APPLICATIONS OF HEURISTIC SEARCH ON PHYLOGENY RECONSTRUCTION PROBLEMS

by
Süha Orhun Mutluergil

Submitted to the Graduate School of Engineering and Natural Sciences in partial fulfillment of the requirements for the degree of Master of Science

Sabanci University
August 7, 2012

APPLICATIONS OF HEURISTIC SEARCH ON PHYLOGENY RECONSTRUCTION PROBLEMS

Approved by:

Esra Erdem 

Hakan Erdoğan 

Albert Kohen Erkip 

Berrin Yanıkoğlu 

Hüsnü Yenigün 

Date of approval: 07.08.2012

Abstract

Phylogenies or evolutionary trees for a given family of species show the evolutionary relationships between these species. The leaves denote the given species, the internal nodes denote their common ancestors and the edges denote the genetic relationships. Species can be identified by their whole genomes and the evolutionary relations between species can be measured by the number of rearrangement events (i.e. mutations) that transform one genome into another. One approach to infer phylogeny from genomic data is by solving median genome problems for three genomes, or the genome rearrangement problem for pairs of genomes, while trying to minimize the total evolutionary distance among the given species.

In this thesis, we have developed and implemented two search based algorithms for phylogeny reconstruction problem based on solving median genome problems for circular genomes of the same length without gene duplication.

In order to show applicability and effectiveness of our algorithms, we have tested them with randomly generated instances and two real data sets: mitochondrial genomes of *Metazoa* and chloroplast genomes of *Campanulaceae*.

Özet

Verilen bir tür ailesi için oluşturulan filojeniler veya evrim ağaçları bu türler arasındaki evrimsel ilişkileri gösterir. Ağacın yaprakları verilen türleri, ara düğümleri ortak ataları ve ayrıtları da genetik ilişkileri belirtir. Türler genom bilgileriyle tanımlanabilir ve türler arasındaki evrimsel ilişkiler bir genomu diğerine dönüştüren genom yeniden düzenleme olayları (mesela mutasyonlar) ile ölçülebilir. Genom verisinden filojeni çıkarımı yapmak için kullanılan yaklaşımlardan biri, türler arasındaki toplam evrimsel mesafeyi en aza indirgemeye çalışırken genom üçlüleri için ortanca genom problemi çözmek veya genom çiftleri için genom yeniden düzenleme problemi çözmektir. Bu tezde, yeniden filojeni kurma problemini çözmek amacıyla, gen tekrarı içermeyen aynı uzunluktaki dairesel genomlar için ortanca genom problemi çözümüne dayanan iki tane arama tabanlı algoritma geliştirdik ve gerçekledik. Algoritmalarımızın uygulanabilir ve etkili olduğunu gösterebilmek adına rastgele üretilmiş örnekler ve Metazoa'ya ait türlerin mitokondri genomlarını ve Campanulaceae ailesine ait türlerin kloroplast genomlarını içeren iki gerçek veri kümesiyle algoritmalarımızı sınadık.

This thesis is financially supported by Tubitak-BIDEB 2210 grant for master studies.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

We present applications of heuristic search from Artificial Intelligence to three well-studied problems in computational biology:

- **Genome Rearrangement Problem:** Given two genomes, i.e., sequences of genes, find the minimum number of rearrangement events (e.g., inversions, transpositions) that transform one genome to another.
- **Median Genome Problem:** Given three genomes, find a genome such that the total number of rearrangement events that transform each given genome to this genome is minimum.
- **Phylogeny Reconstruction Problem:** Given n genomes, find an unrooted full binary tree, whose leaves correspond to the given genomes and internal nodes correspond to their common ancestors, such that the total number of rearrangement events that transform the given genomes to their children is minimum.

These problems are important in understanding the evolutionary relations between species.For instance,, the phylogenetic relationship between the pathogens from individuals in an epidemic contribute valuable epidemiological information about transmission chains and epidemiologically significant events [42, 33].

A computational problem is defined as a search problem over a search space, which can be viewed as a directed graph whose vertices correspond to search states and the edges denote transitions, with an initial state and a goal. The aim is to find a path from the initial state to a goal state; the path characterizes a solution to the given problem. Heuristic search provides strategies for finding this path, utilizing a heuristic function.

The contributions of this thesis can be summarized as follows:

- We modeled the computational biology problems described above as search problems.
- We introduced algorithms to solve these problems, using the existing heuristic search strategies (i.e., A^* search, IDA^* search, greedy best-first search) from artificial intelligence and heuristics from computational biology.
- We studied their optimality and computational complexity.
- We implemented the algorithms and tested them with artificial and real data sets.

Rest of this thesis is organized as follows: A brief introduction to search problems and widely-used search strategies are presented in Chapter 2. We give a formal definition of the genome rearrangement problem in Chapter 3. This chapter also explains our model for the genome rearrangement problem and the algorithm based on A^* search strategy to solve it. The median genome problem, our search model and the algorithm based on greedy best-first search strategy is presented in Chapter 4. In Chapter 5, our studies on the phylogeny reconstruction problem is presented. In this chapter, two algorithms based on greedy best-first search strategy are introduced and their properties are studied. We conclude this chapter with presenting results of these algorithms on two real data sets: chloroplast genomes of *Campanulaceae* and mitochondrial genomes of *Metazoan* families. We conclude with a brief summary of our contributions and remarks on future work in Chapter 6.

Chapter 2

Heuristic Search

A search problem is a computational problem where the goal is to find a solution in a solutions space (a set of possible solutions). The solution space is sometimes called a search or state space and its elements are called the states. During the search, moving from a state to another is possible by some actions.

The input of a search problem (over a search space $\mathcal{S}$) consists of the following:

- **Initial State:** An element of $\mathcal{S}$ from where the search starts.
- **Goal Test:** A function from $\mathcal{S}$ to $\{true, false\}$ which decides whether a given state satisfies properties of goal state or not.
- **Successor Function:** A function $succ : \mathcal{S} \to \mathcal{P}(\mathcal{S})$ that returns the set of possible states that can be reached from a given state where $\mathcal{P}(\mathcal{S})$ is the power set of $\mathcal{S}$ i.e., set of all possible subsets of $\mathcal{S}$.
- **Step Cost Function:** A function $c : \mathcal{S} \times \mathcal{S} \to \mathbb{N}$ that determines the cost of moving from one state to another. We assume that step cost function is nonnegative.

The output of a search problem is a sequence of states leading from the initial state to a goal state. Quality of the solution can be determined by the total cost of the edges in the path. An *optimal solution* is a solution with the lowest total edge cost.

Most of the search algorithms build a "search tree", starting from the initial state. Search tree contains nodes which are constructed from the states of the problem. New nodes are generated by applying the successor function *succ* to corresponding states of nodes. After new nodes are generated, the algorithm chooses one of them to continue

search, unless the chosen node does not contain a goal state. A generic search algorithm is presented in Algorithm 1[1].

Algorithm 1 TreeSearch

Input: initial state *init*, successor function *succ*, goal test function GoalTest, Step cost function *c*

Output: A solution or failure

```
1:
2: fringe ← ∅ //Leaves of the search tree ordered using c
3: Make a node from init and insert it to fringe
4: while true do
5:    if fringe is empty then
6:       return failure
7:    end if
8:    node ← first element of fringe
9:    if GoalTest(node) succeeds then
10:      return solution
11:   end if
12:   for Each state s in succ(corr. state of n) do
13:      Make a node for s and insert it to fringe
14:   end for
15: end while
```

We evaluate a search algorithm's performance according to four criterion:

- **Completeness:** If a solution exists, then can algorithm find it?
- **Optimality:** Is the solution found by algorithm optimal?
- **Time Complexity:** What is the asymptotical complexity of the number of steps required to find a solution?
- **Space Complexity:** What is the maximum amount of memory used during search?

The performance of an algorithm can be measured with respect to different factors like:

- *Branching factor:* The maximum number of nodes generated at any state
- Depth of the shallowest goal node where depth of a node is the length of the path from the initial node to this node

[1] Algorithm is adapted from Section 3.4. of [59]

- The length of the maximum-length path in the search tree
- The minimum value of c for states in state space
- The total cost of an optimal solution

Heuristic search utilizes a heuristic function to determine the next state to explore in the search tree. A^* search is one of the most popular heuristic search strategies. Consider the TreeSearch algorithm depicted in Algorithm 1. An A^* search algorithm is a variation of this algorithm: It evaluates nodes in the fringe with respect to a function f from set of nodes in the search tree to $\mathbb{N}$. An A^* search picks a node in the fringe with minimum f value as the current node. For a node n, $f(n)$ is defined in terms of two functions: a cost function $g(n)$ that returns the total cost of the path from the initial node to n, and a heuristic function $h(n)$ that estimates the total cost of the path from n to a goal node. Then,

$$f(n) = g(n) + h(n).$$

Intuitively, $f(n)$ estimates the total cost of a solution that passes through n.

Before explaining properties of A^* search, we present a similar heuristic search strategy called *Greedy Best-First Search* (GBFS). GBFS expands a node that is predicted to be the closest to a goal node considering only heuristic function h. More formally, GBFS expands nodes according to evaluation function $f = h$.

An A^* search is both complete and optimal if the heuristic function h satisfies some properties. For every node n in the search tree, if $h(n)$ never overestimates the cost to reach a goal state, i.e., the total cost of every path from n to a goal state is not less than $h(n)$, then h is an *admissible heuristic*. If h is admissible, then the A^* search algorithm is optimal.

For every node n and every successor n' of n, if

$$h(n) \leq c(n, n') + h(n'),$$

then h is called *consistent* (or *monotone*). Every monotone heuristic is admissible. Therefore, an A^* search algorithm with a monotone heuristic always finds optimal solutions.

The time and space complexity of an A^* search algorithm depends on the heuristic function h. Let $h^*(n)$ denote the minimum actual total cost of getting from a node n

to a goal state. If for every node n in the search tree

$$|h(n) - h^*(n)| \leq \mathcal{O}(\log h^*(n)),$$

then the time complexity of the A^* search is polynomial ([59]). However, almost for all heuristics, the difference between the actual cost and the heuristic estimate is proportional to the path cost. Consequently, the time complexity of the algorithm grows exponentially. In order to keep track of the paths from the initial node, the A^* needs to store all generated states. Therefore, the space complexity of A^* search is also exponential.

Iterative-Deepening A^* (IDA^*) search is an iterative application of A^* search. The idea is to generate states whose f value are above a threshold $flimit$. An IDA^* algorithm is shown in Algorithm 2 and 3.

Algorithm 2 IDA^*

Input: initial state *init*, successor function *succ*, goal test function GoalTest, Step cost function c

Output: a solution or failure

```
1: for flimit ← 0 to ∞ do
2:    result ← DepthLimitedA*(init,succ,GoalTest,c,flimit)
3:    if result ≠ failure then
4:       return result
5:    end if
6: end for
```

A summary of A^* and IDA^* search algorithms in comparison with Breadth-First Search (BFS) and Depth-First Search (DFS) algorithms are presented in Table 2.1[2].

Table 2.1: Evaluation of search strategies: b is the branching factor, d is the depth of the shallowest solution, m is the maximum depth of search tree, k is the optimal solution cost and e is the minimum edge cost. A^* and IDA^* search algorithms are assumed to have monotone h functions. Superscripts denote: ^{a} if b is finite, ^{b} it step costs are identical.

Criterion	BFS	DFS	A^*	IDA^*
Completeness	Yes^a	No	Yes	Yes
Optimality	Yes^b	No	Yes	Yes
Time Comp.	$\mathcal{O}(b^{d+1})$	$\mathcal{O}(b^m)$	$\mathcal{O}(b^{k/e})$	$\mathcal{O}(b^{k/e})$
Space Comp.	$\mathcal{O}(b^{d+1})$	$\mathcal{O}(bm)$	$\mathcal{O}(b^{k/e})$	$\mathcal{O}(k)$

[2]The table is adapted from Section 3.4. of [59]

Algorithm 3 DepthLimitedA^*

Input: initial state *init*, successor function *succ*, goal test function GoalTest, Step cost function c, $flimit$

Output: A solution or failure

```
1:
2: fringe ← ∅ //Leaves of the search tree ordered using c
3: Make a node from init and insert it to fringe
4: while true do
5:    if fringe is empty then
6:       return failure
7:    end if
8:    node ← the lowest f-valued element of fringe
9:    if GoalTest(node) succeeds then
10:      return solution
11:   end if
12:   for each state s in succ(corr. state of n) do
13:      If f(node) ≤ flimit, make a node for s and insert it to fringe
14:   end for
15: end while
```

Chapter 3

Genome Rearrangement Problem

Genetic information is transferred between organisms through hereditary molecules inside cells. Those molecules are called *chromosomes*. Set of all chromosomes (or set of all hereditary molecules) is referred as *genome*. Basically, chromosomes are made of DNA which is a double stranded sequence of nucleotides. One strand of DNA can be obtained from the other strand. Specific DNA segments contains information for production of other molecules (i.e., proteins, enzymes). Those segments are called as *genes*. Therefore, a genome can be viewed as a sequence of genes.

One way to measure evolutionary relatedness of two species is by comparing their genomic content, considering possible mutations that the species are exposed during reproduction. The idea is to find the number of mutations that transform one genome to another; so the less the number of these mutations the closer the genomes are.

Mutations can change the order, content or the length of genomes. Some mutations (called point mutations) occur in a small scale affecting a nucleotide of a DNA whereas some mutations occur in larger scale affecting greater DNA sequences. These large-scale mutations are called *genome rearrangement events* (or *rearrangements* in short). Researchers mostly consider the genome rearrangement events while comparing genomes because they occur less frequently than the point mutations [58]. Since these events are rare, finding the minimum number of rearrangements as a measure of evolutionary distance between two species is plausible.

With these motivations, the researchers have studied the following computational problem to better understand the evolutionary relations between species:

> **Genome rearrangement problem** Given the gene order of two whole genomes, find the minimum number of rearrangement events that transform one genome to

the other.

A genome can contain different number of chromosomes. If it contains just one chromosome, it is called a *monochromosomal* genome. Otherwise, it is called a *multichromosomal* genome. Although, most of the advanced species have multichromosomal genomes, they contain single chromosome organelles like chloroplasts [50], [28] or mitochondria [51].

A chromosome may be *circular* which means chromosome forms a circle. This is the case for bacteria. On the other hand, a chromosome can be linear. This is the case for more complex genomes like animal and plant genomes.

A gene may occur only once or multiple times inside a genome. We call a genome whose genes are all different, as a *genome without duplicate genes* and a genome whose some genes occur multiple times, as a *genome with duplicate genes*. In the following we consider monochromosomal, circular genomes without gene duplication.

There are numerous rearrangement events:

- If a segment of a genome inside a chromosome is reversed and its orientation of genes inside segment is also reversed but location of the segment does not change, it is called an *inversion* (or a *reversal*).

- If a genome segment is moved from one location to another without changing its genes' orientation, this rearrangement event is called as *transposition*. One can notice that this definition is equivalent as interchanging locations of genome segments without changing their orientation.

- A *translocation* rearrangement is defined on multichromosomal linear genomes. It operates on two chromosomes. Firstly, it splits each chromosome into two parts from arbitrary points and then ties end part of one chromosome to beginning of the other without changing orientation.

- *Block interchanges* exchanges two genome segments without changing their orientation. Notice that a transposition can be seen as block interchange of two adjacent segments.

- A *transversion* is an inverted transposition. It moves a genome segment from one location to other like a transposition but it changes orientation of moved segment i.e. it inverts moved segment.

- If two linear chromosomes are merged into one linear chromosome, it is called a *fusion*. If just one gene is inserted into a specific location of a chromosome, it is called an *insertion*.

- Conversely, a *fission* operation splits a genome into two. If just one gene is deleted from a chromosome, it is called a *deletion*.

- If two arbitrary points on a genome is chosen and the genome is split from these points and then those ends could be connected in all possible ways, this rearrangement is called a *double cut-and-join (DCJ)*. DCJ could only be applied to multichromosomal genomes, since a DCJ on a single chromosome may produce two chromosomes. DCJ is an important rearrangement since many other rearrangements can be obtained by one or two DCJ rearrangements and it is biologically relevant.

Most of these rearrangement events are not applicable or suitable for monochromosomal genomes or circular genomes. Moreover, there are debates over some rearrangements whether they are really biologically relevant or not. Therefore, we consider inversions and transpositions only.

Our consideration of inversions and transpositions as rearrangement events fit the phylogeny reconstruction model known as *Generalized Nadeau-Taylor (GNT) model* [76] which is based on *Nadeau-Taylor model* [49]. In this model, only inversion, transposition and transversions can occur. Moreover, each inversion is equiprobable. Same is true for transpositions and transversions. However, if occurrence of an inversion has probability ω_i, transposition has probability ω_t and transversion has probability ω_v, then $\omega_i + \omega_t + \omega_v = 1$. We use this model with $\omega_i = \omega_t = 0.5$ and $\omega_v = 0$. Also, this model requires an estimation for number of rearrangements. We assume uniform distribution for each $k \leq n$.

Now we have given the necessary biological background and informally described the genome rearrangement problem with our assumptions, let us describe how we mathematically model this biology problem. In the following, we first describe how we view genomes as mathematical objects, and how we define rearrangement events. Then, we define the genome rearrangement problem. After reviewing the related work, we present our approach for solving this problem. In particular, we discuss how we model this problem as a search problem, and describe our search algorithms that utilizes some heuristics to solve it. We provide a theoretical analysis of these algorithms as well as

an experimental evaluation with randomly generated data sets.

3.1 Problem Description

3.1.1 Mathematical Description of Genomes

As we mentioned earlier, genome rearrangement problem heavily depends on structure of genome and rearrangements considered. For this study, we are working with monochromosomal cyclic genomes without duplicate genes. As we stated earlier, these genomes can be represented by a mathematical object called genomic circular signed permutations (GCSP). Before we present the definition of a GCSP, let us recall some definitions about permutations defined on a set $\mathcal{A} = \{1, 2, ..., n\}$:

Definition 1 (Permutation)**.** *A bijection $\pi : \mathcal{A} \to \mathcal{A}$ is called a permutation. The image of $i \in \mathcal{A}$ under π is denoted by π_i and π is denoted as a sequence.*

For instance, if we take $n = 4$, $\pi = (3, 2, 1, 4)$ is a permutation. It means $\pi(1) = 3$, $\pi(2) = 2$, $\pi(3) = 1$ and $\pi(4) = 4$.

