8-9-2014

Using Genetic Algorithm to solve Median Problem and Phylogenetic Inference

Nan Gao
University of South Carolina - Columbia

Using Genetic Algorithm to solve Median Problem and Phylogenetic Inference

by

Nan Gao

Bachelor of Computer Science
Hefei University of Technology 2005

Master of Safety of Engeerning
Tsinghua University 2008

Submitted in Partial Fulfillment of the Requirements

for the Degree of Doctor of Philosphy in

Computer Science

College of Engineering and Computing

University of South Carolina

2014

Accepted by:

Jijun Tang, Major Professor

Jianjun Hu, Committee Member

John R. Rose, Committee Member

Song Wang, Committee Member

Lingrong Wu, Committee Member

Lacy Ford, Vice Provost and Dean of Graduate Studies

Acknowledgments

I would like to thank the generous support from my advisor Dr. Jijun Tang for being a knowledgeable and inspiring mentor to me; Dr. John R. Rose, Dr. Jianjun Hu, Dr. Wang Song and Dr. Rongling Wu for being my graduation committee members and providing their knowledge and guidance. I would also thank my parents and my boyfriend for their endless love and patience to help me to fight with my disease.

Abstract

Genome rearrangement analysis has attracted a lot of attentions in phylogenetic computation and comparative genomics. Solving the median problems based on various distance definitions has been a focus as it provides the building blocks for maximum parsimony analysis of phylogeny and ancestral genomes. The Median Problem (MP) has been proved to be NP-hard and although there are several exact or heuristic algorithms available, these methods all are difficulty to compute distant three genomes containing high evolution events. Such as current approaches, MGR[1] and GRAPPA [2], are restricted on small collections of genomes and low-resolution gene order data of a few hundred rearrangement events. In my work, we focus on heuristic algorithms which will combine genomic sorting algorithm with genetic algorithm (GA) to produce new methods and directions for whole-genome median solver, ancestor inference and phylogeny reconstruction.

In equal median problem, we propose a DCJ sorting operation based genetic algorithms measurements, called GA-DCJ. Following classic genetic algorithm frame, we develop our algorithms for every procedure and substitute for each traditional genetic algorithm procedure. The final results of our GA-based algorithm are optimal median genome(s) and its median score. In limited time and space, especially in large scale and distant datasets, our algorithm get better results compared with GRAPPA and AsMedian.

Extending the ideas of equal genome median solver, we develop another genetic algorithm based solver, GaDCJ-Indel, which can solve unequal genomes median problem (without duplication). In DCJ-Indel model, one of the key steps is still sorting

operation[3]. The difference with equal genomes median is there are two sorting directions: minimal DCJ operation path or minimal indel operation path. Following different sorting path, in each step scenario, we can get various genome structures to fulfill our population pool. Besides that, we adopt adaptive surcharge-triangle inequality instead of classic triangle inequality in our fitness function in order to fit unequal genome restrictions and get more efficient results. Our experiments results show that GaDCJ-Indel method not only can converge to accurate median score, but also can infer ancestors that are very close to the true ancestors.

An important application of genome rearrangement analysis is to infer ancestral genomes, which is valuable for identifying patterns of evolution and for modeling the evolutionary processes. However, computing ancestral genomes is very difficult and we have to rely on heuristic methods that have various limitations. We propose a GA-Tree algorithm which adapts meta-population [4], co-evolution and repopulation pool methods In this paper, we describe and illuminate the first genetic algorithm for ancestor inference step by step, which uses fitness scores designed to consider co-evolution and uses sorting-based methods to initialize and evolve populations. Our extensive experiments show that compared with other existing tools, our method is accurate and can infer ancestors that are much closer to true ancestors.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Gene Sequence Data

There are variety of data types used in phylogenetic inference problem. The data are typically represented in the form of matrix and each row represents taxa and each column represents the individual characters. Early phylogeneticists primarily used morphological characters which stand for the physical attributes of the organisms. In theory, inference can be described as any characters that are inherited and are able to change over time. Biological sequence data is used mostly in modern phylogenetic inference field, such as DNA, RNA or protein sequences [5, 6] representing the same gene in different of organisms. Sequence data are composed of a series of characters which are referred to as bases or nucleotides in the case of DNA (A, C, G and T) and RNA (A, C, G and U) sequence, and amino acids such as the protein sequence (A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W and Y)[7]. The reason for using for sequence data is primarily due to the ease to process and build model: with the process of sequence evolution many characters can be easily gathered and used in the tractability of statistically modeling .

Gene order and genome rearrangements with gene order

Based on the ordering and strand of genes on a chromosome, biologist may represent each chromosome by an ordering of signed genes. In 1936, Dobzhansky and Sturtevant

gave the first paper in which they used the degrees of disorder between the segments of genes in two genomes to measure the distances between different organisms [8, 9]. A genome is the collection of genes in the order in which they are placed along one or more chromosomes, so gene-order data enables the reconstruction of evolutionary events far back in time [10, 11].

A genome can be represented by a ordering (circular or linear) set of n genes $\{g_1, g_2, \cdots, g_n\}$. Each gene is assigned with an orientation that is either g_i as positive or $-g_i$ as negative. The adjacent is defined as an ordered pairs of (g_i, g_j) or ($-g_j$, $-g_i$), which means g_i and g_j appear consecutively in one genome.

