REVERSAL DISTANCE PROBLEM

Adithya Krishna, Adithya S Nair, Anoop Boby Manuel, Athul Gireesh, Navneeth Krishna

Department of ComputerScience and Engineering

Adithya Krishna (AM.EN.U4AIE21005)

Adithya S Nair (AM.EN.U4AIE21006)

Anoop Boby Manuel(AM.EN.U4AIE21015)

Athul Gireesh (AM.EN.U4AIE21020)

Navneeth Krishna(AM.EN.U4AIE21047)

***Abstract*— Genome rearrangement is a research area capturing wide attention in molecular biology.One of the most researched models of genome rearrangements for determining the evolutionary relationship between two genomes at chromosome level is the reversal distance problem.The problem of estimating reversal distance between two genomes is modeled as sorting by reversals.There have been numerous attempts to solve this problem, but most have been approximation algorithms. These are far quicker operations and can approximate quite well, but they are still approximations.This work introduces an exact greedy algorithm for sorting by reversals**

***Keywords*— Include at least 5 keywords or phrases**

I. Introduction

Gene ordering is changed by

a genome rearrangement event and the genomic architecture of a species is altered by a series of genome rearrangements . While the gene level evolution is measured by local mutations such as insertions, substitutions, and deletions, genetic evolution at the chromosome level is based on global rearrangement events.Reversals and translocations are the most common rearrangement events observed in mammalian genetic evolution. A reversal event changes the order of genes acting on the same chromosome while a translocation rearranges the genes swapping segments between two chromosomes.

Genome rearrangement by reversals alone has been considered a worthwhile study to understand evolutionary distance of different species at the chromosome level.Genome rearrangement by reversals provides a good method of studying evolutionary history between two species.

There are a lot of approximation algorithams but they are still approximations. Here our work introduces an exact greedy algorithm for sorting by reversals.

II. Methodology

A. Defining the Reversal Distance Problem

The reversal distance problem is defined as such: Given two sequences, a and b, find the minimum number of reversals that will transform a such that it equals b

Here's an example, using the numbers 1 to 9:

a0 = 3, 10, 8, 2, 5, 4, 7, 1, 6, 9

b = 5, 2, 3, 1, 7, 4, 10, 8, 6, 9

The problem is represented as the reordering of number sequences because this is an easier way to represent the genome than with gene names or raw DNA strings. Each number represents a specific gene, with the same number representing the same gene in both sequences.

A reversal is an operation that takes any subsequence in sequence a, reverses the subsequence's order, then reinserts the subsequence back into the same position.

In our example above:

Note the zero. We begin with a0 and will increment the variable name with every reversal.

a1 = 8, 10, 3, 2, 5, 4, 7, 1, 6, 9

a2 = 5, 2, 3, 10, 8, 4, 7, 1, 6, 9

a3 = 5, 2, 3, 8, 10, 4, 7, 1, 6, 9

a4 = 5, 2, 3, 1, 7, 4, 10, 8, 6, 9 = b

Since a4 = b, the reversal distance between a and b is 4.

B. How the Reversal Distance Algorithm Works

The reason it is called an "exact" algorithm should be obvious, but it does distinguish it from the approximation algorithms . The "greedy" part refers to the fact that the algorithm takes a breadth-first approach, accumulating all possible paths until it has ruled them out. This differs from a depth-first approach that follows each path individually until the end.

C. Key Concepts

· The first concept you need to understand is the problem.
· The second concept is the one we have explained — reversals.
· you need to know about breakpoints.

Breakpoints are points in the sequence where the adjacency does not match. For instance:
a = 1, 2…4, 3…5, 6, 7, 8, 9, 10
b = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

In sequence a, the breakpoint is the one that is marked with ellipsis. In the target sequence, 2 is not adjacent to 4, so there is a breakpoint between them. Likewise, 3 is not adjacent to 5 in b, so there is a breakpoint between 3 and 5 in a.
Breakpoints can also appear at the beginning and end of sequence
· you need to know that a reversal performed on a sequence cannot remove more than 2 breakpoints, though it can remove fewer.