Since elements of permutations do not contain sign we need to extend definition such that negative integers could be involved:

Definition 2 (Signed Permutation)**.** *A signed permutation is a permutation πi where each π_i is assigned to + or -.*

Now, we can define a binary operation $\circ$ on signed permutations as follows:

Definition 3 (Composition of signed permutations)**.** *Let π and σ be two signed permutations of length n. Then, $\circ : \mathcal{P} \times \mathcal{P} \to \mathcal{P}$ is a binary operation such that for all $x \in \mathcal{A}$, $(\sigma \circ \pi)(x) = \sigma(\pi(x))$.*

Circularity can be obtained from signed permutations by the property that shifted version of a circular permutation is again the same circular permutation. Therefore, let us define the *shift* operation first:

Definition 4 (Shift)**.** *Given $1 \leq i < n$, a* shift *on a signed permutation π can be described as another signed permutation $\theta(i)$, such that for all $1 \leq j \leq n$, $(\theta(i) \circ \pi)_j = \pi_{(j+i-1 \ mod \ n)+1}$.*

Consider signed genome π defined in previous example. $\theta(2) \circ \pi = (4, 2, -3, -1)$.

In order to proceed to genomic circular signed permutation consider following equivalence relation with symbol $\equiv$ between signed permutations:

Definition 5 (GCSP). *Let π and σ be two signed permutations over $\mathcal{A}$. Then, $\pi \equiv \sigma$ if there exists $1 \leq i < n$ such that $\sigma = \theta(i) \circ \pi$ or $\sigma = \theta(i) \circ (-\pi_n, ..., -\pi_1)$. We call the equivalence class of π a genomic circular signed permutation (GCSP) of $\mathcal{A}$, and denote it as $[\pi]$.*

Again, equivalence class of our example π above is $[\pi] = \{(-3, -1, 4, 2), (-1, 4, 2, -3), (4, 2, -3, -1), (2, -3, -1, 4), (-2, -4, 1, 3), (-4, 1, 3, -2), (1, 3, -2, -4), (3, -2, -4, 1)\}$. Now, we can precisely define GCSPs based on this equivalence relation as in [44, 68].

Consider genomes (or in our case the single chromosome) with n genes. We can represent each gene with a label from the set $\{1, ..., n\}$. Then, genomes can be represented as genomic circular signed permutations.

3.1.2 Mathematical Definition of Rearrangement Events

Now, we can define rearrangements on GCSPs:

Definition 6 (Inversion). *For some $1 \leq i \leq j \leq n$, an* inversion *$inv(pi_i, pi_j)$ on a GCSP $[\pi]$ is a GCSP obtained by inverting the gene sequence between two labels π_i and π_j as follows:*

$$inv(\pi_i, \pi_j) \circ [\pi] = [(\pi_1, ..., \pi_{i-1}, -\pi_j, ..., -\pi_i, \pi_{j+1}, ...\pi_n)]$$

Definition 7 (Transposition). *For some $1 \leq i < j \leq k \leq n$, a* transposition *$trp(\pi_i, \pi_j, \pi_k)$ on a GCSP $[\pi]$ is a GCSP obtained by inverting the gene sequence from π_j to π_k before π_i as follows:*

$$trp(\pi_i, \pi_j, \pi_k) \circ [\pi] = [(\pi_1, ..., \pi_{i-1}, \pi_j, ..., \pi_k, \pi_i, ..., \pi_{j-1}, \pi_{k+1}, ..., \pi_n)]$$

For instance, for previous $[\pi] = [(-3, -1, 4, 2)]$, $inv(-1, 4) \circ [\pi] = [(-3, -4, 1, 2)]$ and $trp(4, 2, -1) \circ [\pi] = [(-1, 2, -3, 4)]$.

Note that rearrangements take images of i, j, k instead of i, j, k as parameters since there is no definite index position in this circular structure. Moreover, rearrangements are applicable for all GCSPs over the elements of $\mathcal{A}$.

3.1.3 Mathematical Description of the Genome Rearrangement Problem

Now, we define genome rearrangement problem on monochromosomal cyclic genomes without duplicate genes, with transposition and inversion rearrangements [44]:

> Given two GCSPs $[\pi]$ and $[\sigma]$ find the minimum $k \in \mathbb{N}$ such that there exists a sequence of inversions and transpositions $\eta^1, ..., \eta^k$ such that $[\sigma] = \eta^k \circ ... \circ \eta^1 \circ [\pi]$.

Actually, this problem can be reduced to a problem called *sorting GCSPs by reversals and transpositions* [61] where the goal is to find the minimum $k \in \mathbb{N}$ number of rearrangements $\eta^1, ..., \eta^k$ such that $\eta^1 \circ ... \circ \eta^k \circ [\pi] = [I]$. The sequence of rearrangement events is called a sorting sequence.

3.2 Related Work

Various versions of genome rearrangement problems have been widely studied. One of the most widely studied problem in this area is sorting signed permutations by reversals which is the same problem as the genome rearrangement problem for signed linear genomes [7]. It can be solved by a polynomial time algorithm [35] of complexity $\mathcal{O}(n^4)$ as well as an algorithm of asymptotical time complexity $\mathcal{O}(n)$ [4] where n is the number of genes. There have also been studies to develop parallelized algorithms like in [39, 66]. Although this problem has linear time solutions, it is interesting that sorting unsigned permutations by reversals problem is NP-Hard [18].

There are variations of sorting signed integers by reversals. One of them assigns weights to reversals such that the longer the reversed sequence, the more weight assigned to it, then finds a sequence of reversal such that the total cost of reversals is minimum by a polynomial time approximation [69]. Another variant of this problem fixes the length of reversals (i.e., a special case of the previous variant with a weight function which is nonzero for only one value) to even numbers [55].

For the circular genomes, sorting GCSPs by reversals problem has also been studied. [44] presents a polynomial time algorithm for this problem.

Problem gets more complicated if we involve transpositions. Sorting permutations only by transpositions has recently been shown to be NP-Hard [16]. Notice that there are signed permutations which can not be transformed to identity permutation I by transpositions only due to the fact that transpositions do not change the signs of genes.

However, problem of sorting signed permutations with transpositions and reversals is valid since all signed permutations can be transformed into each other with these rearrangements. It is possible to assign different weights to transpositions and reversals, and change the problem to finding minimum total weight while sorting a signed permutation to I. The complexity of this problem is still unknown. However, there are various approximation results for different α values where α is the ratio between the weight of a transposition and the weight of an inversion. For $\alpha = 1$, there are 2-approximation algorithms presented by [34, 75, 43]. For $1 \leq \alpha \leq 2$ a 1.5-approximation algorithm exists [6]. For $\alpha = 2$ there is a $(1+\varepsilon)$-approximation algorithm for any $\varepsilon > 0$ [30], based on well known polynomial time algorithm developed for inversions only [35]. It can be observed that any transposition on signed permutations can be obtained by three reversals. Therefore, for $\alpha \geq 3$ the solution to the problem of sorting by reversals and transpositions will be no better than sorting only by reversals which has linear time solution algorithm.

There are also studies for sorting signed permutations by reversals, transpositions and transversions. For instance, a 1.5-approximation algorithm assuming that the weights of all rearrangements are equal, is presented in [37].

Similar problems have been studied for genomes with duplicate genes. Since we do not consider in this thesis the rearrangements which change the gene content of genomes, we assume that genomes are *balanced* (i.e., they contain the same number of copies of each gene). Genome rearrangement problem for balanced genomes with gene repetition is defined similar to the genome rearrangement problem for genomes without gene repetition by using inversions, transpositions and/or transversions. For *unbalanced* genomes, rearrangements which changes the gene content of genome like insertion, deletion, fusion or fission are required. Rearranging unbalanced genomes is considered to be harder than balanced ones. We will not refer to any problems or theoretical results for problems on unbalanced genomes in this thesis, since there are numerous different genome rearrangement problems for even rearranging unbalanced strings by the same rearrangements. Therefore, we consider problems on balanced genomes for the rest of this part. Genomes with gene duplication can be represented by sequences.

The problem of rearranging by reversals persists to be difficult for the genomes with duplicate genes [56], [20]. This problem is shown to be NP-Hard [56]. However, there are approximation algorithms. If the maximum occurrence of a symbol in one of the

sequences is c, then this problem can be solved by a $\mathcal{O}(c)$ algorithm [40]. Also, there is a $\mathcal{O}(n^{0.69})$ approximation algorithm to solve this problem [24].

The problem of rearranging unsigned sequences by transpositions is also studied [23]. It is shown to be NP-Hard [56]. For the similar reason explained above (i.e. there may be two genomes which can not be transformed into each other by transpositions only), this problem is not defined for signed sequences. There are approximation algorithms for this algorithm as well. The algorithm mentioned above [40] can solve this problem as well. In addition, there is $\mathcal{O}(log\ nlog^*\ n)$ approximation algorithm presented in [25] where log^*n is the number of log function applied to n until a constant is obtained.

Also, block-interchange rearrangements are studied for unsigned sequences and proven to be NP-Hard [21].

Another line in genome rearrangement studies does not consider aforementioned rearrangements. We have already stated that DCJ rearrangement is biologically relevant and can replace various other rearrangements. Sorting signed permutations by DCJ problem is presented in [78] first time. Then, polynomial time algorithms developed to solve this problem. [8] presents an $\mathcal{O}(n)$ algorithm for this problem.

There is also software for solving genome rearrangement problem with inversions, transpositions and transversions. DERANGE II [11] can handle both circular and linear genomes and the user can assign different weights to each rearrangement type. GENOMEPLAN [29] models this problem as a planning problem in *Action Description Language (ADL)* [52] in study and solves by a planner called TLPLAN [3].GENOMEPLAN can also solve problems with duplicate genes. GRIMM can handle linear or circular, monochromosomal or multichromosomal, signed or unsigned genomes with translocations, reversals, fusions and fissions [72].

3.3 Modeling the Genome Rearrangement as a Search Problem

We have modeled the genome rearrangement problem as a search problem.

Search states A search state consists of a single GCSP instead of two GCSPs. As we stated earlier, the genome rearrangement problem (that we defined for circular monochromosomal genomes without duplicate genes and with transpositions and inversions) can be reduced to the problem of sorting GCSPs by transpositions and inver-

sions by relabeling the genes. Therefore, it is sufficient to keep information about one genome after relabeling and try to find a sequence of rearrangements that transform it to identity GCSP $[I]$. Hence, we represent a search state by a single GCSP (obtained after relabeling). Then, state space becomes $\mathcal{G}$. Note that since we do not represent a search state by two genomes, the memory consumption is not as much. In the following, we use the terms "state" end "GCSP" interchangeably.

Successor function Let $2^{\mathcal{G}}$ denote the power set of $\mathcal{G}$ which is a multi set consisting of all possible subsets of $\mathcal{G}$. Then, the successor function is defined as $\delta : \mathcal{G} \rightarrow 2^{\mathcal{G}}$. Basically, δ takes a GCSP Π and maps it to the set of GCSPs that can be obtained from Π by a transposition or an inversion. We have considered two variations of the successor function under some restrictions. We explain these restrictions after we explain the search model.

Goal Test A state s with a GCSP Π is a goal state if and only if $\#bp(\Pi) = 0$

To utilize A^* search strategies we have also defined a cost function and a heuristic function.

Cost function g Since we are looking for the minimum number of rearrangements between two GCSPs without giving priority to some of them, for a state s, $g(s)$ is the number of rearrangements applied from initial state until obtaining s.

To define our heuristic function, we need to recall well-known breakpoint definition [44].

Definition 8 (Breakpoint)**.** *Let* Π *and* Σ *be two GCSPs of length* n*. Then, for some* $a, b \in \{-n, ... -2, -1, 1, 2, ..., n\}$*,* (a, b) *is a breakpoint of* Π *if there exists* $\pi \in \Pi$ *and* $1 \leq i < n$ *such that* $\pi_i = a$*,* $\pi_{i+1} = b$ *and for all* $\sigma \in \Sigma$ *and* $1 \leq j < n$ *neither* $\sigma_j = a$ *and* $\sigma_{j+1} = b$ *nor* $\sigma_j = -b$ *or* $\sigma_{j+1} = -a$*. We denote the number of breakpoints of* Π *by* $\#bp(\Pi)$*.*

Although mathematical definition is complicated, we can say that two consecutive elements (a, b) in Π is a breakpoint if neither (a, b) nor $(-b, -a)$ occurs consecutively in Σ. Consider $\Pi = [(-3, -2, 1, -4)]$ and $[I]$. Only breakpoints of Π are $(-2, 1)$ and $(1, -4)$. Also note that two GCSPs are equal if and only if both of them have 0 breakpoints.

Heuristic function h For a state s characterizing a GCSP Π, $h(s) = \#bp(\Pi)/3$.

To define restrictions on the successor function δ, we need definition of strip [21]:

Definition 9 (Strip)**.** *Let* Π *and* Σ *be two GCSPs of length* n*. Then, for some* $k \leq n$*,* $1 \leq i \leq k$*,* $l_i \in \{-n, ..., -2, -1, 1, 2, ..., n\}$ *and* $\pi \in \Pi$*,* $(l_1, ..., l_k)$ *is called a strip if* l_1 *and* l_k *are involved in a breakpoint of* π *but for all* $1 \leq j < k$ (l_j, l_{j+1}) *is not a breakpoint of* π*.*

Strips can be seen as maximal consecutive elements that commonly exist in both GCSPs. This definition also can be clarified by an example. Take same Π and $[I]$ of above example. Then, only strip in Π is $(-4, -3, -2)$. Note that strips of a GCSP are disjoint and both GCPSs contain the same set of strips. Moreover, the elements of breakpoints are the end points of strips.

Now, we can more precisely define the restrictions on the successor function δ. For a state s containing GCSP Π of length n, another state s' containing GCSP Σ is an element of $\delta(s)$ if

(R_1) $\#bp(\Sigma) < \#bp(\Pi)$ with respect to $[I]$ and

(R_2) one of the followings holds:

- for some $a, b \in \{-n, ..., -1, 1, ..., n\}$, $\Sigma = inv(a, b) \circ \Pi$ and a and b are end points of some strips
- for some $a, b, c \in \{-n, ..., -1, 1, ..., n\}$, $\Sigma = trp(a, b, c) \circ \Pi$ and a, b, c are end points of some strips.

Note that δ does not generate states for all possible transpositions and inversions of a GCSP with these restrictions. According to the first restriction R_1, only the rearrangements that does not split strips are allowed. According to the restriction R_2, the rearrangements that reduces the number of breakpoints with the goal state are allowed.

3.4 Theoretical Results

In this section, we present some results and properties of the A^* search model described in previous section.

Let s first analyze the properties of our heuristic function.

Proposition 1. *Let s be a state with GCSP Π and $\#bp(\Pi)$ be number of breakpoints of Π with respect to $[I]$. Then, without restrictions R_1 and R_2, $h(s) = \#bp(\Pi)/3$ is monotone.*

Proof. Consider a state s' with GCSP Σ, which can be obtained from Π by a single inversion or a single transposition. A transposition between Π and Σ can reduce at most three breakpoints since it operates on three points on a GCSP and an inversion can reduce at most two breakpoints since it operates on two points. Therefore,

$$\#bp(\Pi) \leq \#bp(\Sigma) + 3.$$

Then,

$$\begin{aligned} \frac{\#bp(\Pi)}{3} &\leq \frac{\#bp(\Sigma)}{3} + 1 \\ h(s) &\leq h(s') + c(s, s') \end{aligned}$$

Consequently, h is monotone. □

Due to this proposition, h is admissible and the A^* search (without restrictions R_1,R_2) terminates with an optimal solution.

Termination and Optimality Now, we give some results about the optimality of the solutions. Our restrictions R_1 and R_2 prunes search tree and restricts the search space. Therefore, we need to examine how R_1 and R_2 affects the quality of solutions. We will examine affects of restrictions to optimality separately starting from R_1. To do that we need definition of a minimal GCSP [21]:

Definition 10 (Minimal GCSP)**.** *Let Π be a GCSP and have r strips with respect to some other GCSP Σ. Then minimal GCSP of Π is a GCSP consisting of elements from $\{-r, ..., -1, 1, ..., r\}$ obtained from Π by replacing each strip by an element of this set renumbering each element such that breakpoints and matching elements are preserved. It is denoted by $min(\Pi)$.*

For instance take $\Pi = [(2, -5, -4, 3, 1)]$. Then, its minimal version with respect to $[I]$ is $min(\Pi) = [(1, -3, 2)]$.

Note that R_1 allows rearrangements to operate only on end points of strips. Therefore we can think of this as algorithm only using R_1 restriction solves sorting $min(\Pi)$

problem. Therefore, in order to show that R_1 does not violate the optimality we need to show that two problems have the same length sorting sequences.

We need further notation for our claims. Let $id(\Pi)$ denote the length of the minimum sorting sequence of GCSP Π only using inversions, $td(\Pi)$ denote the length of the minimum sorting sequence for only using transpositions, and $tid(\Pi)$ denote the length of the minimum sorting sequence of Π by using transpositions and inversions. Same notation can be used for permutations and signed permutations.

Actually, the same problem is widely studied for permutations. For a permutation π, $id(\pi) = id(min(\pi))$ has been tried to be shown for unsigned permutations where inversion can only flip permutation but can not change its sign. In [36], it is shown that $id(\pi) = id(min(\pi))$ for unsigned permutations if all strips of π has length more than 2 based on the polynomial time algorithm of sorting signed permutations with inversions described in [35]. Again for a permutation π, $td(\pi) = td(min(\pi))$ is shown in [22]. We will use this method while proving our results.