Suppose a genome (G) with liner ordering

$$g_1, g_2, ..., g_{i-1}, g_i, g_{i+1}, ..., g_{j-1}, g_j, g_{j+1}, ..., g_n$$

an inversion between g_i and g_j, which $i \leq j$, generates the genome with liner ordering

$$g_1, g_2, ..., g_{i-1}, -g_j, -g_{j-1}, ..., -g_i, g_{j+1}, ..., g_n$$

A transposition on this linear ordering genome G has an influence on three points: i,j and k, where $i \leq j$ and $k[i\,,\,j]$, inserting the interval ordering segment g_i, g_{i+1}, ... ,g_j immediately after gk. It produces the genome

$$g_1, g_2, ..., g_{i-1}, g_{j+1}, ..., g_k, g_i, g_{i+1}, ..., g_j, g_{k+1}, ..., g_n$$

An inverted transposition is simply defined as a transposition followed by an inversion, also called a transversion.

Distance Computation Measurements

We use distance [12, 13] to describe the degree of disorders between two genomes (G_1 and G_2). The distance is defined as the minimum number of evolution events required to transform one gene order into the other. Based on different domestic events, there are several distance measurements.

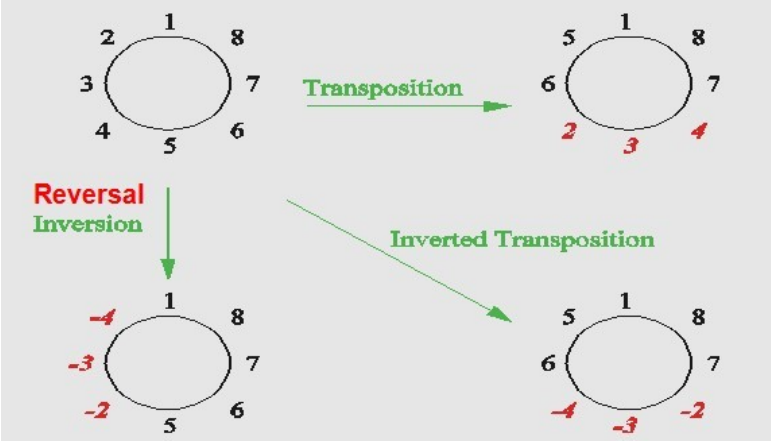

Figure 1.1 Genome Rearrangements Examples

- The Breakpoint Distance[14]. The breakpoint distance presents the minimum total number of breakpoints (adjacencies present in one genome but absent in the other) between two genomes.

- The Inversion Distance [15]. The inversion distance (inversions are the most documented hypothesized mechanisms of evolution events) measures the minimal inversions needed to transform one genome into the other. Based on the breakpoint graph, HP algorithm determines how to calculate inversion distance (Fig 1.1). Hannenhalli and Pevzner [16] proved that the inversion distance between two signed permutations of n genes is given by:

$$n - \#cycles + \#hurdles + (1\,if\,notpresent; 0\,otherwise) \tag{1.1}$$

  Moret et al. implement a tool called GRAPPA which can give both breakpoints and inversion distances. Later, MGR, proposed by Bourque and Pevzner, based on GRAPPAs distance computation parts, focuses on multi-chromosomal genomes.

- Double-Cut and Join Distance. Yancopoulos et al. [58] proposed an universal double-cut-and-join (DCJ) operation. A double-cut-and-join (DCJ) operation

occurs when two breaks are appeared in the chromosomes of a genome and the cut fresh telomeres are reconnected to form a new single adjacency.

DCJ subsumes all other rearrangement events such as inversions, translocations, fissions and fusions. We use an adjacency graph to determine the DCJ distance between two genomes. Figure 1.1 shows an example of a DCJ adjacency graph. Genome A and genome B are supposed to have equal gene orders. In the graph, we use a vertex to stand for every adjacency and telomere in A, and repeat the same process for B. For each gene terminal, we draw an edge connecting the two vertices which contain the same terminal in A and B. After that, telomere vertices have a degree of one and adjacencies have a degree of two, so the graph is composed of paths and cycles. A DCJ event can modify at most two adjacencies or telomeres, so the potential results of a single event are separating one cycle into two, removing a cycle from an existing path, connecting the ends of a path of even length to form a cycle, or splitting a path of even length into two paths of odd length.

Recently DCJ operation attracts lots of attentions because its ease of computation and less chromosome structure constraints, so it provides a simpler and unifying model for genome rearrangement.

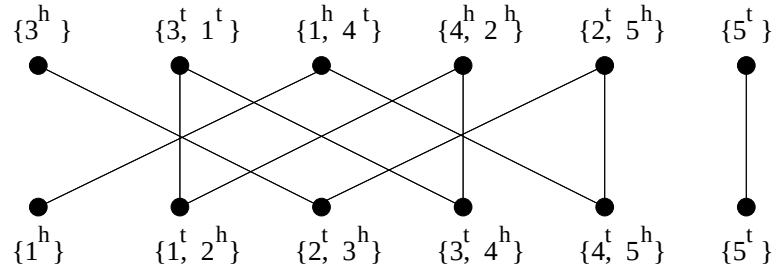

Figure 1.2 Adjacency graph and DCJ distance of two genomes $G_1 = (3, -1, -4, 2, 5)$ and $G_2 = (1, 2, 3, 4, 5)$. The number of cycles C is 1, the number of paths I is 2, the DCJ distance is $N - (C + I/2) = 3$.

Phylogeny Reconstruction and Median Problem

In 1982, the quantitative analysis of gene order data was first addressed with the introduction of the chromosome inversion problem [**?**]. Phylogenetic focuses on the study of evolutionary relationships among groups of organisms (e.g. species, populations)[17, 18] based upon similarities and differences in their physical and/or genetic characteristics [19, 20, 21].A phylogenetic tree or evolutionary tree is often described as a binary tree. Its leaves are the given set of descendants organisms and internal nodes stand for extinct ancestors connected by different lengths of edges. Gene-order data has the ability to study the whole-genome, so it is a good solution to resolve the well-known gene tree vs. species tree problem [22, 23, 24].

