 1998.