Firstly, we can see the correspondence between rearrangements of $min(\Pi)$ and Π for some GCSP Π. Every rearrangement on $min(\Pi)$ can be mimicked by a rearrangement on Π. Consider a/an transposition/inversion on $min(\Pi)$. Operation points of the rearrangement represent strips in Π. By applying the same rearrangement to the corresponding end points of Π we obtain a new GCSP of which minimal version is the same as rearrangement applied to $min(\Pi)$. For instance, consider same $\Pi = [(2, -5, -4, 3, 1)]$ and $min(\Pi) = [(1, -3, 2)]$ of above example. $inv(-3, 2)$ on $min(\Pi)$ can be mimicked by $inv(-5, 3)$ on Π which result in same minimum GCSPs $[(1, -2, 3)]$. Similarly, $trp(1, -3, 2)$ on Π can be mimicked by $trp(1, -5, 3)$ on Π and both rearrangements results in minimum GCSP $[(-3, 1, 2)]$. Using this similarity, we can prove our result:

Proposition 2. *Let Π and $min(\Pi)$ be GCPSs of length n and r respectively. Then, $tid(\Pi) = tid(min(\Pi))$.*

Proof. First, we show that $tid(\Pi) \leq tid(min(\Pi))$. By the mimicking rearrangements of $min(\Pi)$ it is easy to show that for every sequence of rearrangements that transforms $min(\Pi)$ to $[I]$, one can find a sequence of rearrangements of the same length that transforms Π to $[I]$. Therefore, $tid(\Pi) \leq tid(min(\Pi))$.

Now, we show $tid(\Pi) \geq tid(min(\Pi))$. We form a new GCSP Σ by inserting $n-r-1$ asterisks to $min(\pi)$ as follows. For all labels a in $min(\Pi)$ and corresponding strip γ in Π, we insert asterisks to the clockwise adjacency of a such that number of asterisks added are one less than length of γ. For instance, let $\Pi = [(-4, -3, -5, 1, 2)]$. Then, $min(\Pi) =$

$[(-2, 3, 1)]$ and $\Sigma = [(-2, *, -3, 1, *)]$. We can find a bijection between labels of Π and Σ such that for any label a in $min(\Pi)$, a is mapped to first element of corresponding strip in clockwise direction. Then, for any rearrangement on Π one can use this mapping to find a rearrangement in Σ. A sequence of rearrangements that transforms Π to $[I]$ will also transform Σ to $[I]$ if we ignore asterisks. However, some inversions/transpositions may be applied to only asterisks and therefore they are not applied to $min(\Pi)$. Therefore, there can be a shorter sequence of inversions that transforms $min(\Pi)$ to $[I]$ for all such sequences. Consequently, $tid(\pi) \geq tid(min(\Pi))$. □

This proposition shows us that we can find optimal length sequence of rearrangements with our algorithm under restriction R_1.

However, we can not guarantee optimality if we involve R_2. Consider $\Pi = [(7, 6, 5, 4, 3, 2, 1)]$. If restriction R_2 is considered, Π can be sorted by applying minimum seven rearrangements (actually, only transpositions) which brings an integer to the correct position at each rearrangement. However, there exists a sequence of four transpositions that sorts Π, if we do not consider R_2 [79].

Complexity We also examine effects of R_1 and R_2 on computational complexity. First, lets consider the case without any restrictions. We know that a transposition operates on three points in a GCSP. Moreover, any three points forms a valid transposition. Then, we can generate $\mathcal{O}(n^3)$ new states from a state with a GCSP of length n. Similarly, an inversion operates on two points and any two points on a GCSP forms an inversion. Therefore, we can generate $\mathcal{O}(n^2)$ new states from a state with a GCSP of length n. Consequently, branching factor of each search state is $\mathcal{O}(n^3)$.

Now, let us explain how R_1 affects the branching factor of the states.

Proposition 3. *Let s be a state with GCSP Π and $\#bp(\Pi)$ be number of breakpoints of Π with respect to $[I]$. Then, A^* search algorithm with R_1 generates $\mathcal{O}(\#bp(\Pi)^3)$ new states from s.*

Proof. With R_1, the A^* algorithm only generates states with rearrangements that operate on the ends of strips. By definition, they are breakpoints and every breakpoint is an end of a strip. Following the steps for the unrestricted case, algorithm generates $\mathcal{O}(\#bp(\Pi)^3)$ states by applying transpositions and $\mathcal{O}(\#bp(\Pi)^2)$ new states by applying inversions. Consequently, $\mathcal{O}(\#bp(\Pi)^3)$ new states are generated. □

Since there can be at most n breakpoints, the branching factor does not asymptotically change with R_1. However, for close species with long gene sequences, the effect of R_1 is significant. But if we take R_2 into account, the branching factor changes considerably.

Proposition 4. *Let s be a state with GCSP Π of length n. Then, A^* search algorithm with R_2 generates $\mathcal{O}(n^2)$ new states from s.*

Proof. Since R_2 ensures decreasing number of breakpoints, rearrangement applied to Π must relieve at least 1 breakpoint. Without loss of generality, assume that applied rearrangement brings (a, a') together for $a, a' \in \{-n, ..., -1, 1, ..., n\}$ and (a, a') is not a breakpoint with respect to $[I]$. First consider that applied rearrangement is a transposition. Then, one of the operation points of this transposition must be adjacent to a and other one should be adjacent to a'. Third operation point can be chosen freely. Therefore, $\mathcal{O}(n)$ transpositions can bring (a, a') together. Secondly, assume applied rearrangement is an inversion. Again one of the operation points must be adjacent to a and other should be adjacent to a'. Therefore, there can be at most 2 inversions that brings (a, a') together. Consequently, there can be $\mathcal{O}(n)$ rearrangements that brings (a, a') together. Since Π may have at most n breakpoints, $\mathcal{O}(n^2)$ new states can be generated from s. □

If we consider R_1 and R_2 together as we did in our algorithm, the branching factor becomes combination of the previous two results.

Proposition 5. *Let s be a state with GCSP Π and $\#bp(\Pi)$ be number of breakpoints of Π with respect to $[I]$. Then, A^* search algorithm with R_1 and R_2 generates $\mathcal{O}(\#bp(\Pi)^2)$ new states from s.*

This result has been shown by Tansel Uras (personal communication).

3.5 GenomeSEARCH-A^*

Our implementation of the A^* search algorithm to solve the genome rearrangement problem is called GenomeSEARCH-A^*. The states of the search model is implemented as a collection of a vector of pairs of integers, a vector of triples of integers and two integers. Each element of this collection contains information about the search states explained in the following.

GCSP As we stated earlier, we keep a single GCSP inside a state since we can reduce the genome rearrangement problem to the corresponding sorting problem by transforming one of the GCSPs to $[I]$ by relabeling its elements. A GCSP of a state is represented by a vector of pairs of integers in our implementation. This data structure is described in detail in Section 3.5.1. This data structure helps successor function to operate on the ends of strips which is required by the restriction R_1. In this GCSP representation, locating a particular element in the GCSP and applying a rearrangement takes constant time. Therefore, it is useful for our implementation.

Rearrangement history We keep a vector of triples of integers inside the states to keep all the rearrangements applied from the initial GCSP to the current GCSP. An element (triple of integers) in this vector represents an inversion or a transposition. Each integer in the triple shows the operation point of the rearrangement. Since inversions operate on two points, last integer is set to -1 for an inversion. Therefore, one can understand whether a given triple is an inversion or a transposition by looking at the last integer of the triple. This vector reduces memory consumption of the states. Because with the help of this vector, we just keep frontier states —not the full search tree— and obtain the applied rearrangement history when required. Moreover, we can obtain g value of a state for A^* search easily by checking the size of this vector.

Start gene One of the integers inside the state implementation represents the *start gene* of the GCSP of a state. Although GCSPs are circular and have no definite end points, this arbitrary point eases our job while traversing a GCSP for generating all possible transpositions and inversions from the current GCSP. Start gene changes if the previous start gene falls inside a strip.

Breakpoint count The second integer inside the state implementation represents the number of breakpoints of the GCSP of a state with respect to $[I]$. It is not necessary to calculate number of breakpoints from the scratch for newly created GCSPs. It can be deduced from number of breakpoints of the GCSP of the predecessor state and how many breakpoints healed by the rearrangement applied to the GCSP of the predecessor state. Since h value of a state is a-third of the number of breakpoints of the GCSP of this state, h value of the states can be directly obtained from this integer.

Moreover, a heap of states are utilized for the A^* search. The heap gives the state in the fringe with the minimum f value in the constant time and inserting a new state

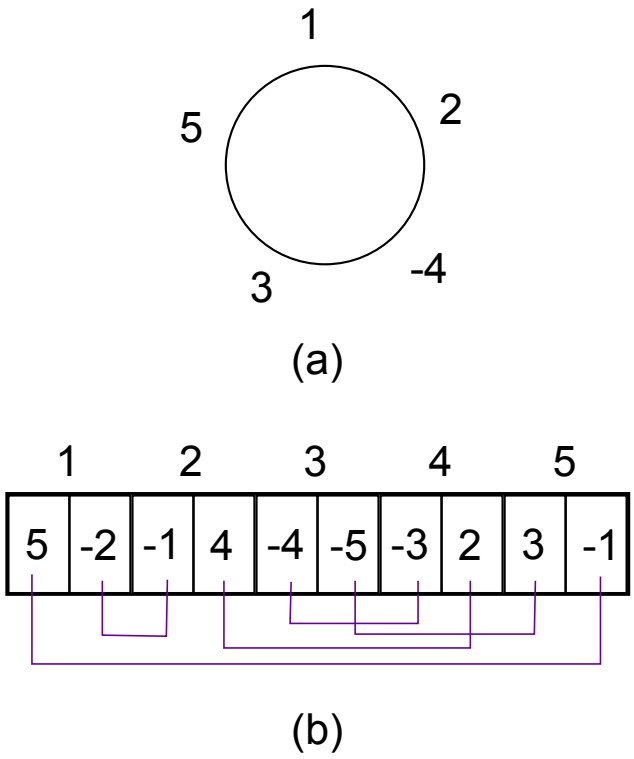

Figure 3.1: Implementation of a GCSP in GENOMESEARCH-A^*. (a) A GCSP of length five. (b) Representation of this GCSP in our implementation.

to the heap takes time logarithmically proportional to the size of the heap. Therefore, using heap to organize states in the fringe increases efficiency of our implementation.

3.5.1 Data Structure for the GCSP Representation

Our data structure for representing GCSPs is a vector of a pair of integers. Each index in this vector represents a gene in the corresponding genome of the GCSP. Using a vector, we can reach a particular gene in constant time.

If we label two ends of a gene as left and right ends, pair of integers inside the vector points to the left and right neighbors of this gene. Integers in a pair can be positive or negative. If it is negative, it means that this end of the gene is adjacent to the left end of its neighbor. Otherwise, it is adjacent to the right end of its neighbor. This data structure is depicted in Figure **??**.

With this data structure, strips can also be handled easily. Let $(a_1, ..., a_k)$ and $(b_1, ..., b_m)$ be two strips that come together as $(a_1, ..., a_k, b_1, ..., b_m)$ after a rearrangement. If (a_k, b_1) is not a breakpoint, we can omit all the genes between a_1 and b_m since restriction R_1 prevents rearrangements on this genes. We can achieve this by setting

right pointer of a_1 to $-b_m$ and setting left pointer of b_m to a_1 in constant time. Then, we can traverse a GCSP more efficiently and prevent generation of rearrangements that violate R_1.

Moreover, we can apply the rearrangements in constant time. To apply an inversion $inv(a, b)$ operating on two points a and b, changing the left pointer of a, the right adjacent of b and the right pointer of b, the left adjacent of a suffices. Note that applying an inversion on a GCSP implemented by a linked-list or a vector of integers can not be done in constant time, since signs of the all the elements between the two operation points should be reversed.

Similarly, a transposition $trv(a, b, c)$ operating on three points a, b and c can be implemented by changing the left pointers of a,b the right adjacent of c and the right pointers of the left adjacent of a, the left adjacent of b, c.

3.6 Experimental Results

In our experiments, we compared GenomeSEARCH-A^* with GenomePLAN and GenomeSEARCH-IDA^* in terms of computation times and accuracy. Moreover, we conducted experiments to measure effects of R_1 and R_2 in terms of computation time.

We conducted experiments on two data sets. First data set includes 100 random GCSPs of length 12. Each GCSP corresponds to a problem instance. Random GCSPs are obtained from $[I]$ by 50 random swaps. Each swap takes two labels and interchanges their content and changes their sign by %50 chance.

Second data set also contains 100 random GCSPs of length 12 and each GCSP corresponds to a problem instance. GCSPs are obtained from $[I]$ by six random inversions and six random transpositions.

All of the experiments are run on a machine with eight Intel Xeon E5310 CPUs (1.60 GHz, 8M Cache) and 16 GB memory. It contains CentOS 5.8 operating system and the software is compiled with $g++$ 4.1.2 $c++$ compiler.

For each problem instance, we put 1000 seconds time limit and 15 gb memory limit. If an instance exceeds the limits, this instance is omitted from the results and statistics.

3.6.1 Effect of Restrictions

As we stated in previous sections (see section 3.4), our restrictions make an asymptotic contribution to branching factor of A^* search which reduces it from $O(n^3)$ to

$O(\#bp(\Pi)^2)$ where Π is a GCSP of length n. Therefore, we were expecting significant time difference between performances of regular GenomeSEARCH A^* search and the one without restrictions. Our experimental results also supported our beliefs. Comparative results can be seen in Table 3.1:

Table 3.1: Results of the experiment that compares computation time of our algorithm with and without restrictions R_1 and R_2

Plan Length		With Restrictions		W/O Restrictions	
		# of Instances	Avg. Time	# of Instances	Avg. Time
Data Set 1	8	4	0.745	4	122.354
	9	7	6.073	2	582.014
	10	44	45.678	N/A	N/A
	11	45	238.590	N/A	N/A
Data Set 2	7	2	0.15	2	3.45
	9	12	7.430	1	792.490
	10	39	52.893	N/A	N/A
	11	45	212.440	N/A	N/A
	12	1	592.018	N/A	N/A

One can see from the table that, without restrictions, no plan of length greater than 9 is found. This algorithm can not find solutions to 94 instances from the first data set and 97 instances from the second data set. The algorithm with the restrictions could not solve only single instance due to exceeding the memory limit. We also observed that, instance solved by the both algorithms have the same plan length. Since we know that the algorithm without restrictions finds optimal solutions, the algorithm with R_1 and R_2 also finds optimal solutions for this data sets.

3.6.2 A^* and IDA^*

In order to measure performance of A^* search with other similar search algorithms, we ran experiments with IDA^* search. IDA^* search algorithm also contains two restrictions those we used for A^* search. Performance of IDA^* search compared with A^* search is shown in the Table 3.2.

As it can be seen from table, both search algorithms have similar performance until plan length 10. For instances of solution length 10 and 11, IDA^* search is slightly faster. We think that this result occurs because all of edges are integer cost and there are many states with same depth or g value. Therefore, at each iteration of IDA^* search, lots of new states are explored and thus iterations does not cause considerable

Table 3.2: Results of the experiment that compares computation time of A^* search strategy with IDA^* search strategy with restrictions R_1 and R_2 for the genome rearrangement model

Plan Length		A^* with Restrictions		IDA^* with Restrictions	
		# of Instances	Avg. Time	# of Instances	Avg. Time
Data Set 1	8	4	0.745	4	0.854
	9	7	6.073	7	7.526
	10	44	45.678	44	37.843
	11	45	238.590	45	179.395
Data Set 2	7	2	0.15	2	0.22
	9	12	7.430	12	7.842
	10	39	52.893	39	47.538
	11	45	212.440	45	203.448
	12	1	592.018	2	834.582

time loss since they are few. Although the A^* search seems faster for the instances with the solution length 12, the A^* search can not solve one of the instances which we suspect to have an optimal solution of the length 12, due to high memory requirements. However, IDA^* solves this instance since it has modest memory requirements.

3.6.3 GenomePLAN and GenomeSEARCH-A^*

In order to compare performance of our search algorithm with a planning approach, we run our instances on GenomePLAN. For detailed information on this software, please visit GenomePLAN homepage. We did not change default values for priority or cost options. We just changed search strategy from Depth First Search (DFS) to Breadth First Search (BFS) since DFS does not guarantee an optimal solution. Results were surprisingly different since GenomePLAN could not find a solution. In the Table 3.3, we compared these results with GenomeSEARCH-A^* without restrictions.

Table 3.3: Results of the experiment that compares computation time of A^* search strategy without restrictions with GenomePLAN using BFS strategy

Plan Length		GenomePLAN		A^* W/O Restrictions	
		# of Instances	Avg. Time	# of Instances	Avg. Time
Data Set 1	8	N/A	N/A	4	122.354
	9	N/A	N/A	2	582.014
Data Set 2	7	N/A	N/A	2	3.45
	9	N/A	N/A	1	792.490

3.7 Conclusion

In this section we mathematically modeled genome rearrangement problem for circular genomes without gene repetition as a search problem.

Then, we developed an A^* based algorithm to solve this problem. For a state s with GCSP Π, We determined heuristic function $h(s) = \#bp(\Pi)/3$ where $\#bp(\Pi)$ is the number of breakpoints of Π with respect to $[I]$. We proved monotonicity of h. Therefore, our basic A^* algorithm terminates with an optimal solution.

We also applied two restrictions R_1 and R_2 on the successor state function in order to obtain better computation times. R_1 allows rearrangements that only operate on strips and R_2 allows rearrangements that only reduces breakpoints between GCSP of the current state and $[I]$. We have shown that the A^* algorithm with R_1 does not violate optimality of the solutions, whereas we can find an instance for R_2 such that the A^* algorithm does not generate an optimal solution.

We implemented our algorithm as a software called GenomeSEARCH-A^*. For this implementation, we used a special data structure (a vector of pair of integers) for GCSPs inside search states which allows us applying inversions and transpositions on the GCSP in constant time and eases traversing the GCSP without considering genes inside the splits.

We conducted experiments with GenomeSEARCH-A^* on two simulated data sets. We compared computation time of the A^* algorithm with and without R_1 and R_2, with IDA^* using the same restrictions and with another software called GenomePLAN. We have observed that the restrictions increase the time efficiency of the algorithm. IDA^* and A^* have the similar time results. However, IDA^* may solve instances that A^* can not solve due to its modest memory requirements. Also, GenomeSEARCH-A^* performs better than GenomePLAN with the help of restrictions that may violate the optimality.