The methods of phylogenetic reconstruction of gene-order data, currently, have two main directions [25]: one is Distance-based methods and the other is Parsimony-based methods [26, 27].

One of effective Distance-based algorithms for generating phylogenetic trees is called Neighbor Joining [28]. The brief idea of NJ is: constructing a matrix containing the evolutionary distance between each pair of a given set of genomes; then iteratively combining the best pair of candidate leaves into a single distance matrix row and column as a new node which represents an internal node in the phylogenetic tree connected with the best pair. After that the rest of the distance matrix is modified with respective to the new node. The original purpose for Neighbor Joining was developed with sequence based distances, but it is completely compatible and commonly used with gene order in earlier years.

Maximum parsimony is another strategy for generating phylogenetic trees specific to gene order and content data. The method assumes that most gene order evolutionary events are very unlikely and the tree that uses the fewest number of events to evolve from ancestors to descendants is most likely to be the true evolutionary tree with true tree topology. A lot of maximum parsimony algorithms have been

developed and implemented using different distance measurements such as inversion, DCJ, and others [The abcs of mgr with dcj]. The common solution idea of these methods is based on solving the median of three problem which is defined as given three genomes (connected or not), finding the medial genome which can minimize the distance between itself and the other three genomes. The goal of parsimony method is to minimize the total number of evolutionary operations between ancestor and descendant along a phylogenetic tree. The median problem is the critical part in ancestor genome finding and phylogenetic reconstruction problem.

We assume without loss of generality there are four genomes G_1, G_2, G_3 and genome G_m. The median problem on three genomes is to find G_m that minimizes the median score

$$d(G_1\,, G_m) + d(G_2\,, G_m) + d(G_3\,, G_m) \tag{1.2}$$

Based on the triangle inequality, we can get the prefect median score is

$$\lceil \frac{d(G_1, G_2) + d(G_1, G_3) + d(G_2, G_3)}{2} \rceil \tag{1.3}$$

which is used as the lower bound for a median problem. However, solving even the simplest case of median problem when the number of genomes is three is NP-hard for most distance measurements [29, 30, 31].

Our research topic follows the parsimony-based methods not only because the currently accurate and efficient tools, such as GRAPPA, MGR[1] are in this area. MGR solves the median problem by applying "good" events to the three initial genomes, one at a time in round robin order. After repeating doing that procedure, a single "prefect" genome (the median genome) is generated. GRAPPA [32], by using a bounded exhaustive search, compares different tree scores with multiple medians spanning trees until it converges into a stable tree structure and that is the "perfect" evolutionary tree. All of these methods can adopted with different distance methods, such as inversal and DCJ distance.

There are several exact solutions to solve median problem classified by different distance measurements (inversion, breakpoint and DCJ distances) [33, 34, 35]. Among them, the best one is the DCJ median solver proposed by Xu and Sankoff [35]. The adjacency graph method is used by Xu in AsMedian to solve DCJ distance. It can be expanded from two to three genomes, so it provides a large amount of valuable information about the potential optimal structures for the median genome. Though the ASMedian solver could outstandingly reduce the computational costs of median searching, it yet runs very slow when the genomes are distant, exhausts large amount of spaces and even looses some accuracy. Meanwhile, there are some heuristic methods such as GASTS and SCJ. Although they have the ability to fast solve median problem with high-resolution genomes in a relative stable amount of storage spaces, the accuracy is still not so good with distance increasing. Besides that, the distance measurements they used are not so universal (inversion and break point-like SCJ distance).

Genetic Algorithm

Genetic algorithms, or GAs, are based upon evolutionary principles of natural selection, mutation, and survival of the fittest (Dulay, 2005) [36]. A genetic algorithm maintains amount of population which consists of a set of potential solutions and after some generation evolve one or several fittest solution(s).

In general, GA will go through four steps: initial pool, crossover, mutation and get result(s) [37]. If the last three operations stay constant all over the period of the algorithm, a genetic algorithm is the simple one. Crossover procedure helps exchange the genetic information of creatures and living organisms. Mutation procedure randomly change the creatures' genetic information. Fitness selection reproductive off-spring of adapted creatures and pass their good genetic information in their environment.

A good fitness function is one of the critical keys to a successful genetic algorithm.

We use it to evaluate potential solutions and it usually decides the evolution direction. When developing a genetic algorithm, firstly, we should consider what the format is for real solution and how each solution can be represented in the algorithm. The simplest one is a string of bits. To initial population of potential solutions, we could use random methods (usually can not get good results) or algorithms in some specify fields. Each member of the population is evaluated and recorded using predefined fitness function. Crossover and mutation are two important steps to generate new generation and pass good hereditary material to the next generation. The two procedures repeat till the algorithm reaches some stopping criteria, such as the best fitness score reaches the predefined value or a certain number of generations have been produced.

A number of researchers have investigated non-binary genetic algorithms theoretically and some of them have been successfully used in applications [37]. Bhattacharyya and Koehler [38] and Leung et al. [39] first addressed non-binary genetic algorithms with cardinality 2^v. The non-binary model has different crossover and mutation operators with normal binary model which fits the definition of a generalized binary string operation. So in our genetic algorithm, for particular data format and event constrain rules, we use non-binary genetic algorithm operations which can reflect specific evolutionary events. However, it will need us to find and embed suitable operations to substitute simple genetic algorithm procedures.