D. Algorithm

The greedy strategy can be applied for an optimal solution for sorting by reversals. An exact algorithm using the greedy approach is proposed in this section.
Since sorting by reversals is a combinatorial problem, there are ($\frac{n}{2}$) reversals to consider. The exact greedy algorithm finds an optimal solution by choosing a reversal which eliminates breakpoints by the largest number in each reversal step. For the algorithm all possible reversals on two breakpoints in a permutation π are observed. In other words, it considers ($\frac{b}{2}$) reversals for an initial reversal, where b is b(π) for a
permutation π. A reversal r on two breakpoints can decrease b(π) by at most 2. The reversal does not increase b(π) since it affects only the two breakpoint positions in the current permutation π. In other words, any reversal that results in the following can be a candidate.

b(π) - b(π wr) c {0, 1, 2}

The algorithm chooses all reversals which result in the same b(π), the lowest value of b(π), after every reversal step. The result permutations are kept as candidates for the next reversal step. The same greedy strategy is continued until a result permutation with no breakpoints is achieved. A sequence of reversals that results in the identity permutation offers an optimal solution for sorting by reversals. The exact greedy algorithm is motivated by the fact that the largest decrease of b(π) in each reversal step drives b(π) to zero in the minimal steps.

Let fb(π, ρai), fb(π, ρbi), and fb(π, ρci) be the decreases of b(π) after i-th reversal ρa , ρb, and ρc respectively, where 1 [i [t for the reversal distance t. Let the reversal ρa be a sequence of reversals that decreases b(π) by the largest number in each reversal step. Let the reversal ρb and ρc be two arbitrary reversals that decrease b(π) but not necessarily by the largest number.

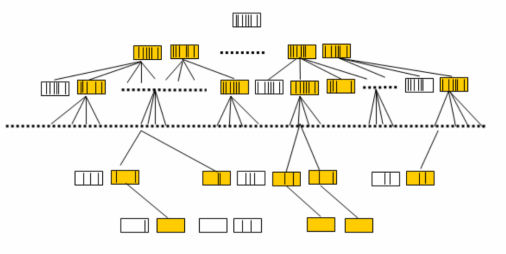
In other words,
fb(π, ρa1) m fb(π, ρb1) m fb(π, ρc1),
fb(π, ρa2) m fb(π, ρb2) m fb(π, ρc2),
fb(π, ρa3) m fb(π, ρb3) m fb(π, ρc3),
……..
fb(π, ρat) m fb(π, ρbt) m fb(π, ρct),
then,
fb(π, ρa1) + fb(π, ρa2) + …. + fb(π, ρat) m
fb(π, ρb1) + fb(π, ρb2) + …. + fb(π, ρbt) m
fb(π, ρc1) + fb(π, ρc2) + …. + fb(π, ρct).

When b(π) = fb(π, ρa1) + fb(π, ρa2) + …. + fb(π, ρat), the number of reversals t through the reversals of ρa guarantees an optimal reversal distance while either the reversals ρb and ρb do not guarantee the minimal t. When b(π) = fb(π, ρb1) + fb(π, ρb2) + …. + fb(π, ρbt) or b(π) = fb(π, ρc1) + fb(π, ρc2) + …. + fb(π, ρct), we might get a reversal distance which is smaller than t through the reversals of ρa. Thus, the optimal reversal distance d(π) of a permutation π can be achieved by choosing a reversal which decreases b(π) by the largest number in every reversal step. Algorithm 5 shows the exact greedy algorithm.

SORTING BY REVERSAL SEXACT(π)
1 takes all possible reversals ρι on two breakpoints
2 ρι = reversals that result the smallest b(π)
3 πι = π w ρι
4 while b(πι) > 0
5 ρι = reversals that result the smallest b(πι)
6 πι = πι w ρι Algorithm 5. Exact greedy algorithm

The visual explanation of Algorithm 5 is shown in Figure 10. In Figure 10 the square block on the top represents an initial permutation π and the lines inside the block show breakpoints in π. Square blocks after the top block represent permutations resulting from possible reversals from a previous step. Colored square blocks are shown as the result of permutations chosen by the greedy strategy. Colored square blocks with no lines at the bottom show the identity permutations, which can

be more than one solutions for the permutation. Any reversal path generates the identity permutation at the bottom offers an optimal reversal distance d(π)



Even though the greedy strategy achieves an optimal solution, the exact algorithm requires keeping intermediate result permutations to determine the next best reversal. The computational resource required by the algorithm increases as it approaches to middle steps and decreases towards the end of the reversals.