Chapter 4

Median Genome Problem

Another widely studied topic in comparative genomics is the median genome problem. Roughly, the problem is given the whole genomes of three species, finding the genome of their common ancestor. Not only this problem is used to infer ancestral information but also can be used for constructing phylogenies for more than three species.

Actually, this problem has an intuitive counterpart in geometry: Given three points A, B, C on a plane (or corners of the triangle $\triangle ABC$), another point X is called a *Fermat point* or *geometric median* if $|AX| + |BX| + |CX|$ is minimum. The problem of finding X is called *Fermat's problem*. Problem is named after mathematician Pierre de Fermat since it is first proposed by Fermat in a letter to Evangelista Toricelli who solved it ([38]). Similar reasoning applies for the median genome problem. If we consider space of all possible genomes of same length instead of $\mathbb{R}^2$ with a suitable distance measure, then the median genome problem is to find a genome of which total distance to the initial genomes is minimum.

Undoubtedly, distance measure should be defined precisely to define the problem in genomes space. There are two biologically relevant and widely used distance functions in this space. One of them is called the *breakpoint distance* and other is called the *event distance* function. The first one relies on the number of breakpoints between two genomes and other the one measures the minimum number of rearrangement events between two genomes. Again, we consider inversions and transpositions. As in the genome rearrangement problem case, we consider genomes that are circular, monochromosomal and without duplicate genes.

4.1 Problem Definition

As in the genome rearrangement problem case, we represent monochromosomal cyclic genomes without duplicate genes as GCSPs. Similarly, transpositions and inversions are represented as signed permutations. Let us first define the distance function.

Definition 11 (Distance function)**.** *Let X be a set. Then, a function $d : X \times X \to \mathbb{R}_+$ is called a distance function or a metric on X if for all $x_1, x_2, x_3 \in X$:*

- $d(x_1, x_2) = 0$ *if and only if* $x_1 = x_2$
- $d(x_1, x_2) = d(x_2, x_1)$
- $d(x_1, x_3) \leq d(x_1, x_2) + d(x_2, x_3)$

Now, we will define two metrics on the set of GCSPs as in [14]. The first one is called the breakpoint distance and the second one is called the event distance function.

Definition 12 (Breakpoint distance)**.** *The breakpoint distance between two GCSPs G_1 and G_2 is number of breakpoints between them. It is denoted by $d_{BP}(G_1, G_2)$.*

Definition 13 (Event distance)**.** *Event distance between two GCSPs G_1 and G_2 is the minimum number of transpositions and inversions to transform G_1 into G_2 (i.e., the solution of the genome rearrangement problem for G_1 and G_2. It is denoted by $d_E(G_1, G_2)$.*

For practical reasons, we use d as a placeholder for d_{BP} and d_E from now on. Now, we can define the median genome problem [14].

Definition 14 (Median genome problem)**.** *Let G_1, G_2, G_3 be three GCSPs of the same length n. Then, the median genome problem is to find a GCSP G of length n such that $d(G, G_1)+d(G, G_2)+d(G, G_3)$ is minimum. Then, G is called the median of G_1, G_2, G_3. The sum $d(G, G_1) + d(G, G_2) + d(G, G_3)$ is called the median distance for G_1, G_2, G_3 with respect to G.*

If we choose d as d_E, this problem is called the *Ancestral Median Problem (AMP)*[14] and if we choose d as d_{BP} it is called the *Breakpoint Median Problem (BMP)*[15]. BMP is computationally easier to cope with because number of breakpoints between two GCSPs can be computed in polynomial time where calculating only transposition event distance is shown to be NP-Hard as we explained previously (see Section 3.2). However, BMP has

some drawbacks: There may be more than one candidate for median of three GCSPs and the breakpoint distance gives us no further information for deciding on which one is the gene configuration of the biological common ancestor of the corresponding three genomes. Consider $G_1 = [(1, 2, -4, -3, 5)]$, $G_2 = [(1, -4, -3, -2, 5)]$ and $G_3 = [(1, 2, 3, 4, -5)]$. If we solve BMP for them, we can find one of four possible ancestors which are G_1, G_2, G_3 and $[I] = [(1, 2, 3, 4, 5)]$. But, if we solve AMP with reversals only, the ancestor is only $[I]$.

Let G_1, G_2, G_3 be three arbitrary GCSPs and G be their median with respect to some distance function d. Let $k = d(G, G_1) + d(G, G_2) + d(G, G_3)$. In this part, we provide an upper bound and a lower bound on k with respect to the pairwise distances of G_1, G_2 and G_3 not depending on the distance function.

Firstly, we show a well-known lower bound on k using the triangle inequality assumptions on the distance function (Definition 11). We know that

$$d(G, G_1) + d(G, G_2) \geq d(G_1, G_2)$$

$$d(G, G_1) + d(G, G_3) \geq d(G_1, G_3)$$

$$d(G, G_2) + d(G, G_3) \geq d(G_2, G_3)$$

Summing these three inequalities we obtain:

$$d(G, G_1) + d(G, G_2) + d(G, G_3) \geq \frac{d(G_1, G_2) + d(G_1, G_3) + d(G_2, G_3)}{2}$$

Now, using the minimality of k we show an upper bound on k. Consider G_1, without loss of generality. The GCSP G_1 is a valid candidate as the median G_1, G_2, G_3 if $d(G_1, G_2) + d(G_1, G_3)$ is minimum. Therefore, for any median G for G_1, G_2, G_3,

$$d(G_1, G_2) + d(G_1, G_3) \geq d(G, G_1) + d(G, G_2) + d(G, G_3). \tag{4.1}$$

Same reasoning applies to G_2 and G_3. Then, we obtain:

$$d(G_1, G_3) + d(G_2, G_3) \geq d(G, G_1) + d(G, G_2) + d(G, G_3) \tag{4.2}$$

$$d(G_1, G_2) + d(G_2, G_3) \geq d(G, G_1) + d(G, G_2) + d(G, G_3) \tag{4.3}$$

By summing the equations (4.1), (4.2), (4.3) we obtain:

$$\frac{2}{3}(d(G_1,G_2)+d(G_1,G_3)+d(G_2,G_3)) \geq d(G,G_1)+d(G,G_2)+d(G,G_3).$$

Consequently,

$$\frac{2}{3}(d(G_1,G_2)+d(G_1,G_3)+d(G_2,G_3)) \geq k \geq \frac{d(G_1,G_2)+d(G_1,G_3)+d(G_2,G_3)}{2}. \quad (4.4)$$

4.2 Related Work

In this section, we summarize related work on AMP and BMP separately.

BMP is the easier problem since breakpoint distance can be calculated for all genome types in linear time. BMP was first introduced in [62]. For signed and unsigned permutations, problem is shown to be NP-Complete in [53]. However, for multichromosomal linear genomes, BMP has $\mathcal{O}(n^3)$ solution [71]. For the case of GCSPs, problem is NP-Hard as shown in [15]. An exact solution algorithm to problem for signed permutations exists by reducing it to Traveling Salesman Problem [62]. There is also a 7/6 approximation algorithm [54]. There is a software system called BPAnalysis, that solves BMP based on algorithms in [63] for linear genomes without duplicate genes by inversions, transpositions and translocations.

AMP may have variations with respect to the chosen rearrangements. If we consider only inversions for the AMP, the median genome problem is proven to be NP-Hard [19]. Moreover, the problem is even APX-Hard [9]. AMP with transpositions only is shown to be NP-Complete [5]. There is a software system called MGR[14] that can be used to solve AMP. Although its main purpose is to create phylogenies for arbitrary number of genomes, its main algorithm is based on AMP problem. It can handle circular or linear genome models. However, it solves AMP considering reversals only since there exists $\mathcal{O}(n)$ algorithm for computing reversal distance. Another tool to be considered for AMP is called GRAPPA. Initially it was only considering reversals as rearrangements but later an extension is made for transpositions [79]. Moreover, it contains a special algorithm [70] for solving AMP in GCSPs using reversals.

4.3 Modeling Median Genome Problem As a Search Problem

We modeled the median genome problem as a search problem as follows:

States The search states are characterized by sets of three GCSPs. Let s be a state and G_1, G_2, G_3 be its GCSPs. Then, we denote s as a set $s = \{G_1, G_2, G_3\}$. The state space is denoted by $\mathcal{S}$ in this section.

Successor function Now, let us explain successor function $\delta : \mathcal{S} \to \mathcal{S}$. Note that, unlike the successor function we introduced in Section 3.3 for the genome rearrangement problem, δ does not map a state to a set of states but only to a state. Let $s = \{G_1, G_2, G_3\}$ be a state. Then, we define two sets of states

$$\mathcal{G}_{inv}(s) = \{\{G'_1, G'_2, G'_3\} | G'_i = inv(l_1, l_2) \circ G_i \text{ for exactly one } 1 \leq i \leq 3 \\ \text{and some labels } l_1, l_2 \in G_i \text{ and } G'_j = G_j \text{ for } j \neq i\}$$

and

$$\mathcal{G}_{tr}(s) = \{\{G'_1, G'_2, G'_3\} | G'_i = trp(l_1, l_2, l_3) \circ G_i \text{ for exactly one } 1 \leq i \leq 3 \\ \text{and some labels } l_1, l_2, l_3 \in G_i \text{ and } G'_j = G_j \text{ for } j \neq i\}.$$

$\mathcal{G}_{inv}$ denotes set of GCSPs triples obtained by applying an inversion to exactly one GCSP of s whereas $\mathcal{G}_{tr}$ denotes set of GCSPs triples obtained by applying a transposition to exactly one GCSP of s. Then, $\delta(s)$ is in $\mathcal{G}_{inv} \cup \mathcal{G}_{tr}$. Informally, we chose the next state among the states which are obtained by applying a single inversion or a transposition to one of GCSPs of the current state.

To choose which GCSP to apply a rearrangement in order to generate the next state $\delta(s)$, we order GCSPs of s according to the following evaluation function $f_s : s \to \mathbb{N}$. We compute the total pairwise distance of a GCSP in s to other two GCSPs in s. More formally, our evaluation function f_s for $1 \leq i \leq 3$ is calculated as:

$$f_s(G_i) = \sum_{1 \leq j \leq 3} d(G_i, G_j).$$

For instance, $f_s(G_1) = d(G_1, G_2) + d(G_1, G_3)$. The idea is to choose a GCSP G in s such that $f_s(G)$ is minimum. We choose the GCSP which is the furthest away from

other two GCSPs because we assume that median should be closer to the pair which is the closest each other.

Without loss of generality, assume that $f_s(G_1) \geq f_s(G_2) \geq f_s(G_3)$. Then, $\delta(s)$ is defined as follows:

- If, for some labels $l_1, l_2, l_3 \in G_1$, there exists a transposition $trp(l_1, l_2, l_3)$ such that, for $G_1' = trp(l_1, l_2, l_3) \circ G_1$ $d(G_1', G_2) < d(G_1, G_2)$ and $d(G_1', G_3) < d(G_1, G_3)$, or there exists an inversion $inv(l_1, l_2)$ such that for $G_1' = inv(l_1, l_2) \circ G_1$, $d(G_1', G_2) < d(G_1, G_2)$ and $d(G_1', G_3) < d(G_1, G_3)$, then $s' = \{G_1', G_2, G_3\}$.

- Else if, for some labels $l_1, l_2, l_3 \in G_2$, there exists a transposition $trp(l_1, l_2, l_3)$ such that, for $G_2' = trp(l_1, l_2, l_3) \circ G_2$, $d(G_2', G_1) < d(G_2, G_1)$ and $d(G_2', G_3) < d(G_2, G_3)$, or there exists an inversion $inv(l_1, l_2)$ such that for $G_2' = inv(l_1, l_2) \circ G_2$ $d(G_2', G_1) < d(G_2, G_1)$ and $d(G_2', G_3) < d(G_2, G_3)$, then $s' = \{G_1, G_2', G_3\}$.

- Else if, for some labels $l_1, l_2, l_3 \in G_3$, there exists a transposition $trp(l_1, l_2, l_3)$ such that, for $G_3' = trp(l_1, l_2, l_3) \circ G_3$, $d(G_3', G_2) < d(G_3, G_2)$ and $d(G_3', G_1) < d(G_3, G_1)$, or there exists an inversion $inv(l_1, l_2)$ such that for $G_3' = inv(l_1, l_2) \circ G_3$, $d(G_3', G_2) < d(G_3, G_2)$ and $d(G_3', G_1) < d(G_3, G_1)$, then $s' = \{G_1, G_2, G_3'\}$.

- Else, for some labels $l_1, l_2, l_3 \in G_1$, there exists a transposition $trp(l_1, l_2, l_3)$ such that for $G_1' = trp(l_1, l_2, l_3) \circ G_1$, $d(G_1', G_2) + d(G_1', G_3) < d(G_1, G_2) + d(G_1', G_3)$, or there exists an inversion $inv(l_1, l_2)$ such that for $G_1' = inv(l_1, l_2) \circ G_1$, $d(G_1', G_2) + d(G_1', G_3) < d(G_1, G_2) + d(G_1', G_3)$, then $s' = \{G_1', G_2, G_3\}$.

Here is the idea behind $\delta(s)$: We start from the furthest GCSP and try to find a rearrangement which gets this GCSP closer to other two GCSPs. If we can not find such a rearrangement from the furthest, then we try to find such rearrangements from other two GCSPs in the order imposed by f_s. If no such rearrangement exists from the other two GCSPs, we apply a rearrangement to the furthest GCSP such that it gets closer to one of other two GCSPs.

Goal Test Given a state s with three GCSPs G_1, G_2, G_3 and a distance function from set of all pairs of GCSPs to $\mathbb{N}$, s is a goal state if and only if $d(G_1, G_2) = 0$, $d(G_1, G_3) = 0$ and $d(G_2, G_3) = 0$.

Once we model the median genome problem as a search problem, we can decide for the search strategy. Here, we developed an algorithm based on the Greedy Best-

First Search (GBFS) strategy. For a state s with GCSPs G_1, G_2, G_3 we determined $f(s) = h(s) = d(G_1, G_2) + d(G_1, G_3) + d(G_2, G_3)$.

Our algorithm for solving the median genome problem is described in Algorithm 4:

Algorithm 4 Median Algorithm

Input: Three GCSPs G_1, G_2, G_3, distance function d
Output: G: Median of input genomes

while $G_1 \neq G_2 \vee G_2 \neq G_3 \vee G_1 \neq G_3$ **do**
  /*Order GCSPs with respect to f_s */
  $G'_1 \leftarrow max_{g \in L}\{\sum_{g' \in L} d(g, g')\}$
  $G'_2 \leftarrow max_{g \in L \setminus \{G'_1\}}\{\sum_{g' \in L} d(g, g')\}$
  $G'_3 \leftarrow L \setminus \{G'_1, G'_2\}$
  if Find a rearrangement ρ s.t. $d(\rho \circ G'_1, g) < d(G'_1, g)$ for all $g \in L \setminus \{G'_1\}$ /*Try to find a rearrangement for the furthest GCSP, making it closer to other two*/ **then**
    $G'_1 \leftarrow \rho \circ G'_1$
  else if Find an event ρ s.t. $d(\rho \circ G'_2, g) < d(G'_2, g)$ for all $g \in L \setminus \{G'_2\}$ /*Else try to find a rearrangement for the second furthest GCSP, making it closer to other two*/ **then**
    $G'_2 \leftarrow \rho \circ G'_2$
  else if Find an event ρ s.t. $d(\rho \circ G'_3, g) < d(G'_3, g)$ for all $g \in L \setminus \{G'_3\}$ /*Else try to find a rearrangement for the third furthest GCSP, making it closer to other two*/ **then**
    $G'_3 \leftarrow \rho \circ G'_3$
  else/*Else find a rearrangement for the furthest GCSP, making it closer to one of others */
    $G'_1 \leftarrow \rho \circ G'_1$ where,
    $\sum_{g \in L \setminus \{G'_1\}} d(\rho \circ G'_1, g) < \sum_{g \in L \setminus \{G'_1\}} d(\gamma \circ G'_1, g)$, for all events γ
  end if
  $G_1 \leftarrow G'_1$
  $G_2 \leftarrow G'_2$
  $G_3 \leftarrow G'_3$
end while
$G \leftarrow G_1$

4.4 Theoretical Results

First, we show that Algorithm 4 terminates, then we give some results about the optimality of its solutions and lastly we analyze asymptotical time complexity of the algorithm.

Termination Our main aim here is to show the following result:

Proposition 6. *Algorithm 4 is guaranteed to terminate.*

We need the following two lemmas to prove Proposition 6:

Lemma 1. *Let G_1 and G_2 be two GCSPs of length n. There exists a finite sequence of rearrangements that transform G_1 to G_2.*

Proof. For all integers $1 \leq k \leq n$, if the orientation of k is different between G_1 and G_2, then we apply an inversion on G_1 that only changes the sign of k. Then, in finite number of steps, all labels get the same sign. Now, since all labels have the same sign, one can iteratively apply appropriate transpositions on G_1 to obtain G_2. More precisely, let (l_1, l_2) be consecutive labels in G_1 and (l_1, l_3) be consecutive elements in G_2. Then, applying $trp(l_1, l_3, l_3)$ on G_1 heals the (l_1, l_2) breakpoint. □

Lemma 2. *Let G_1 and G_2 be any two GCSPs of length n. Then, for $d = d_E$ or $d = d_{BP}$ there exists a rearrangement that reduces the distance $d(G_1, G_2)$ if $d(G_1, G_2) > 0$.*

Proof. Let us first prove this lemma for the event distance function. We know that there exists a sequence of rearrangements that transforms G_1 to G_2 by Lemma 1. Pick one of these sequences such that number of rearrangements is minimum (i.e., $d_E(G_1, G_2)$ is minimum). Since $d_E(G_1, G_2) > 0$, sequence contains at least one rearrangement. Then, applying the first rearrangement ρ to G_1 should reduce the event distance since we know $d_E(G_1, G_2) - 1$ length sequence from $\rho \circ G_1$ to G_2.