In 1996, Matsuda presents the first GA method to solve the phylogeny inference problem. The ability of GAs to find near-optimal solutions quickly in the face of complex data makes them ideal candidates for the problem of phylogenetic inference, especially when many taxa are included or complicated evolutionary models are applied [40]. Later PHYML (Guindon and Gasquel, 2003) [41] and RAxML (Stamatakis et. al, 2005) [**?**, 42] have been developed. Both programs implement enhancements those reduce the burden of branch-length optimization in the evaluation of new topologies, especially in distant data sets. In 2006, Zwickl developed GARLI

(Genetic Algorithm for Rapid Likelihood Inference)[43] which allows ML phylogenetic searches to be performed on datasets consisting of thousands of sequences [44]. Using GA to solve phylogenetic reconstruction with sequence data has been studied for a long time [45]. However, GA with gene order data has not been touched yet. In this proposal, we first explore this field and make some steps forward in some research directions.

Co-evolution

McKelvey (1997) has discussed that evolution of organizations cannot be understood independently from the simultaneous evolution of the environment. He addressed a co-evolutionary perspective to study organization adaptation. The co-evolutionary paradigm can be broadly classified into two main categories which are competitive co-evolution and cooperative co-evolution respecting to the relationship among sub-populations in that ecology system. All of the two types of evolutionary approaches need to consider several design issues such as problem decomposition, subpopulation size, parameter interactions and so on. In our algorithm, each internal node gives its own contributions in the whole species evolutionary history, so we only consider cooperative co-evolution, in which each subpopulation collaborate to solve the whole problem and exchange information within each other during the evolutionary process.

1.2 Research Contribution

The research presented in this work contributes to using genetic algorithm to solve median problem and phylogeny reconstruction in three major ways:

1. Development of an equal genomes median solver: GaDCJ. We developed a Genetic algorithm median solver using DCJ distance measurement. Our GA-based method uses genomic sorting to generate initial population and find offspring

by crossover and mutation procedures. Using GA, we have the ability to extend optimal median solution space with limited space and time, so it is not easy to stack at local optimal, even when the three genomes are very distant.

2. Development of an unequal genomes median solver: GaDCJ-Indel. Following the four steps of classic genetic algorithm, we propose the first genetic algorithm based median solver with unequal content genomes, but without duplications, taking into consideration of DCJ and indel operations with different event ratios. Our GaDCJ-Indel algorithm not only can efficiently give accurate results from small distance to large distance datasets, but also needs relative stable small memory spaces which is the shortcoming for most of current equal genome median solvers.

3. Development of two GA-Phylogeny algorithm: Ga_PMAG and Ga_Gasts. We propose a new method to score and infer ancestor genome structure with a fixed tree topology. to infer accurate ancestors with large distance whole-genome gene order data in a specify tree structure. Our genetic algorithm is a cooperative co-evolutionary method based on meta-population. By using good initialization methods and sorting based crossover and mutation procedures, as well as careful consideration of co-evolution, our GA-based method can reach relatively close tree scores. In future, our approaches will be adopted to include other events such as deletions, insertions and duplications.

Chapter 2

Genetic Algorithm Median Problem Solver

2.1 Introduction

With the increasing availability of fully sequenced genomes, we are now able to conduct genomic evolution study beyond the mere sequence level. Rearrangement of gene orders by operations such as reversal (also called inversion), transposition, fission, and fusion are known to be an important evolutionary mechanism.

As these events are rare, they can be used to reconstruct evolutionary histories that extend far back in time [46]. Other than reconstructing deep evolutionary histories, another important application of genome rearrangement analysis is to infer gene order within both ancestral and contemporary genomes. Such inference is valuable for identifying patterns of evolution and for modeling the evolutionary processes (e.g. hot spots of rearrangement). As a result, genome rearrangement analysis has attracted a lot of attentions from biologists, mathematicians, and computer scientists [47, 48, 49] since the pioneering papers of Sankoff [50].

Handling rearrangement events directly is mathematically very difficult: it took almost a decade to find the first polynomial algorithm that computes the reversal distance (i.e. the minimum number of reversal operations to transform one genome into another) [16], and it was just recently proved that the transposition distance is NP hard. Yancopoulos et al. [51] proposed a simplified model that used the universal double-cut-and-join (DCJ) operation to account for all rearrangement events, which cuts a chromosome at two places and rejoins the four ends of the two cut places in a

new way. Although there is no direct biological evidence for DCJ operations, these operations are very attractive because it provides a simpler and unifying model for genome rearrangement [52].

Main methods to infer ancestral gene orders are parsimony-based methods such as `GRAPPA` [53] and `MGR` [54]. The core of MGR and GRAPPA is a set of algorithms to solve the median problems of k genomes, which is to find an ancestral genome that can minimize the sum of the pair wise distances between itself and each of the k given genomes.

GA was widely used in solving many hard optimization problems, including those in computational biology [55, 56, 57]. Since genome rearrangement deals with chromosomes, evolutions and mutations, it will be natural to think that the approach of genetic algorithm can be easily adopted into solving the DCJ median problem. However, there are some major difficulties and the biggest problem is that the search space is simply too large: given genomes with N genes, the possible number of gene orders is $2^N N!$.

It poses serious questions on the major aspects of genetic algorithms:

- how should we generate the starting population?
- what is the best fitness score?
- and how to generate the next generation and pick the better one to survive?