E. Implementation of the Reversal Distance Algorithm

1) Reversal Function : This function will take a sequence and two indices, perform a reversal and return the resulting sequence.

```
def performReversal(sequence, start_index, end_index):
    prefix = sequence[:start_index]
    reversed_subsequence = sequence[start_index:end_index][::-1]
    suffix = sequence[end_index:]
    return prefix + reversed_subsequence + suffix
```

2) Breakpoint Function : This function takes in a sequence and a target sequence and returns a list of breakpoints in the first sequence.

```
def findBreakpoints(sequence, target_sequence):
    breakpoints = []
    for index in range(len(sequence)-1):
        current_element = sequence[index]
        adjacent_element = sequence[index+1]
        if abs(target_sequence.index(current_element) - target_sequence.index(adjacent_element)) != 1:
            breakpoints.append(index+1)
    return breakpoints
```

```
def findMinimumBreakpointReversals(sequences, target_sequence):
    reversals = []
    for sequence in sequences:
        breakpoints = findBreakpoints(sequence, target_sequence)
        for start_index_i in range(len(breakpoints)-1):
            for end_index_i in range(start_index_i+1, len(breakpoints)):
                reversals.append(performReversal(sequence, breakpoints[start_index_i], breakpoints[end_index_i]))
    min_bp = len(target_sequence)
    minimum_reversals = []
    for reversal in reversals:
        num_breakpoints = len(findBreakpoints(reversal, target_sequence))
        if num_breakpoints < min_bp:
            min_bp = num_breakpoints
            minimum_reversals = [reversal]
        elif num_breakpoints == min_bp:
            minimum_reversals.append(reversal)
    return minimum_reversals
```

4) *Driver function :* The driver function which takes the original sequence and target sequence and returns the reversal distance.Basically driver function keeps track of the number of reversals and performs a simple check to see if the current sequence equals the target sequence.

```
def getReversalDistance(sequence, target_sequence):
    sequence = ["-"] + sequence + ["+"]
    target_sequence = ["-"] + target_sequence + ["+"]
    reversals = 0
    current_sequences = [sequence]
    while target_sequence not in current_sequences:
        current_sequences = findMinimumBreakpointReversals(current_sequences, target_sequence)
        reversals += 1
    return reversalsrsals
```

3) *Minimum Breakpoint Reversal Function :* This function takes a list of sequences and the target sequence and returns a list of sequences with the property that they have the minimum number of breakpoints possible after performing one reversal on each of the passed sequences

III. Conclusions

Genome rearrangement by reversals has served an important role in understanding evolutionary history between two species in terms of reversal distance. The sorting by reversals problem is modeled to find a series of reversals that converts one genome into another

This paper introduced a new exact greedy algorithm. The exact algorithm offers an optimal reversal distance, however, it requires significant computational time and storage for the solution compared to approximation algorithms when the number of breakpoints is large

Acknowledgment

We wish to show our appreciation to Dr.Manjusha Nair M. and Dr Aswathy R Nair who guided us throughout this project.We would like to thank our friends and family for providing us with deep insights in this study.

We would also like to extend our special thanks to Amrita university and our Chancellor Amma in providing us with her grace and presence in guiding us through this study.

References

[1] https://scholarworks.sjsu.edu/cgi/viewcontent.cgi?article=1103&context=etd_projects
[2] https://medium.com/@matthewwestmk/calculating-reversal-distance-using-parks-exact-greedy-algorithm-87c62d690eef
[3] Python for Bioinformatics By Sebastian Bassi
[4] Greedy algorithm - Wikipedia.
[5] Hidden_Messages.pdf (cmu.edu)