Now, we show this for the breakpoint distance function. For some $a, a', b \in \{-n, ..., -1, 1, ..., n\}$, let (a, b) be adjacent in G_1 and (a, a') be adjacent in G_2. Without loss of generality, assume that a is positive. Then, there are two cases for a': for some $c, d \in \{-n, ..., -1, 1, ..., n\}$, G_1 contains two consecutive elements $(-a', d)$ or a strip $(a', ..., c)$ where a' and c are the ends of the strip.

For the first case, consider inversion $inv(b, -a')$. Clearly, (a, b) and $(-a', d)$ are breakpoints in G_1 . $inv(b, -a')$ heals (a, b) breakpoint with (a, a'). Therefore, $d_{BP}(G_1, G_2)$ reduces at least by one.

For the second case, consider transposition $trp(b, a', c)$. Note that (a, b), clockwise adjacency of a' and counter-clockwise adjacency of c are all breakpoints of G_1. $trp(b, a', c)$ heals (a, b) breakpoint by replacing it with (a, a'). Therefore, $d(G_1, G_2)$ reduces at least by one.

Consequently, for all G_1, G_2 there exists a rearrangement that reduces $d_{BP}(G_1, G_2)$. □

Proof of the Proposition 6. Loop containing lines 2-19 terminates when all of the three GCSPs become identical to each other. So, we need to show that GCSPs become identical inside this loop (lines 16-18). Let $K = d(G_1, G_2) + d(G_1, G_3) + d(G_2, G_3)$. Observe that the three GCSPs are identical if and only if $K = 0$. We will show that if $K \neq 0$, then K decreases at each iteration.

At each iteration, Algorithm 4 executes lines 6-7 or 8-9 or 10-11 or 12-14. If it executes 6-7, it should have found a rearrangement that makes the furthest genome closer to the other two genomes. Therefore, K decreases by at least 2 in the next state. Similar reasoning applies for lines 8-9 and 10-11 and K decreases by at least 2. If none of above lines are executed Algorithm 4 executes 13-14. In these lines, it tries to find a rearrangement that makes the furthest GCSP the closest to one of the others. By Lemma 2 we know that there exists a rearrangement that makes the furthest genome closer to any other GCSP. We can just pick the one which makes them the closest. Consequently, K decreases by at least one. Considering all the cases, K strictly decreases at each iteration.

Since $K > 0$, the loop is guaranteed to terminate in finite number of steps. After the while loop terminates, rest of algorithm executes just a simple assignment on line 20. Therefore, the algorithm is guaranteed to terminate. □

Optimality Suppose that $d = d_E$.

Proposition 7. *Let G be the output GCSP of Algorithm 4 for the input GCSPs G_1, G_2 and G_3 and M be the median of G_1, G_2, G_3. Then, $\sum_{i=1}^{3} d_E(G_i, G) \leq 3 \sum_{i=1}^{3} d_E(G_i, M)$.*

We need the following lemma to prove Proposition 7.

Lemma 3. *Let G_1, G_2 and G_3 be the input GCSPs, and G be output of Algorithm 4. The number of iterations of the loop starting at line 1, is greater than or equal to $\sum_{i=1}^{3} d_E(G_i, G)$.*

Proof. Algorithm 4 picks one of the three GCSPs at each iteration, and applies one rearrangement to it until GCSPs become identical. Therefore, the number of iterations are the same as the total number of rearrangements applied to the input GCSPs. Now, pick one of the input GCSPs, say G_1, and consider the number of rearrangements applied to this GCSP by our algorithm. It should not be less than $d_E(G_1, G)$ since $d_E(G_1, G)$ is the minimum length sequence transforming G_1 to G. Therefore, the total number of rearrangements and iterations can not be less than $\sum_{i=1}^{3} d_E(G_i, G)$. □

Proof of Proposition 7. Let d_C, d_M and d_G be defined as follows:

$$d_M = \sum_{i=1}^{3} d(G_i, M)$$

$$d_G = \sum_{i=1}^{3} d(G_i, G)$$

$$d_C = d(G_1, G_2) + d(G_2, G_3) + d(G_1, G_3).$$

Without loss of generality, assume that $d_{G_1G_2}$ is the largest pairwise distance among the input GCSPs. We have shown a lower bound for d_M previously in (4.4) (Section 4.1:

$$\frac{d_C}{2} \leq d_M. \tag{4.5}$$

Although $d_{G_1G_2}$ is the longest distance, it obeys the triangle inequality. Therefore:

$$d(G_1, G_2) \leq d(G_2, G_3) + d(G_1, G_3)$$

We can add $d(G_1, G_2)$ to the both sides of the inequality and obtain:

$$d(G_1, G_2) \leq \frac{d_C}{2} \tag{4.6}$$

From (4.6) and (4.5), we obtain:

$$d(G_1, G_2) \leq d_M \tag{4.7}$$

Now, we will try to put an upper limit on d_F. By Lemma 3 we know that the number of iterations of algorithm is not less than d_G, and d_C is not less than the number of iterations since algorithm reduces d_C by 1 or 2 at each iteration, until it becomes 0. Consequently:

$$d_G \leq d_C. \tag{4.8}$$

However, we know that $d_{G_1G_2}$ is the largest pairwise distance among input genomes, therefore :

$$d(G_1, G_2) \geq d(G_1, G_2)$$

$$d(G_1, G_2) \geq d(G_1, G_3)$$

$$d(G_1, G_2) \geq d(G_2, G_3)$$

Then, from above three inequalities and (4.8), we obtain:

$$d(G_1, G_2) \geq \frac{d_G}{3} \tag{4.9}$$

From (4.9) and (4.7), we obtain:

$$d_G \leq 3d_M \tag{4.10}$$

According to (4.10), the median distance of G_1, G_2, G_3 with respect to the output of Algorithm 4 can not be more than three times the median distance of the optimal solution. □

Complexity Now, we give some results about computational complexity of the algorithm. Complexity of the algorithm depends on our choice of the distance function d. Therefore, we will prove our complexity results separately for the event distance and breakpoint distance.

Proposition 8. *If $d = d_{BP}$, then Algorithm 4 has complexity $\mathcal{O}(n^4)$ where n is the length of input and output GCSPs.*

Proof. The asymptotical time complexity of the algorithm depends on the loop running through lines 1-19. This loop runs until all GCSPs become identical. Initially, the breakpoint distance between two GCSPs can not be more than n since each GCSP has n consecutive labels. Therefore, $K = d_{BP}(G_1, G_2) + d_{BP}(G_1, G_3) + d_{BP}(G_2, G_3) \leq 3n$. As we show in Proposition 6 K decreases by at least 1 for each iteration, and the loop terminates when $K = 0$. Consequently, while loop iterates $\mathcal{O}(n)$ times.

Now, we will analyze complexity of statements inside the loop. Lines 3-5 and 16-18 take constant time. Lines 6\8\10 and 12 generate possible rearrangements. Since a transposition operates on three points on GCSPs, and any chosen triple inside a GCSP constitutes a transposition, there are $\mathcal{O}(n^3)$ transpositions. Similarly, an inversion operates on two points and thus there are $\mathcal{O}(n^2)$ inversions. Consequently, there are $\mathcal{O}(n^3)$ possible rearrangements. For each rearrangement, the algorithm checks whether this rearrangement makes two GCSPs closer. We can check in constant time the change in the number of breakpoints between two GCSPs by considering the operation points of the applied rearrangement. Consequently, lines 6\8\10\12 have $\mathcal{O}(n^3)$ time complexity.

Lines 7\9\11\13-14 take constant time since they are simple assignments and arithmetic operations. As a result, the time complexity of the while loop is $\mathcal{O}(n^3)$.

Since the loop iterates $\mathcal{O}(n)$ times and each iteration has $\mathcal{O}(n^3)$ complexity, Algorithm 4 has $\mathcal{O}(n^4)$ computational complexity. □

Now we can prove a similar result for the event distance function.

Proposition 9. *If $d = d_E$, then Algorithm 4 performs $\mathcal{O}(n^4)$ event distance calculations where n is the length of the input and output GCSPs.*

Proof. The event distance calculations are performed inside the loop running through lines 1-19. This loop runs until all genomes become equal. Initially, event distance between two GCSPs can not be more than $2n$ since we can convert one GCSP into another in $2n$ steps by applying the procedure explained in Lemma 1. Therefore, $K = d_{BP}(G_1, G_2) + d_{BP}(G_1, G_3) + d_{BP}(G_2, G_3) \leq 6n$. As we show in Proposition 6 K decreases by at least 1 for each iteration, and the loop terminates when $K = 0$. Consequently, the while loop iterates $O(n)$ times.

Now, we will analyze the statements inside the loop. The distance function is used in lines 6\8\10 and 12. These lines generate possible rearrangements. As stated in the previous proof, we can generate $\mathcal{O}(n^3)$ transpositions and $\mathcal{O}(n^2)$ inversions from a GCSP. Consequently, there are $\mathcal{O}(n^3)$ possible rearrangements. For each rearrangement, the algorithm checks whether this rearrangement makes two GCPSs closer. Therefore, we need to compute the event distance for the two GCSPs. Consequently, lines 6\8\10\12 have $\mathcal{O}(n^3)$ distance function calculation.

Since the loop iterates $\mathcal{O}(n)$ times and each iteration has $\mathcal{O}(n^3)$ event distance calculations, our algorithm has $\mathcal{O}(n^4)$ event distance calculations. □

4.5 MedianSEARCH Implementation

We implemented our algorithm as a software named MedianSEARCH. It can solve the median genome problem for both the breakpoint distance and the event distance functions.

For this algorithm, we represented GCSPs as linked lists with integer keys. With this data structure, given labels we could apply a transposition or an inversion in linear time by changing the appropriate pointers and signs of integers.

Moreover, generating all possible inversions and transpositions is easier by iterating over linked list. To explain generation of the rearrangements we need two observations.

For any GCSP G, split G into two disjoint strips s_1 and s_2 such that each strip s_i has the first element a_i and the last element b_i in the clockwise direction (see Figure 4.5) . Then, it can be observed that inverting s_1 yields the same resulting GCSP with inverting s_2. Therefore, resulting GCSP of any inversion on G can be obtained by inverting a segment of which length is not more than half of the length of G.

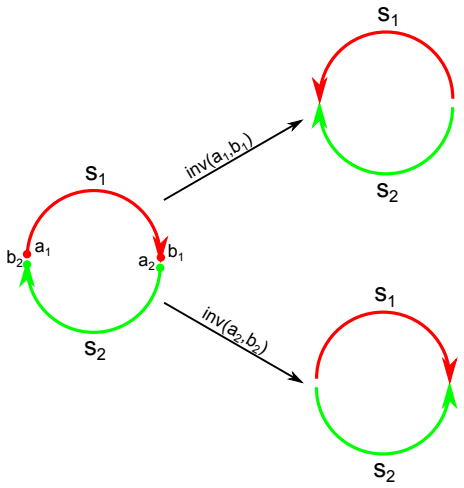

Figure 4.1: Inversion symmetries on a GCSP G. Inverting segment s_1 on G yields G_1 and inverting segment s_2 on G yields G_2. G_1 and G_2 are the mirror reflections of each other. Therefore they are the same.

Similarly, split G into three disjoint strips t_1, t_2 and t_3 such that each strip t_i has the first element a_i and the last element b_i in the clockwise direction (see Figure 4.5). Then, it can be observed that shifting t_2 to the counter-clockwise adjacency of t_1 by a transposition yields the same resulting GCSP with shifting t_3 to the counter-clockwise adjacency of t_2 by a transposition and shifting t_1 to the counter-clockwise adjacency of t_13 by a transposition. Therefore, resulting GCSP of any transposition on G can be obtained by transposing a segment of which length is not more than one-third of the length of G.

We keep two iterators it_1 and it_2 on elements of the linked list for generating inversions. Initially, both iterators are on the same element. We advance it_2 until length of the elements between it_1 and it_2 is the half length of the linked list. Each advance of it_2 generates a new inversion. it_2 stops in the half way since same GCSP can be generated later due to observation above. After the length of elements between it_1 and it_2 becomes half of the length of the linked list, we advance it_1 by one element and

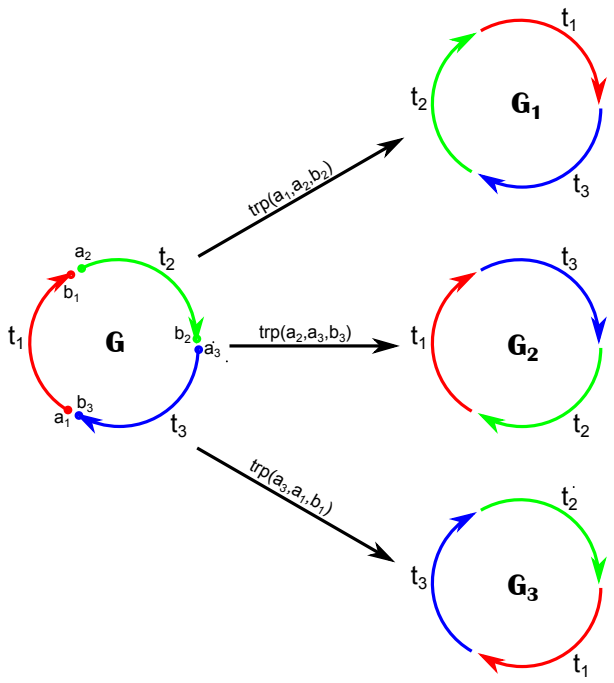

Figure 4.2: Transposition symmetries on a GCSP G. Shifting t_2 to the counter-clockwise adjacency of t_1 yields G_1. Shifting t_3 to the counter-clockwise adjacency of t_2 yields G_2. Shifting t_1 to the counter-clockwise adjacency of t_3 yields G_3. G_1, G_2 and G_3 are rotated versions of each other. Therefore they are the same.

bring it_2 to the same place with it_1. We repeat the same procedure until it_1 returns to its original position.

We keep three iterators it_1, it_2 and it_3 on elements of the linked list for generating transpositions. Initially, it_2 and it_3 points to the clockwise adjacency of it_1. We advance it_3 until length of the elements between it_2 and it_3 is the one-third of the length of the linked list. Each advance of it_3 generates a new transposition. it_3 stops on this point since same GCSP can be generated later due to the observation above. After the length of elements between it_2 and it_3 becomes one-third of the length of the linked list, we advance it_2 by one element and bring it_3 to the same place with it_2. Same procedure is repeated until it_2 comes to the counter-clockwise adjacency of it_1. Then, it_1 is advanced by one element and it_2 and it_3 is brought to the clockwise adjacency of it_1. The whole procedure is repeated until it_1 comes to its initial position.

We keep three linked lists representing GCSPs for our current state. In each iteration, we just change one of the linked-lists if an appropriate rearrangement is found.

Moreover, we only keep states of the fringe in the memory since the rearrangements applied to the input GCSPs so far are not required for the output.

The breakpoint distance between a GCSP and identity GCSP can be calculated in linear time using linked lists. It can be achieved by traversing linked list once and checking for each element whether its clockwise neighbor is the neighbor in the identity GCSP or not. Therefore, BMP problem can be solved efficiently. In order to solve AMP problem we need a tool to calculate the event distance with inversions and transpositions. To measure the event distance, we used GenomeSEARCH software (developed by Tansel Uras), which can solve genome rearrangement problem for circular monochromosomal genomes without gene duplicates with inversions and transpositions. Although GenomeSEARCH does not guarantee optimality (since it is based on a greedy depth-first search algorithm), it is efficient. This is important because our algorithm needs to calculate the event distance $\mathcal{O}(n^3)$ times at each iteration, where n is length of GCSPs of the current state.

4.6 Conclusion

In this section, we modeled the median search problem considering two distance measures between genomes: breakpoint and event distance. Then, we developed a search algorithm to solve this problem and analyzed optimality and computational complexity. We implemented this algorithm leading to a software named MedianSEARCH.

Chapter 5

Phylogeny Reconstruction

A phylogeny is a leaf-labeled tree whose leaves represent current species, internal nodes represent common ancestors and edges show genetic relations between species. Edges maybe labeled as well by some shared traits of species e.g., with the gene order in a genome or morphological character states. In this thesis, they represent evolutionary distances between species based on a metric function.

In the following, we present formal description of the problem first. Then, we give information about the recent work on the phylogeny reconstruction problem. Then, we directly present our algorithms for solving the problem. We have developed two search based algorithms based on the Greedy Best-First Search strategy. After analyzing theoretical properties of these algorithms, we explain our implementation PhyloHS.

5.1 Problem Description

Before we give a definition of a phylogeny, let us recall unrooted full binary trees.

Definition 15 (Unrooted full binary tree)**.** *An unrooted full binary tree T is an unrooted tree such that every node $n \in T$ has degree three or one.*

Then, a phylogeny is defined as follows:

Definition 16 (Phylogeny)**.** *A phylogeny for a given set G of GCSPs is an unrooted full binary tree (V, E) with the set G L of leaves and a bijection fn that maps every leaf in L to a GCSP in G. We denote a phylogeny with a tuple (V, E, G, L, fn).*

Then, we define the phylogeny reconstruction problem as follows:

Definition 17 (Phylogeny Reconstruction Problem)**.** *Let $\mathcal{G}$ denote the set of all possible GCSPs of $\{1, ..., n\}$. Given distance function $d : \mathcal{G} \times \mathcal{G} \to \mathbb{N}$, the phylogeny reconstruction problem is to find a phylogeny (V, E, G, L, fn) such that*

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

is minimum.

This problem is called *Multiple Genome Rearrangement Problem* in [14]. Intuitively, G denotes the given set of GCSPs, $V \backslash G$ denotes GCSPs of common ancestors and fn maps given genomes in the set L to their GCSPs.

As a distance function d, we consider breakpoint and event distances as we did for the median genome problem. Recall that for two GCSPs Π and Σ, $d_{BP}(\Pi, \Sigma)$ denotes the breakpoint distance and $d_E(\Pi, \Sigma)$ denotes the event distance between Π and Σ. We use d to denote d_{BP} or d_E.