There is a critical issue we need to consider when we adopt Genetic Algorithm: the premature convergence. There are several approaches for handling the premature issue:

- M. Srinivas, and L. M. Patnaik in 1994 proposed a paper called [58]" Adaptive probabilities of crossover and mutation in Genetic algorithm". They used

$$p_c = \frac{k_1}{(f_{max} - f^-)} \tag{2.1}$$

$$p_m = \frac{k_2}{(f_{max} - f^-)} \tag{2.2}$$

to automatically adjust the probabilities of crossover and mutation.

- YeeLeun proposed the method called Degree of Population Diversity. In this paper [59], a concept of degree of population diversity was gave to quantitatively characterize and theoretically analyze the problem of premature convergence in genetic algorithms (GAs) within the frame work of Markov chain.

- J. Andre, in his paper [60], said " to fight the premature convergence of GA, we emphasize at last two deciding alterations made to the algorithm: an adaptive reduction of the definition interval of each variable and the use of a scale factor in the calculation of the crossover probabilities."

In our problem, based on previous research results and empirical data of ourselves and other papers, the solution space of median problem is convex or is not so complex and irregular. So in our problem, I would like to use adaptive crossover and mutation probability method to avoid premature issue. I already adopted this method into our algorithm, and then I will do various experiments to analyze the results.

2.2 Motivation

The*DCJ median problem* of three genomes is specifically defined as the problem to find a median genome that minimizes the summation of distances measured by DCJ operation between the given three genomes(Figure 2.1). It has been proven that this problem is NP-hard even for three simple genomes. Because mathematically the DCJ distance is much simpler than handling the events directly, parsimony methods using DCJ median solvers outperform other methods in terms of speed and accuracy. Among all existing exact solvers, the best is `ASMedian` proposed by Xu and Sankoff [35], which uses the concept of Adequate Subgraph to decompose the problem into

smaller, more easily solved subproblems, thus significantly it can reduce total computation time. However it still runs very slowly with high gene rearrangement rate. For datasets with N genes and r (expected) number of events per edge, when the ratio of r/N is larger than 50%, all median solvers have great difficulty in finishing the analysis within hundreds of giga bytes space or after months of computation [52].

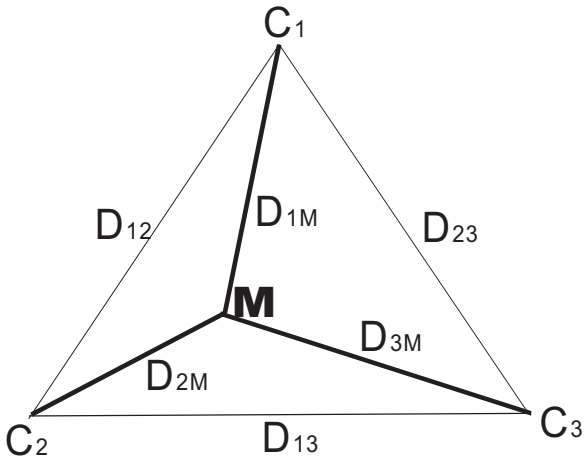

Figure 2.1 The DCJ Median Problem and Its Bounding Box.

All these facts motivated us to design a new algorithm that combines genetic algorithm (GA) with genomic sorting which has the ability to solve the DCJ median problem in limited time and space, especially in large and distant datasets.

2.3 Genetic Algorithm Median Solver with gene order data

The major difficulty of using Genetic Algorithm approach in searching a median is that the search space is simply too large: given genomes with N genes, the possible number of gene orders is $2N^N!$. It poses serious questions on the major aspects of genetic algorithms:

- how should we generate the starting population?
- what is the best fitness score?

- and how to generate the next generation and pick the better one to survive?

In this dissertation, we will present our sorting-based methods to explain and solve these problems one by one.

Initial Population Generation

The initial population has deep impact on the performance of a GA-based method. In the DCJ median problem, as the search space is very large, randomly pick some genomes as start will not work as most likely these genomes will all be far away from the desired median. Our approach is based on the following observation: given three genomes, the median genome is likely to be on the path from one of the leaf genomes to another. Although this does not readily give us a median solver as the possible number of sorting paths are very large, it does suggest a strategy to generate the initial population: for any given pair of known genomes G_i and G_j with distance d_{ij} , we will find genomes that are on the sorting path from G_i to G_j and are $d_{ij}/10, 2d_{ij}/10, \cdots, 6d_{ij}/10$ steps away from G_i(Figure **??**). Such genomes on the sorting path can be easily generated using the DCJ sorting algorithm described earlier.

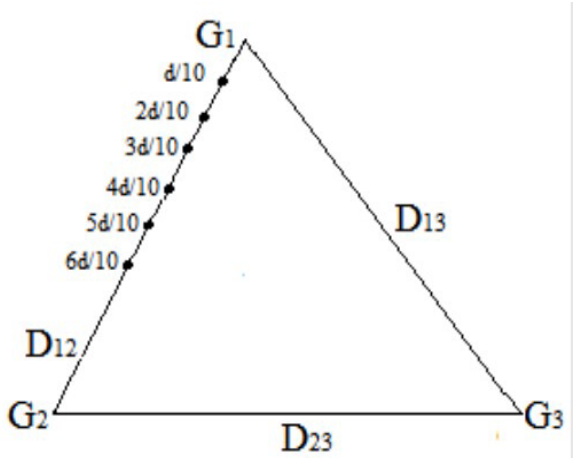

Figure 2.2 The 6 Sorting Steps Along the Path From G_1 to G_2

To obtain enough diversity, we generate 50 genomes per sampled step, resulting in 1, 800 genomes in the initial population (there are 6 pairs of genomes as the starting genomes are different). As seen in the experimental results, this strategy is quite effective and sometime only a few steps are required to converge onto very accurate results.

Selection and Fitness Function

A critical parameter to be carefully tuned in GA is the selection pressure which is the process of selecting the best individual(s) for the next generation, governed by the fitness function [61]. In the DCJ median problem [62], an obvious choice is to use the median score as the fitness function, and the one with a lower score will have better fitness. In practice, we use the following fitness score: given N genes and the perfect median score Sbest, if a genome G has median score S, its fitness score is defined as

$$FG = N - (S - S_{best}) \tag{2.3}$$

As the DCJ distance between any two genomes cannot exceed N, the above fitness function guarantees that the one closer to the median will have better fit, and the score is ranged between 0 and N.

In GA [63, 64], an important step is to select individuals into the candidate pool who can produce offspring those having better fitness score should have higher chance to get into the pool and pass its good genes to the next generation. There are some classical mechanisms to select these individuals, based on different situations. For example, in Roulette Wheel Selection, each individual has its probability of being selected in the candidate pool as its fitness score divided by the sum of fitness scores from all individuals. Truncation Selection selects the top $1/p$ individuals and each will be copied p times into the pool.

In the DCJ median problem, the range of the fitness function is very small ($[0\,,N]$) compared to the possible number of genomes ($2^N N!$), thus many individuals will have the same fitness score. This situation will get worse when the search approaches the end where the best candidate may have a fitness score that is only a few numbers away from the worst. Furthermore, two individuals with very different ordering of genes may have the same fitness score, but the difference of orderings may result in very different search patterns: some may quickly converge to a good solution as they have genes better grouped while the others may not converge at all.

To overcome this problem, we adopt a hybrid approach of these traditional selection methods. We first select the top 10% individuals and reproduce them (without change) into the next generation, as individuals with good genomic structure is hard to find and we want to preserve that as long as possible.

We then put every individual in the remaining 90% into the candidate pool and give them equal chance of being selected to produce offspring. To ensure better genes are passed down, we devised the following crossover and mutation operations that are based again on genomic sorting. Figure 2.3 gives an example of what's scenarios of each stage in one DCJ sorting sequence.

Crossover

Crossover is used for two selected individuals to exchange genetic material and produce offspring. In some genetic algorithms, this procedure can be as simple as exchange blocks of the encoding strings.

However, in the DCJ median problem, since each individual is represented as a gene order and each gene should appear exactly once in one individual, such exchange will result in invalid offspring. For example, if we exchange the last two genes of $g_1 = 1, 2, 3, 4, 5$ and $g_2 = -2, 3, -5, -1, 4$, the resulted offspring will be $g_1 = 1, 2, 3, -1, 4$ and $g_2 = -2, 3, -5, 4, 5$, both violate our requirements and are invalid. It is not an

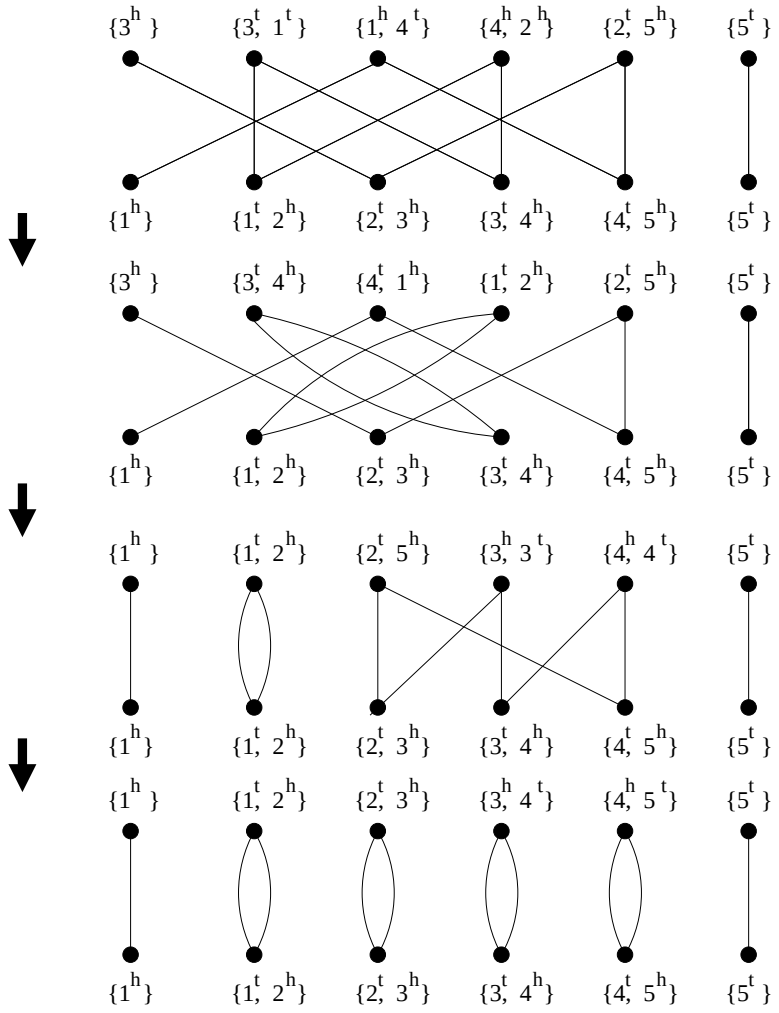

Figure 2.3 Adjacency graphs of each stage of one DCJ sorting sequence that transforms (3 -1 -4 2 5) to (1 2 3 4 5).

easy task to convert them back into valid gene orders.