Phylogeny reconstruction problem is referred as the large parsimony problem [64]. It is considered as a specific case of the full Steiner tree problem explained [32].

5.2 Related Work

The phylogeny reconstruction problem considering the breakpoint distance function was introduced in [10] and [63]. Same problem considering the reversal distance, which is the event distance with reversals only, is introduced in [45] and [14]. The former problem is NP-Complete [53] whereas the latter problem is both NP-Complete and APX-Hard [17].

There are three main approaches for solving the phylogeny reconstruction problem. The distance based approach gets a distance matrix that estimates the path lengths between pairs of the leaves of the phylogeny via given distance function, and infers a tree topology from that matrix as to minimize the total distance of the phylogeny. The distance matrix is based on the pairwise-distances of the input genomes. One of the earliest studies using this approach [12] considers the breakpoint distance and constructs phylogenies using the neighbor-joining method [60]. Later, this method has become the most popular one for building phylogenies with the distance-based approach. However, the initial pairwise-distances of the input genomes and the distance function plays important roles for obtaining optimal results using the neighbor-joining method. In

fact, the distance matrix should be *additive* or *nearly additive*[1]. Therefore, there exists some studies for estimating distance matrix based on the pairwise-distances of input genomes such as [77]. Another method to construct phylogenies using the distance based approach is UPGMA [67]. This method constructs phylogenies assuming the molecular clock hypothesis which states that the average rate of mutation is approximately same for each species in a given time period. Therefore, two genomes that stem from a common ancestor shall have the same event distance. However, the molecular clock assumption is not correct for most cases. For instance, tube-nosed seabirds have one half of the mutation rate of many other birds [57] and the mutation rates of many turtles are approximately one-eighth of the mutation rate of small mammals [2]. Therefore, constructing phylogenies with the molecular clock assumption for some species including turtles or tube-nosed seabirds may produce unrealistic scenarios.

There is a phylogeny reconstruction software called PHYLIP[31] based on these two methods.

The second approach is based on direct optimization. The idea is to construct a phylogeny from input genomes iteratively, by finding the common ancestors of some genomes at each iteration, forming partial trees and connecting them with the goal of minimizing the total distance of the phylogeny. One of the methods based on this approach tries to find the internal nodes for each tree topology and leaf replacement, by solving the median genome problems for the nodes adjacent to them and reducing the cost of the adjacent edges [65]. Recently developed algorithm EMRAE uses a similar technique [80, 81]. There are two phylogeny reconstruction software using this method: GRAPPA(considers both the breakpoint and the reversal distances) [48, 47, 46]; BPAnalysis (considers the breakpoint distance) [63]. Another method based on direct optimization constructs an initial small tree (i.e., with three nodes) and then inserts the remaining leaves into this tree iteratively by "edge-splitting" [14]. To split an edge, the median genome problem is solved for the vertices the edge and one of the remaining leaves. Therefore, to find an "optimal" edge-splitting this method solves large number of median genome problems. This method is implemented as a tool called MGR; MGR considers reversal distance only.

The third approach uses statistical methods to reconstruct phylogenies. They model rearrangements as stochastic processes and try to develop probabilistic evolution paths for genomes. One of the studies models trees, edge lengths and the number of each rearrangement (inversion and transposition) for each as random variables [41]. It assumes

that each possible tree topology is equiprobable, i.e., uniformly distributed. The length of each edge is modeled as a gamma random variable and the number of rearrangements for an edge is modeled as a Poisson distribution with mean equal to the length of this edge. Then, a phylogeny is reconstructed using a Monte Carlo Markov Chain method.

Although the first and the third approaches are computationally efficient, constructed phylogenies by the first approach does not provide genomes of the ancestors (or internal nodes of the phylogeny) and the third approach requires numerous parameters for distributions of tree topologies, the lengths of edges in the phylogeny and operation indices of the rearrangements such that small changes in these parameters results in dramatic changes in the posterior probabilities of the resulting phylogenies. Therefore, we preferred the direct optimization approach to develop solutions to the phylogeny reconstruction problem.

5.3 Algorithms

We have developed two algorithms to solve the phylogeny reconstruction problem based on the direct optimization approach.

Each of the algorithms uses our methods for solving the median genome problem as explained in Chapter 4. In the following, MEDIAN denotes one of our algorithms that solves the median genome problem depending on the distance function. We denote the median distance of three GCSPs x, y, z with respect to a median s and a distance function d that maps set of pairs of GCSPs to $\mathbb{N}$, by $w_s^d(x, y, z) = d(x, s) + d(y, s) + d(z, s)$. If s is the median of x, y, z; we denote $w_s^d(x, y, z)$ as $w^d(x, y, z)$ in short.

PhyloHS-1 Our first algorithm (Algorithm 5) is similar to that of [14] explained above: They both construct phylogenies iteratively by applying edge-splitting. However, [14] considers only reversals whereas our algorithm considers both reversals and transpositions since we have an efficient tool for solving the median genome problem with reversals and transpositions.

First, Algorithm 5 constructs a phylogeny for three input GCSPs x, y, z so that $w^d(x, y, z)$ is minimum (lines 3-8). Then, it iteratively inserts each remaining input GCSP as a leaf of this phylogeny by applying an edge-split (lines 10-15). Splitting an edge (u, v) by a vertex t replaces (u, v) with the (u, t'), (v, t'), (t, t') where t' is a new vertex having same GCSP with t. When Algorithm 5 inserts one of the remaining input GCSP into the phylogeny, it finds out which edge (u, v) of the phylogeny to split by

input GCSP, say t (line 10), by solving the median genome problem for u, v, t, and by considering $w^d(u, v, t)$: the smaller $w^d(u, v, t)$, the better to split (u, v) by t. Then, all three nodes (u, v, t) are connected to their median by new edges as explained above (lines 12-15).

Algorithm 5 PHYLOHS-1

Input: Set L of GCSPs $G_1, ..., G_k$; a distance function d that maps from set of pairs of GCSPs to $\mathbb{N}$

Output: A phylogeny (V, E, G, L, fn)

1:
2: $T \leftarrow \emptyset$
3: Find $x, y, z \in L$ s.t. $w^d(x, y, z) \leq w^d(x', y', z')$, for all $x', y', z' \in L$
4: $m \leftarrow$ MEDIAN(x, y, z)
5: $m' \leftarrow m$
6: Add m, m', x, y, z to T
7: Form edges (m, x),(m, y),(m', z) and (m', m) in T
8: Remove x, y, z from L
9: **while** $L \neq \emptyset$ **do**
10: Find $x \in L$ and edge $(y, z) \in T$ s.t. $w^d(x, y, z) - d(y, z) \leq w^d(x', y', z') - d(y', z')$, for all $x' \in L$ and $(y', z') \in T$
11: $s_{xyz} \leftarrow$ MEDIAN(x, y, z)
12: Insert x and s_{xyz} to T.
13: Form edges (x, s_{xyz}),(s_{xyz}, y) and (s_{xyz}, z) in T
14: Remove edge (y, z) from T.
15: Remove x from L
16: **end while**

We have shown that PHYLOHS-1 terminates.

Proposition 10. *For a set L of GCSPs, a distance function d,* PHYLOHS*-1 terminates.*

Proof. In Proposition 6, we have shown that Algorithm 4 is guaranteed to terminate. Therefore, we can safely state that MEDIAN function in the PHYLOHS-1 is guaranteed to terminate (line 4).

Now, consider the lines 5-9. Algorithm 5 executes these lines sequentially exactly once; they terminate.

Last thing to show to prove that the loop running through 9-16 terminates. First consider line 10. Since it is inside the loop, L is not empty. Hence, we can find $x \in L$. Moreover, T has some edges due to line 7. Although line 14 removes an edge, line 13 adds some more before it. Therefore, at each iteration of line 10, we can find $(y, z) \in T$. Again, line 10 searches for x, y, z such that $w^d(x, y, z)$ is minimum and it can be found by the well-ordering principle. Lines 11-14 perform some changes on the phylogeny.

In line 15 a GCSP is removed from L and consequently its size reduces by one. Since the execution of each line inside the loop terminates, and the number of elements of L reduces by one at each iteration, the loop eventually terminates by finite-descent. □

We have analyzed the asymptotical time complexity of Algorithm 5, in terms of calls to MEDIAN, since it is the most expensive part of the algorithm. we have given results about computational complexity of this function for various distance measures in Section 4.4, we just give a result on number of calls to MEDIAN function.

Proposition 11. *For a given set of k GCSPS,* PHYLOHS-*1 calls* MEDIAN $\mathcal{O}(k^3)$ *times.*

Proof. MEDIAN function is called in lines 4 and 11 explicitly. The former is executed only once and the latter is executed $\mathcal{O}(k)$ times since the loop containing line 11 iterates $\mathcal{O}(k)$ times.

However, there is some implicit calls to MEDIAN in PHYLOHS-1 while computing w^d. Therefore, median function is called for lines 3 and 10.

First consider line 3. It calls MEDIAN for every triple in L. Therefore, $\binom{k}{3}$ —which is $\mathcal{O}(k^3)$— calls to MEDIAN are performed in line 3.

Then, consider each iteration of line 10 and let $T_i = (V_i, E_i)$ and L_i represent the parts of T and L constructed at iteration i, respectively. At the first iteration, $|L_1| = k-3$ and $|E_1| = 4$. Therefore, line 10 performs $4(k-3)$ calls to MEDIAN. Since, $|L_{i+1}| = |L_i| - 1$ due to line 15 and $|E_{i+1}| = |E_i| + 2$ due to lines 13\14; line 10 performs $6(k-4)$ calls in the second iteration and so on. Consequently, the total number of calls to MEDIAN in line 10 is as follows:

$$\sum_{i=3}^{k-1}(k-i)(2k-2)$$

which can be bounded from above as follows:

$$\begin{aligned}\sum_{i=3}^{k-1}(k-1)(2k-2) &< \sum_{i=3}^{k-1}k(2k-2)\\ &< \sum_{i=1}^{k-1}2k(k-1)\\ &= k^2(k-1)\end{aligned}$$

Consequently, MEDIAN is called $\mathcal{O}(k^3)$ times in Line 10. Both lines, 3 and 10, call MEDIAN $\mathcal{O}(k^3)$ times. □

PhyloHS-2 We have developed another algorithm, called PhyloHS-2 (Algorithm 6), to solve the phylogeny reconstruction problem. This algorithm can be considered as an extension of PhyloHS-1 . However, PhyloHS-2 constructs a rooted full binary tree. Then we can obtain an unrooted tree from rooted tree by removing the root.

PhyloHS-2 initially perceives each input GCSP as a partial phylogeny with one root in a forest F (lines 2\3). Then, it tries to merge the phylogenies by solving the median genome problems iteratively, over x, y, z such that

- $C_1(x, y, z)$**:** x, y, z are the roots of three distinct partial phylogenies in F, or
- $C_2(x, y, z)$**:** x is the root of one partial phylogeny in F and (y, z) is an edge in another partial phylogeny in F,

until there is only one phylogeny left (lines 5-16).

At each iteration, PhyloHS-2 finds three GCSPs x, y, z in F such that $C_1(x, y, z)$ or $C_2(x, y, z)$ holds, and $w^d(x, y, z)$ is the minimum (line 5). Then, it solves the median genome problem for x, y, z (line 7 or 14) and connect them to their median by forming new edges (lines 8-12 or 15). Consequently one phylogeny remains in F that is the output of PhyloHS-2 .

Proposition 12. *For a finite set L of GCSPs,* PhyloHS*-2 terminates.*

Proof. Proof is similar to the proof of Proposition 10.

Each iteration of the while loop is guaranteed to terminate. Consider the size of F during iterations. If the block containing lines 6-12 is executed, the size of F reduces by 2 (line 11) and if the block containing lines 13-16 is executed then, the size of F reduces by 1 (line 15). Therefore, the size of F reduces by at least one at each iteration. Since $|F| \in \mathbb{N}$, $|F| = 1$ after finitely many iterations. Consequently, the while loop terminates after finitely many steps. □

We have analyzed the computational complexity of Algorithm 6 also by means of the number of calls to Median.

Proposition 13. *For a given set of k GCSPs* PhyloHS*-2 calls* Median $\mathcal{O}(k^4)$ *times.*

To prove Proposition 13 we need the following lemma:

Lemma 4. *Each iteration of the while loop in* PhyloHS*-2 increases the total number of edges in F by at most by four.*

Algorithm 6 PHYLOHS-2

Input: Set L of GCSPs $G_1, ..., G_k$; a distance function d that maps from set of pairs of GCSPs to $\mathbb{N}$

Output: A phylogeny (V, E, G, L, fn)

1:
2: $F \leftarrow \emptyset$
3: Insert each element of L to F as a tree with root itself.
4: **while** $|F| \neq 1$ **do**
5: Find a genome triple (x, y, z) such that $C_1(x, y, z)$ or $C_2(x, y, z)$ and that $w^d(x, y, z) \leq w^d(x', y', z')$, for all (x', y', z') such that $C_1(x', y', z')$ or $C_2(x', y', z')$
6: **if** $C_1(x, y, z)$ **then**
7: $s_{xyz} \leftarrow$ MEDIAN(x, y, z)
8: Find $i \in \{x, y, z\}$ s.t. $d(i, s_{xyz}) \geq d(j, s_{xyz})$, for all $j \in \{x, y, z\}$
9: Let i' and i'' be remaining elements in $\{x, y, z\} \backslash \{i\}$
10: $s_2 \leftarrow$ MEDIAN(s_{xyz}, i', i'')
11: Merge trees in F containing x,y and z by adding edges (s_{xyz}, i), (s_{xyz}, s_2), (s_2, i') and (s_2, i'')
12: mark s_{xyz} as root
13: **else if** $C_2(x, y, z)$ **then**
14: $s_{xyz} \leftarrow$ MEDIAN(x, y, z)
15: Merge trees containing x and edge (y, z) by adding (y, s_{xyz}), (s_{xyz}, z) and (s_{xyz}, x) and removing (y, z)
16: **end if**
17: **end while**
18: $T \leftarrow F[1]$

Proof. At each iteration of the while loop, either the if block containing lines 6-12 or else if block containing lines 13-15 is executed. The former increases the total number of edges in F by four due to line 11. the latter increases the number of edges in F by two due to line 15. Since these are the only lines in the while loop that effects the number of edges in F, the number of edges in F increases by at most four. □

Proof of Proposition 13. MEDIAN function is called in PHYLOHS-2 in two places: line 7 and 14. At each iteration of the while loop, either line 7 or 14 is executed due to the if-else if block. The while loop iterates at most k times, since initially $|F| = k$ and at each iteration $|F|$ reduces by at least one. Therefore, MEDIAN function is called at most $\mathcal{O}(k)$ times.

Besides this apparent calls, line 5 makes implicit calls to MEDIAN. Line 5 tries to find a GCSP triple satisfying C_1 or C_2 such that its w value is minimum among other triples satisfying conditions. Calculation of w is done via solving the median genome problem for all such triples. Therefore, MEDIAN is called multiple times at line 5. Let

us examine the number of triples satisfying C_1 and C_2 at each iteration separately.

We begin with C_1. At the first iteration, $|F| = k$. Therefore, $\binom{k}{3}$ possible triples satisfy C_1 since each root of a tree is a candidate for the triple. We know that $|F|$ reduces by at least one for each iteration. Therefore, the total number of triples satisfying C_1 is $\binom{k}{3} + \binom{k-1}{3} + ... + \binom{3}{3}$. Then, the following holds:

$$\sum_{i=3}^{k} \binom{i}{3} = \sum_{i=0}^{k} \binom{i}{3} = \binom{k+1}{4} = \frac{(k+1)k(k-1)(k-2)}{4!}$$

Therefore, MEDIAN function is called $\mathcal{O}(k^4)$ times for triples satisfying C_1.

Now, we do a similar calculation for C_2. C_2 requires choosing an edge and a root from distinct trees in F. Since we are looking for an upper bound, we can remove the restriction that the edge and the root must come from distinct trees. Initially, there can be four edges at most and k roots at least in F. Therefore, the number of triples satisfying C_2 is less than or equal to $4k$. By Lemma 4 we know that at each iteration, the total number of edges increase by at most four and $|F|$ reduces by at least one. Therefore, the number of triples satisfying C_2 in the second iteration is $8(k-1)$. Then, then total number of triples satisfying C_2 is:

$$\sum_{i=1}^{k-1} 4i(k-i+1) \leq 4k \sum_{i=1}^{k-1} i = 2k^2(k-1).$$

Therefore, the total number of triples satisfying C_2 is $\mathcal{O}(k^3)$.

Regarding the total number of calls for both C_1 and C_2, line 5 calls MEDIAN function $\mathcal{O}(k^4)$ times and this is also true for the PHYLOHS-2 algorithm since line 5 is the one which calls MEDIAN the most. □

Some Remarks: Note that both PHYLOHS-1 and PHYLOHS-2 are essentially are Greedy Best-First Search Algorithms. States can be viewed as sets of partial phylogenies. Given a set of phylogenies F in a state s, the successor function generates a set of set of phylogenies that can be obtained by all possible edge-splittings for all pairs of partial phylogenies in F or by all possible root-joinings for all triples of partial phylogenies in F. In this model, a goal state is a state with a single phylogeny of which leaves contains all of the input GCSPs. With this search model for a state s with a set of phylogenies $F = \{F_1, ..., F_k\}$, our algorithms utilize greedy best-first search strategy with $h(s) = \sum_{F_i \in F} D(F_i)$ where D is defined as in Definition 17.

5.4 PhyloHS

We implemented our phylogeny reconstruction algorithms in C++. The implementation called PhyloHS . PhyloHS utilizes GenomeSearch, developed by Tansel Uras, to calculate the event distances.

To increase the efficiency of the algorithms, we used linked-lists to represent GCSPs to be consistent with MedianSEARCH described in Section 4.5. Since our phylogeny reconstruction algorithms call frequently the Median function of which implementation is MedianSEARCH which represents GCSPs with linked-lists, GCSPs can be easily passed as parameter to MedianSEARCH using linked-lists.