The method we choose for crossover is based on sorting genomes in `DCJ`. First, we pick two parents (P_1 and P_2) from the candidate pool and compare their fitness score F_1 and F_2. Assume P_2 has better fitness score than P_1, we will generate two child genomes C_1 and C_2. C_1 is generated by selecting a genome which is on the sorting path from P_1 to P_2 (the better one) and is m (randomly chosen) steps away from P_1. In other words, the new child obtains genetic material from both parents by applying DCJ operations on one parent, with respect to the one with better fitness. We do

not generate C_2 by sorting from the worse to the better, as from our experiments, this can easily destroy the good group of genes and leads to bad solutions. Instead, we generate C_2 as the direct copy of P_2 (which has better fitness), given the better genome a higher chance to pass its good structures in the future generations.

Both children will then undergo the mutation procedure described below with the expectation that better offspring may be found.

Mutation

Mutation is used to maintain genetic diversity from one generation of a population to the next. Mutation happens randomly in an evolutionary history and can extend the range of a search. Proper mutations are needed so that `GA` can avoid local minimal by preventing the population from becoming too similar to each other.

In the DCJ median problem, an individual can be mutated by applying a random number of DCJ operations to an individual. However, there are two questions to answer: how many operations are required and which operations should we choose to apply?

From Figure 2.4, one can estimate the distances from the median genome (M') to the three given ones by the following simple calculations:

$$d_{1M'} = \frac{d_{12} + d_{13} - d_{23}}{2} \tag{2.4}$$

$$d_{2M'} = \frac{d_{12} + d_{23} - d_{13}}{2} \tag{2.5}$$

$$d_{3M'} = \frac{d_{23} + d_{13} - d_{12}}{2} \tag{2.6}$$

Although the actual distances may be different from these estimated values, the above estimations are a good indicator that how close a genome is to the median. If one genome has its three edge lengths too far way from these estimated lengths, this genome is likely to be bad and should be mutated toward a better one.

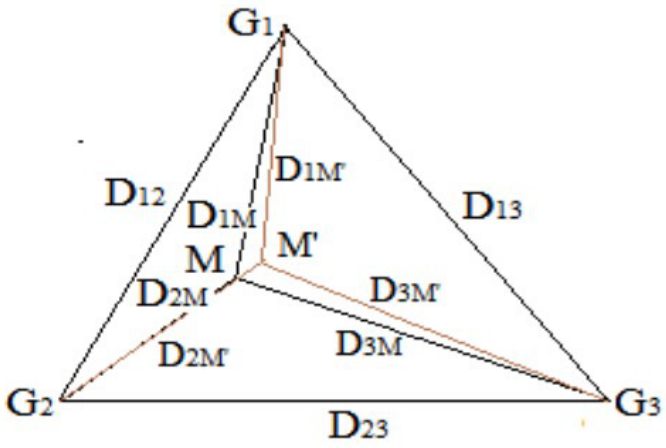

Figure 2.4 Estimate the difference between individual G_M with true G'_M ancestor using triangle inequality method

Our mutation procedure is based on the above observation. For a genome G_M from current population pool, we can compute its three edge lengths to the given genomes, and find the one which has the largest difference of the obtained and estimate lengths. That means we can estimate which one is more away from true ancestor (Figure 2.4). We then sort G some m (randomly chosen) steps closer to that given genome.

We conduct the above procedure on the two child genomes (C_1 and C_2) obtained from the crossover procedure discussed above (with parents P_1 and P_2). As a result, we get two new genomes C'_1 and C'_2. We then choose the two best from the four genomes (C_1, C_2, C'_1 and C'_2), thus maintain enough diversity and enhance the quality of individuals in the next generation.

2.4 Results

We implemented our new GA-based method in C and conducted experiments to assess its accuracy and speed. Simulation is the main approach to evaluate the quality of a phylogeny method, as the evolutionary history is known. In this paper, we conducted extensive simulations following widely used procedures. As ASMedian requires very

large amount of memory when the genomes are distant, we used a shared-memory computer with $256GB$ memory to run the experiments, thus extended the range of problems that can be solved by ASMedian not normally achievable. Although the shared-memory computer is used, each test is done on a single `CPU` with no parallelism.

Setup of Simulations

Because all existing median solvers have very good performance when genomes are close but cannot finish for distant genomes, we divided our experiments into two parts: those can be finished by the exact methods and those cannot. We only compared our new `GA` method with Xu and Sankoff's `ASMedian` solver, as it is the best for the median problem.

In our experiments, the real gene order data is so hard to get and we don't know the true ancestors and topology in history, so in this research field we always use simulation data. So we already know the correct solutions and can compare our results with them.

We tested the methods on simulated datasets of three genomes with 200 genes for each one. We generated trees with three leaves and one internal node, assigned the identity permutation on the internal node and generated the three leaves by applying rearrangement events along each edge respectively. The number of events on each edge is controlled by a birth-death process which is viewed as a good model to fit evolutionary trees. The datasets are grouped by the average edge lengths (r), which are 20, to 200 events per edge in our experiments, with the $\frac{r}{N}$ rates from 0.1 to 1.0, ranging from very easy to extremely difficult. For each r, we generated 10 datasets and averaged the results.

The maximum number of iterations for our `GA` method was set at 500 but will stop earlier if the perfect median score is met. The one genome with the lowest median score will be reported as the result. In our experiments, this maximum number is

large enough that all instances have their best genome appeared with fewer than 500 iterations.

Comparison with ASMedian

For $r \leq 60$ ASMedian is generally very fast while our method is a bit slower. However, the running time of `ASMedian` increases quickly with $r \geq 80$ and requires more than a day to finish, while our `GA` method requires no more than 30 minutes even for the most difficult ones.