To implement PhyloHS-1 , we maintained a heap to store possible edge-splitting events. Each element of the heap contains a GCSP representing the median GCSP of three GCSPs, a pointer to those three GCSPs and the median distance of those GCSPs. The heap is constructed with respect to the median distances. Therefore, finding the edge-split that leads to the minimum median distance takes constant time and inserting a new element takes proportional time to size of the heap.

PhyloHS-1 also utilizes the heap to reduce the number of calls to the Median. New elements (medians) are inserted to the heap by calling Median in two places: after constructing initial phylogeny (after line 8) and after adding an element of L to the existing phylogeny (after line 15). Basically, new elements are inserted to the heap after new edges are formed in the phylogeny such that all possible medians that can be formed by edge-splitting between the remaining elements of L an and the newly formed edges.

Although all calls to the Median in line 3 still have to be done, there is no need to call Median in line 10 anymore since the heap that returns the minimum median distance valued GCSP triples along with their median and median distance. Therefore, we need to consider calls to the Median in lines 3, 8 and 15 to find out the total number.

Line 3 still calls the Median $\binom{k}{3}$ times as before, where k is the initial length of L (see Proposition 10). Since there are four new edges in the phylogeny and $k-3$ elements are left in L, line 8 calls the Median $4(k-3)$ times. Therefore, total calls to the Median before the while loop does not change asymptotically.

However, using the heap changes the number of calls to the Median inside the while loop from $k^2(k-1)$ (see Proposition 10) to $2(k-4)(k-3)$. At iteration i, 4 new edges are formed and L contains $k-i-3$ elements since initial phylogeny is constructed with

three elements from L and one element is joined to the phylogeny after each iteration. Therefore, there are $4(k-i-3)$ calls to the MEDIAN after i^{th} iteration. In total, the while loop iterates $k-4$ times (since $|L|$ decreases by one at each iteration due to Proposition 10) and the PHYLOHS-1 calls the MEDIAN

$$\sum_{i=1}^{k-4} 4(k-i-3) = 2(k-3)(k-4)$$

times inside the while loop. This reduces the number of calls to the MEDIAN inside the loop from $\mathcal{O}(k^3)$ to $\mathcal{O}(k^2)$.

For the implementation of PHYLOHS-2 , we keep the same heap structure. We represent partial phylogenies in F as a collection of nodes which contains a linked-list (representing the GCSP in this node) and two node pointers (representing two children). Besides that we represent L and F together as a vector of pairs of a linked-list and a node where the linked-list represents a GCSP in L and node-pointer represents the partial phylogeny with root this GCSP. This partial phylogenies contain a single node which are elements of L at the beginning. As phylogenies merge, we connect phylogenies via node-pointers inside the nodes and remove elements with an empty phylogeny. This data structure enables us to access an element in L and the corresponding partial phylogeny together.

Usage of the heap structure reduces number of calls to the MEDIAN function asymptotically from $\mathcal{O}(k^4)$ (see Proposition 13) to $\mathcal{O}(k^3)$ for some input set L of k input GCSPs.

Using the heap, Algorithm 6 could execute line 5 without calling the MEDIAN, since elements of the heap contains median of the three GCSPs and the median distance. However, a new median is inserted to the heap by calling the MEDIAN for each newly created edge, newly formed root and for all possible root triples containing this newly formed root at each iteration. Inside the while loop of Algorithm 6, either the block between lines 7-12 or the block between lines 14-15 are executed. The former block merges three phylogenies, forms a new root and three new edges whereas the latter block merges two trees, generates a new root node and forms three new edges.

Consider the i^{th} iteration of the while loop in Algorithm 6. There are at most $4i$ edges in F, at most 4 of them are formed in this iteration due to Lemma 4 and $|F| \leq k-i$ due to proof of Proposition 13. Therefore, there are at most $4i$ new medians are added to the heap due to all possible edge-splittings of the newly formed root by

the block between lines 14-15, $4(k-i)$ new medians are added to due to all possible edge-splittings of the newly formed four edges by the block between lines 14-15, and $\binom{k-2}{3}$ new medians are added due to all possible three root joinings containing the newly formed root by the block between lines 7-12. Since we know that the while loop in Algorithm 6 iterates $k-1$ times at most (see proof of Proposition 13), at most

$$\sum_{i=1}^{k-1} \binom{k-i}{2} + 4(k-i) + 4i = \binom{k}{3} + 4k(k-1)$$

medians are inserted to the heap in total. Therefore, total calls to the MEDIAN is $\mathcal{O}(k^3)$.

5.5 Experimental Results

We performed experiments with two real data sets:*Metazoan* and *Campanulaceae*. *Metazoan* data set contains 11 monochromosomal, circular genomes with 36 genes without gene repetition. This data set contains mitochondrial genomes of two nematodes, an annelid, three mollusca, two arthropods, two echinoderms and a chordate (human). *Campanulaceae* data set contains 13 monochromosomal, circular genomes with 105 genes without gene duplication. This data set contains chloroplast genomes of species from "hare bell" or "bellflower" family.

We tested PHYLOHS-1 considering the event distance and the breakpoint distance. PHYLOHS-2 is implemented with the event distance measure.

Metazoan Phylogenies

- The phylogeny computed by PHYLOHS-1 (event distance) is shown in Figure 5.1. The phylogeny can be represented in Newick format as follows: (((((((Human,Katharina tunicata),Lumbricus terrestris),(Cepaea nemoralis, Albinaria coerulea)),(Asterina pectinifera, Paracentrotus lividus)),(Onchocerca volvulus, Ascaris suum)),Drosophila yakuba),Artemia franciscana)

- The phylogeny computed by PHYLOHS-1 (breakpoint distance) is shown in Figure 5.2. The phylogeny can be represented in Newick format as follows: (Ascaris suum, ((Drosophila yakuba, (((((((Artemia franciscana, Paracentrotus lividus),Cepaea nemoralis),Lumbricus terrestris),Asterina pectinifera),Albinaria coerulea),Onchocerca volvulus),Katharina tunicata)),Human))

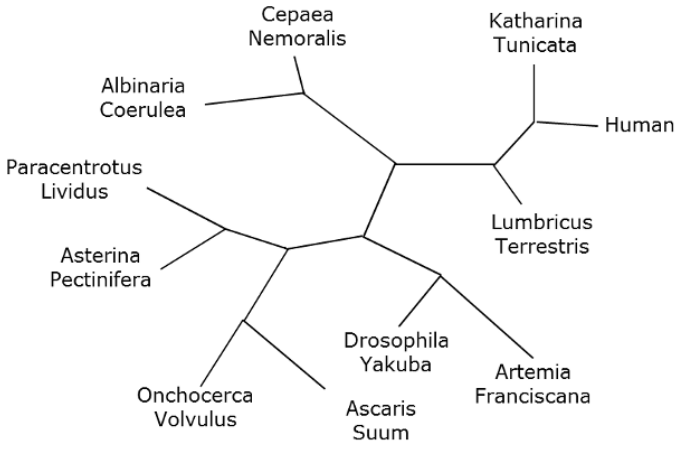

Figure 5.1: Metazoan phylogeny constructed by PhyloHS-1 using the event distance

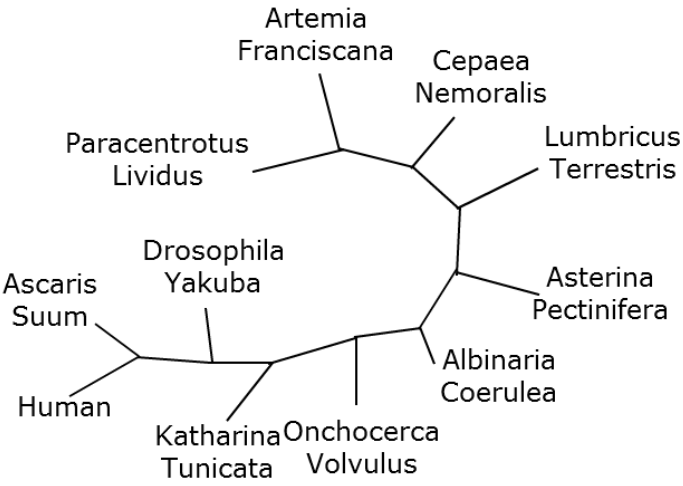

Figure 5.2: Metazoan phylogeny constructed by PhyloHS-1 using the breakpoint distance

- The phylogeny computed by PhyloHS-2 (event distance) is shown in Figure 5.3. The phylogeny can be represented in Newick format as follows:((Human, Drosophila yakuba),(((Artemia franciscana, Asterina pectinifera),(Onchocerca volvulus, (Ascaris suum, Paracentrotus lividus))),(((Albinaria coerulea, Katharina tunicata), Lumbricus terrestris), Cepaea nemoralis)))

Let us compare these results with the other phylogenies represented in the literature.

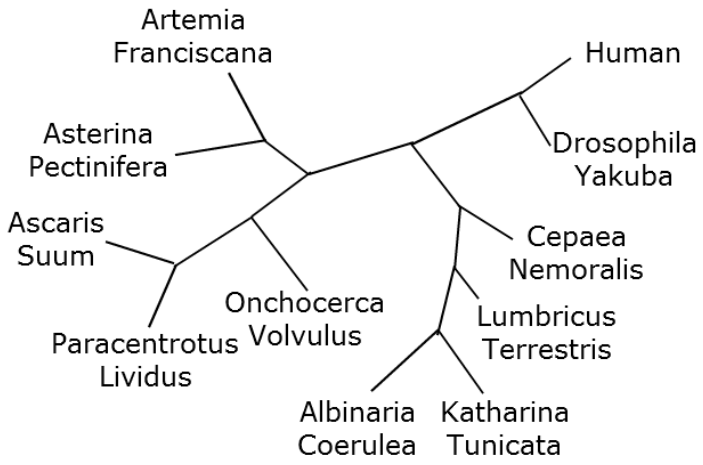

Figure 5.3: Metazoan phylogeny constructed by PhyloHS-2 using the event distance

Table 5.1 compares whether species of same phylum are collected correctly:

For this data set, another important issue is grouping of a Mollusc called Katharina Tunicata (KT). None of the algorithms could group it with other Mollusca in the data set. Uras [73] and Blanchette(1)[13] groups KT with an annelid. Bourque's method firstly combines other mollusca with annelid and then they are combined with KT [14]. Blanchette(2) firstly goups chordate with arthropods, then they are combined with KT [13]. PhyloHS-1 with the event distance combines KT with chordate. PhyloHS-1 with the breakpoint distance groups KT with a chordate an annelid and nematode. PhyloHS-2 combines KT with another mollusc.

Campanulaceae Phylogenies

- The phylogeny computed by PhyloHS-1 (event distance) is shown in Figure 5.4. The phylogeny can be represented in Newick format as follows:((Trachelium, (Campanula, (Adenophora, (((Tobacco, Platycodon), Codonopsis), Cyananthus)))), ((Symphyandra, (Wahlenbergia, Merciera)),(Legousia, (Triodanus, Asyneuma))))

- The phylogeny computed by PhyloHS-1 (breakpoint distance) is shown in Figure 5.5. The phylogeny can be represented in Newick format as follows: (Platycodon,

Table 5.1: Comparison of phylogenies on *Metazoan* data set with recent results with respect to correct matching of the species in the same phylum. PhyloHS-1 (a) refers to PhyloHS-1 with the event distance, PhyloHS-1 (b) refers to PhyloHS-1 with the breakpoint distance. Blanchette (1) and (2) refers to Figures 4.a and 4.b in [13] respectively.

Algorithm	Echinodermata	Nematoda	Arthropoda	Mollusca
PhyloHS-1 (a)	✓	✓	✓	X
PhyloHS-1 (b)	X	X	X	X
PhyloHS-2	X	X	X	X
Bourque [14]	✓	✓	X	X
Blanchette (1) [13]	✓	✓	✓	X
Blanchette (2) [13]	✓	✓	✓	X
Uras [73]	✓	✓	✓	X

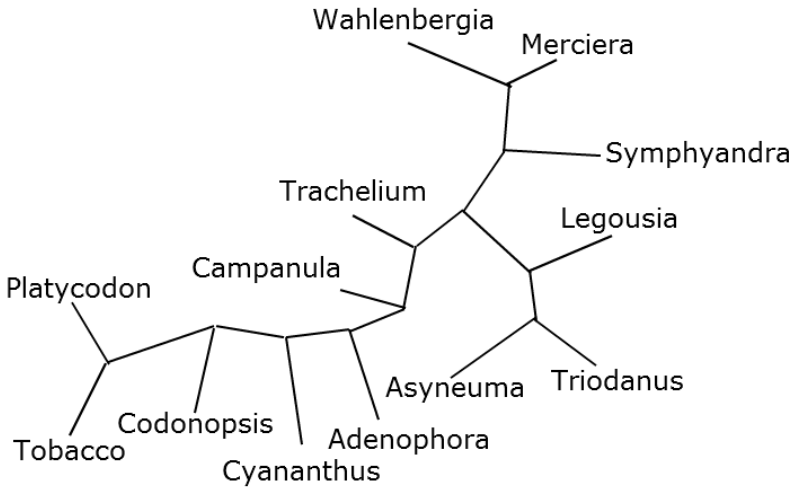

Figure 5.4: Campanulaceae phylogeny constructed by PhyloHS-1 using the event distance

((Asyneuma, Triodanus), (Legousia,((Trachelium,(((((Symphyandra, Codonopsis), Wahlenbergia), Tobacco), Merciera), Cyananthus)), (Campanula, Adenophora)))))

- The phylogeny computed by PhyloHS-2 (event distance) is shown in Figure 5.6. The phylogeny can be represented in Newick format as follows: ((Trachelium, (Campanula, Platycodon)), ((Symphyandra, Codonopsis), (Adenophora, ((Tobacco, (Wahlenbergia,Asyneuma)), ((Legousia, (Cyananthus, Triodanus)), Merciera)))))

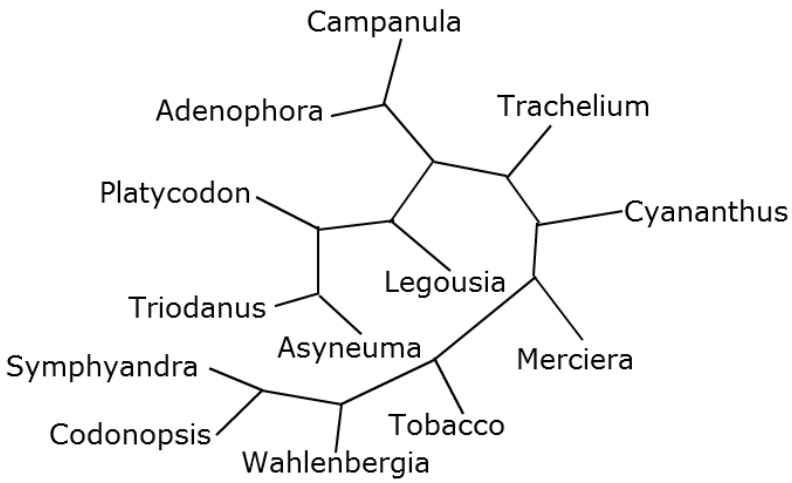

Figure 5.5: Campanulaceae phylogeny constructed by PhyloHS-2 using the breakpoint distance

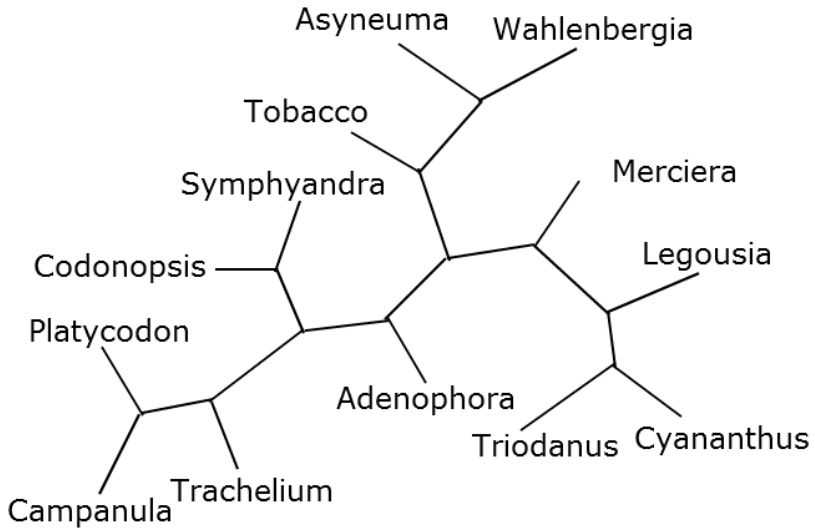

Figure 5.6: Campanulaceae phylogeny constructed by PhyloHS-1 using event distance

Let us compare these results with the other phylogenies represented in the literature.

For Campanulaceae data set [14] and [26] give exactly same trees. However, tree of [27] is constructed via sequence analysis and it is slightly different than first two. [74] is mostly in accordance with first two.

PhyloHS-1 with the event distance groups Wahlenbergia and Merciera together; Legousia, Asyneuma, Triodanus together and Codonopsis, Cyananthus Platycodon and Tobacco together; Trachelium, Campanula and Adenophora together as in the [14],

[26] and [73]. However, grouping of Symphyandra, with Trachelium, Campanula and Adenophora does not occur like in other algorithms.

PhyloHS-1 with the breakpoint distacne could only group Campanula and Adenophora together; and Legousia, Asyneuma and Triodanus together. Rest of the tree is not in accordance with [14], [26] or [27]. However, grouping of Legousia, Asyneuma and Triodanus is more similar to [14] and [26] than [73].

PhyloHS-2 groups Wahlenbergia and Asyneuma together;Codonopsis, and Symphandra together; and Triodanus and Cyananthus together. Location of Tobacco, Campanula, Trachelium and Platycodon do not match with [14], [26] or [27].

5.6 Summary of Contributions

We introduced two algorithms PhyloHS-1 and PhyloHS-2, to reconstruct phylogenies, utilizing the two algorithms for the genome rearrangement problem and the median genome problem. PhyloHS-1 differs from PhyloHS-2 in that PhyloHS-1 constructs phylogenies by edge-splitting only, whereas PhyloHS-2 considers both edge-splitting and joining three phylogenies together from their roots.

We analyzed the complexity of these algorithms and proved their termination. We implemented these algorithms leading to software named as PhyloHS .

We tested PhyloHS with two real data sets: Mitochondrial genomes of *Metozoan* species and chloroplast genomes of species from *Campanulaceae* family. According to the results PhyloHS-1 using event distance constructs phylogenies similar to recent and mostly accepted phylogenies for both *Metozoan* and *Campanulaceae* data set.

Chapter 6

Conclusion

In this thesis, we studied three well-known computational biology problems. We have developed solution algorithms using heuristic search strategies from Artificial Intelligence. We implemented and tested algorithms with real and simulated data sets. Main contributions of this thesis can be summarized as follows:

- We modeled the genome rearrangement problem for circular monochromosomal genomes without duplicate genes using inversion and transpositions. We developed an algorithm for this problem based on the A^* search strategy. We utilized the heuristic function h as one-third of the number of breakpoints of the GCSP inside the states. We have shown that this heuristic function is monotonic and therefore our A^* search finds optimal solutions. Moreover, we introduced two restrictions R_1 and R_2 on the successor function to improve time efficiency of the algorithm. We have shown that R_1 does not violate optimality whereas, incorporating R_2 may lead to suboptimal solutions. We implemented our algorithms as a tool called GenomeSEARCH-A^* using a new data structure for representing the GCSPs which allows application of inversions and transpositions in constant time. We experimented our algorithm with simulated data sets and tested effect of our restrictions, compared different heuristic search strategies A^* and IDA^*; and compared computation time of our tool with existing software GenomePLAN.

- We modeled the median genome problem as a search problem and developed an algorithm to solve this problem for the breakpoint and the event distance functions, based on the Greedy Best-First Search strategy. This time, our heuristic function h is the total pairwise distance of the GCSPs of the state. We have shown that the algorithm terminates and it performs $\mathcal{O}(n^4)$ distance function

computations where n is the length of the input GCSPs. We implemented our algorithm representing GCSPs as linked-lists.

- We developed two algorithms PHYLOHS-1 and PHYLOHS-2 to solve the phylogeny reconstruction problem. Both algorithm utilize Greedy Best-First Strategy. This time our heuristic function is the total pairwise distances of the GCSPs in all partial phylogenies. We have shown that PHYLOHS-1 terminates and solves $\mathcal{O}(k^3)$ median genome problems where k is the number of input GCSPs. We also have shown that PHYLOHS-2 terminates and solves $\mathcal{O}(k^4)$ median genome problems. We implemented our algorithms as a software called PHYLOHS using a heap to find minimum edge-splitting or root-joinings. These improved computational complexity of PHYLOHS-2 to $\mathcal{O}(k^3)$. We tested our algorithms with two real data sets and compared resulting phylogenies with the recent work.

Future Work We have assumed for all of the three problems that rearrangements are inversions and transpositions. However, *Generalized Nadeau-Taylor Model* also involves transversions which we omitted due to increase time efficiency. In the future, we plan to incorporate transversions for solving these problems in order to obtain biologically more relevant results.

We have utilized A^* search strategy for solving the genome rearrangement problem. We have realized that excessive use of memory is the main problem while testing our algorithms. To solve this problem, similar search strategies like Memory-Bounded A^* which has modest memory requirements, can be developed and implemented.

Moreover, R_1 and R_2 restrictions that we used for the genome rearrangement problem are also implemented in GENOMEPLAN. Therefore, we plan to test GENOMEPLAN with depth-first priority search strategy and compare results with GENOMESEARCH-A^*. Furthermore, there exists another software, GENOMESEARCH, which implements a greedy algorithm to solve the genome rearrangement problem. We also plan to apply the same data set on GENOMESEARCH and compare its results with GENOMESEARCH-A^* soon.

We plan to conduct experiments using MEDIANSEARCH software by generating random data sets, which contains multi sets of three genomes, each generated by random number of inversions and transpositions from $[I]$.

We also plan to conduct experiments for comparing the quality of the solutions found by the breakpoint and the event distances. To do this, we can use a data set

generated as above and find the median distance for both solutions found by using the breakpoint and event distances.

Bibliography

[1] K. Atteson. The performance of neighbor-joining methods of phylogenetic reconstruction. *Algorithmica*, 25(2):251–278, 1999.

[2] J.C. Avise, B.W. Bowen, T. Lamb, A.B. Meylan, and E. Bermingham. Mitochondrial dna evolution at a turtle's pace: evidence for low genetic variability and reduced microevolutionary rate in the testudines. *Molecular Biology and Evolution*, 9(3):457–473, 1992.

[3] F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 116(1):123–191, 2000.

[4] D.A. Bader, B.M.E. Moret, and M. Yan. A linear-time algorithm for computing inversion distance between signed permutations with an experimental study. *Journal of Computational Biology*, 8(5):483–491, 2001.

[5] M. Bader. The transposition median problem is np-complete. *Theoretical Computer Science*, 412(12):1099–1110, 2011.

[6] M. Bader and E. Ohlebusch. Sorting by weighted reversals, transpositions, and inverted transpositions. In *Research in Computational Molecular Biology*, pages 563–577. Springer, 2006.

[7] V. Bafna and P.A. Pevzner. Genome rearrangements and sorting by reversals. In *Foundations of Computer Science, 1993. Proceedings., 34th Annual Symposium on*, pages 148–157. IEEE, 1993.

[8] A. Bergeron, J. Mixtacki, and J. Stoye. A unifying view of genome rearrangements. *Algorithms in Bioinformatics*, pages 163–173, 2006.

[9] P. Berman and M. Karpinski. On some tighter inapproximability results. *Automata, Languages and Programming*, pages 705–705, 1999.

[10] M. Blanchette, G. Bourque, and D. Sankoff. Breakpoint phylogenies. *Genome Informatics*, 1997:25–34, 1997.

[11] M. Blanchette, T. Kunisawa, and D. Sankoff. Parametric genome rearrangement. *Gene*, 172(1):GC11–GC17, 1996.

[12] M. Blanchette, T. Kunisawa, and D. Sankoff. Gene order breakpoint evidence in animal mitochondrial phylogeny. *Journal of Molecular Evolution*, 49(2):193–203, 1999.

[13] Mathieu Blanchette, Takashi Kunisawa, and David Sankoff. Gene Order Breakpoint Evidence in Animal Mitochondrial Phylogeny. *Journal of Molecular Evolution*, 49(2):193–203, 1999.

[14] Guillaume Bourque and Pavel A. Pevzner. Genome-Scale Evolution: Reconstructing Gene Orders in the Ancestral Species. *Genome Research*, 12(1):26–36, January 2002.

[15] D. Bryant. The complexity of the breakpoint median problem. *Centre de recherches mathematiques*, 1998.

[16] Laurent Bulteau, Guillaume Fertin, and Irena Rusu. Sorting by transpositions is difficult. *CoRR*, abs/1011.1157, 2010.

[17] A. Caprara. Formulations and hardness of multiple sorting by reversals. In *Proceedings of the third annual international conference on Computational molecular biology*, pages 84–93. ACM, 1999.

[18] A. Caprara. Sorting permutations by reversals and eulerian cycle decompositions. *SIAM Journal on Discrete Mathematics*, 12(1):91–110, 1999.

[19] A. Caprara. The reversal median problem. *INFORMS Journal on Computing*, 15(1):93–113, 2003.

[20] X. Chen, J. Zheng, Z. Fu, P. Nan, Y. Zhong, S. Lonardi, and T. Jiang. Assignment of orthologous genes via genome rearrangement. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 2(4):302–315, 2005.

[21] D.A. Christie. *Genome rearrangement problems*. University of Glasgow, 1998.

[22] D.A. Christie. *Genome rearrangement problems*, chapter 3. University of Glasgow, 1998.

[23] D.A. Christie and R.W. Irving. Sorting strings by reversals and by transpositions. *SIAM Journal on Discrete Mathematics*, 14:193, 2001.

[24] M. Chrobak, P. Kolman, and J. Sgall. The greedy algorithm for the minimum common string partition problem. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 84–95, 2004.

[25] G. Cormode and S. Muthukrishnan. The string edit distance matching problem with moves. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 667–676. Society for Industrial and Applied Mathematics, 2002.

[26] Mary E. Cosner, Robert K. Jansen, Bernard M. E. Moret, Linda A. Raubeson, Li san Wang, Tandy Warnow, and Stacia Wyman. An empirical comparison of phylogenetic methods on chloroplast gene order data in campanulaceae, 2000.

[27] M.E. Cosner, L.A. Raubeson, and R.K. Jansen. Chloroplast dna rearrangements in campanulaceae: phylogenetic utility of highly rearranged genomes. *BMC Evol Biol*, 4(1):27, 2004.

[28] S.R. Downie and J.D. Palmer. Use of chloroplast dna rearrangements in reconstructing plant phylogeny. *Molecular systematics of plants*, 49:14–35, 1992.

[29] E. Erdem and E. Tillier. Genome rearrangement and planning. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, volume 20, page 1139. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

[30] N. Eriksen. $(1+\varepsilon)$-approximation of sorting by reversals and transpositions. *Theoretical Computer Science*, 289(1):517–529, 2002.

[31] J. Felsenstein. Confidence limits on phylogenies: an approach using the bootstrap. *Evolution*, pages 783–791, 1985.

[32] G. Fertin, A. Labarre, and I. Rusu. *Combinatorics of genome rearrangements*, chapter 14. The MIT Press, 2009.

[33] R. Forsberg, M.B. Oleksiewicz, A.M. Krabbe Petersen, J. Hein, A. Bøtner, and T. Storgaard. A molecular clock dates the common ancestor of european-type porcine reproductive and respiratory syndrome virus at more than 10 years before the emergence of disease. *Virology*, 289(2):174–179, 2001.

[34] Q.P. Gu, S. Peng, and H. Sudborough. A 2-approximation algorithm for genome rearrangements by reversals and transpositions. *Theoretical Computer Science*, 210(2):327–339, 1999.

[35] S. Hannenhalli and P. Pevzner. Transforming cabbage into turnip: polynomial algorithm for sorting signed permutations by reversals. In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, pages 178–189. ACM, 1995.

[36] S. Hannenhalli and P. Pevzner. To cut or not to cut (applications of comparative physical maps in molecular evolution). In *Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms*, pages 304–313. Society for Industrial and Applied Mathematics, 1996.

[37] T. Hartman and R. Shamir. A simpler and faster 1.5-approximation algorithm for sorting by transpositions. *Information and Computation*, 204(2):275–290, 2006.

[38] R.A. Johnson. *Modern geometry: An elementary treatise on the geometry of the triangle and the circle*. Houghton, Mifflin company, 1929.

[39] H. Kaplan and E. Verbin. Sorting signed permutations by reversals, revisited. *Journal of Computer and System Sciences*, 70(3):321–341, 2005.

[40] P. Kolman. Approximating reversal distance for strings with bounded number of duplicates. *Mathematical Foundations of Computer Science 2005*, pages 580–590, 2005.

[41] B. Larget, D.L. Simon, and J.B. Kadane. Bayesian phylogenetic inference from animal mitochondrial genome arrangements. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 64(4):681–693, 2002.

[42] T. Leitner and J. Albert. The molecular clock of hiv-1 unveiled through analysis of a known transmission history. *Proceedings of the National Academy of Sciences*, 96(19):10752, 1999.

[43] G.H. Lin and G. Xue. Signed genome rearrangement by reversals and transpositions: models and approximations. *Theoretical Computer Science*, 259(1-2):513–531, 2001.

[44] J. Meidanis, M. Walter, and Z. Dias. Reversal distance of signed circular chromosomes. *Technical report IC–00-23*, 2000.

[45] B. Moret, A. Siepel, J. Tang, and T. Liu. Inversion medians outperform breakpoint medians in phylogeny reconstruction from gene-order data. *Algorithms in Bioinformatics*, pages 521–536, 2002.

[46] B.M.E. Moret, J. Tang, L.S. Wang, and T. Warnow. Steps toward accurate reconstructions of phylogenies from gene-order data. *Journal of Computer and System Sciences*, 65(3):508–525, 2002.

[47] B.M.E. Moret, L.S. Wang, T. Warnow, and S.K. Wyman. New approaches for reconstructing phylogenies from gene order data. *Bioinformatics*, 17(suppl 1):S165–S173, 2001.

[48] B.M.E. Moret, S. Wyman, D.A. Bader, T. Warnow, and M. Yan. A new implementation and detailed study of breakpoint analysis. In *Proc. 6th Pacific Symp. Biocomputing PSB 2001*, pages 583–594. Citeseer, 2001.

[49] J.H. Nadeau and B.A. Taylor. Lengths of chromosomal segments conserved since divergence of man and mouse. *Proceedings of the National Academy of Sciences*, 81(3):814, 1984.

[50] R.G. Olmstead and J.D. Palmer. Chloroplast dna systematics: a review of methods and data analysis. *American Journal of Botany*, pages 1205–1224, 1994.

[51] J.D. Palmer. Chloroplast and mitochondrial genome evolution in land plants. *Cell Organelles*, pages 99–133, 1992.

[52] E.P.D. Pednault. Adl: Exploring the middle ground between strips and the situation calculus. In *Proceedings of the first international conference on Principles of knowledge representation and reasoning*, pages 324–332. Morgan Kaufmann Publishers Inc., 1989.

[53] I. Peer and R. Shamir. The median problems for breakpoints are np-complete. In *Elec. Colloq. on Comput. Complexity*, volume 71. Citeseer, 1998.

[54] I. Peer and R. Shamir. Approximation algorithms for the median problem in the breakpoint model. *Comparative Genomics, Kluwer, Dordrecht*, pages 225–241, 2000.

[55] X. Qi, G. Li, J. Wu, B. Liu, and M. Palis. Sorting signed permutations by fixed-length reversals. *International Journal of Foundations of Computer Science*, 17(4):933–948, 2006.

[56] AJ Radcliffe, AD Scott, and EL Wilmer. Reversals and transpositions over finite alphabets. *SIAM journal on discrete mathematics*, 19(1):224, 2006.

[57] F.E. Rheindt and J.J. Austin. Major analytical and conceptual shortcomings in a recent taxonomic revision of the procellariiformes–a reply to penhallurick and wink (2004). *Emu*, 105(2):181–186, 2005.

[58] A. Rokas. Rare genomic changes as a tool for phylogenetics. *Trends in Ecology & Evolution*, 15(11):454–459, November 2000.

[59] S.J. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Prentice hall, 2010.

[60] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular biology and evolution*, 4(4):406–425, 1987.

[61] D. Sankoff. Edit distance for genome comparison based on non-local operations. In *Combinatorial Pattern Matching*, pages 121–135. Springer, 1992.

[62] D. Sankoff and M. Blanchette. The median problem for breakpoints in comparative genomics. *Computing and combinatorics*, pages 251–263, 1997.

[63] D. Sankoff and M. Blanchette. Multiple genome rearrangement and breakpoint phylogeny. *Journal of Computational Biology*, 5(3):555–570, 1998.

[64] D. Sankoff, C. MOREL, and R.J. CEDERGREN. Evolution of 5s rna and the non-randomness of base replacement. *Nature*, 245(147):232–234, 1973.

[65] D. Sankoff, G. Sundaram, and J. Kececioglu. Steiner points in the space of genome rearrangements. *International Journal of Foundations of Computer Science*, 7(1):1–9, 1996.

[66] Y.F. She and G.L. Chen. Parallel algorithm for computing reversal distance. In *Parallel and Distributed Computing, Applications and Technologies, 2005. PDCAT 2005. Sixth International Conference on*, pages 950–953. IEEE, 2005.

[67] P.H.A. Sneath, R.R. Sokal, et al. *Numerical taxonomy. The principles and practice of numerical classification.* 1973.

[68] A. Solomon, P. Sutcliffe, and R. Lister. Sorting circular permutations by reversal. *Algorithms and Data Structures*, pages 319–328, 2003.

[69] F. Swidan, M. Bender, D. Ge, S. He, H. Hu, and R. Pinter. Sorting by length-weighted reversals: Dealing with signs and circularity. In *Combinatorial Pattern Matching*, pages 32–46. Springer, 2004.

[70] J. Tang and B. Moret. Linear programming for phylogenetic reconstruction based on gene rearrangements. In *Combinatorial Pattern Matching*, pages 101–119. Springer, 2005.

[71] E. Tannier, C. Zheng, and D. Sankoff. Multichromosomal genome median and halving problems. *Algorithms in Bioinformatics*, pages 1–13, 2008.

[72] G. Tesler. Grimm: genome rearrangements web server. *Bioinformatics*, 18(3):492–493, 2002.

[73] Tansel Uras and Esra Erdem. Genome rearrangement: A planning approach. In *Proc. of the 24th AAAI Conference on Artificial Intelligence (AAAI'10)*, 2010.

[74] Tansel Uras and Esra Erdem. Genome rearrangement: A planning approach. In *AAAI*, 2010.

[75] M.E.M.T. Walter, Z. Dias, and J. Meidanis. Reversal and transposition distance of linear chromosomes. In *String Processing and Information Retrieval: A South American Symposium, 1998. Proceedings*, pages 96–102. IEEE, 1998.

[76] L.S. Wang and T. Warnow. Estimating true evolutionary distances between genomes. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 637–646. ACM, 2001.

[77] L.S. Wang, T. Warnow, B.M.E. Moret, R.K. Jansen, and L.A. Raubeson. Distance-based genome rearrangement phylogeny. *Journal of molecular evolution*, 63(4):473–483, 2006.

[78] S. Yancopoulos, O. Attie, and R. Friedberg. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21(16):3340–3346, 2005.

[79] F. Yue, M. Zhang, and J. Tang. A heuristic for phylogenetic reconstruction using transposition. In *Bioinformatics and Bioengineering, 2007. BIBE 2007. Proceedings of the 7th IEEE International Conference on*, pages 802–808. IEEE, 2007.

[80] H. Zhao and G. Bourque. Recovering true rearrangement events on phylogenetic trees. *Comparative Genomics*, pages 149–161, 2007.

[81] H. Zhao and G. Bourque. Recovering genome rearrangements in the mammalian phylogeny. *Genome research*, 19(5):934–942, 2009.