The top of Table 2.1 shows the results for the median scores obtained. For $r \leq 40$, ASMedian and our method achieve the same median scores that are very close to the perfect median score. For $r \geq 40$, although the average median scores of our GA method are larger than those obtained by ASMedian, the difference is small and less than 2% even for the most difficult cases. ASMedain cannot finish any dataset with $r \geq 140$, while our method can still reach genomes with reasonable median scores, within 500 iterations and 30 minutes of computation.

For the unrooted tree defined by the three given genomes, the median genome can be used to estimate the gene order of the internal node, which is the missing ancestor. Thus the distance to the true ancestor (known in simulations) is an additional measure of the quality of median solvers. The bottom of Table 1 shows the average breakpoint distance to the ancestor for the two methods. It is very surprised to see that for almost all datasets, the medians inferred by our GA method are indeed much closer to the true ancestor compared to those inferred by the exact method ASMedian. This suggests that the sorting-based mutation and crossover procedures are very effective and preserve important genomic structures. Even for $r = 200$, the breakpoint distance between the inferred and true ancestor is fewer than 55, comparable to those achieved by `ASMedian` for a far smaller r ($r = 120$ in Table 2.2).

Convergence

A question GA faces is whether it can converge [65]. Table 2.2 shows the average and max number of generations needed to find the best solutions. It is not surprise to see that with higher number of events, the search space becomes much bigger, hence more generations are needed to have good genomes shown. It also can see that although the maximum number of generations is set at 500 in our experiments, the GA method can always find good genomes before 500 generations. The average number of iterations is indeed much small than this upper limitation, thus a better stop criteria may be desired to avoid this waste.

From the average fitness score Figure 3.5 of my experiments, we can see that there is little premature issues here because there is no fluctuation in all lines and all the average fitness scores decreased step by step. Besides that, by using simulation data sets, we know the each correct answer, so we can get the difference between the average score for each generation and the optimal score.

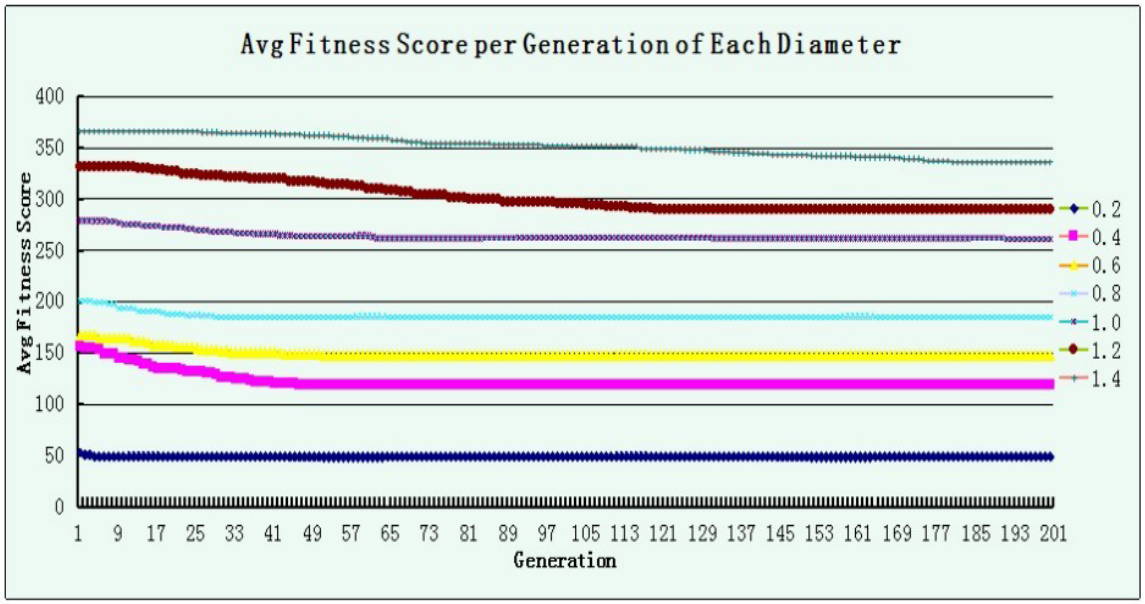

Figure 2.5 Average Fitness Score with Generation Number Increasing

Table 2.1 (Top) Comparison of median scores. (Bottom) Comparison of the breakpoint distance from the inferred median to the true ancestor. r is the averaged number of events per edge. "-"indicates that a method cannot finish.

Comparison of the median scores:

	r=20	r=40	r=60	r=80	r=100
Our GA Method	53.7	109.8	155.5	180.9	232.1
ASMedian	53.7	109.8	154.8	175.5	228
Perfect Score	53.6	109.4	152.2	173.4	210.6
	r=120	r=140	r=160	r=180	r=200
Our GA Method	247.1	279.4	287.7	281.6	309.1
ASMedian	242.3	-	-	-	-
Perfect Score	221.8	242.4	254.8	244.4	261.9

Comparison to the true ancestors:

	r=20	r=40	r=60	r=80	r=100
Our GA Method	0.3	0.4	5.0	9.9	28
ASMedian	0.4	0.3	6.3	15.6	40.7
	r=120	r=140	r=160	r=180	r=200
Our GA Method	32.7	44.9	49.2	57.5	54.9
ASMedian	50.5	-	-	-	-

Table 2.2 Number of generations to find the best genome

	r=20	r=40	r=60	r=80	r=100
Average	7.9	27.3	43	50.6	94.3
Max	21	104	108	110	201
	r=120	r=140	r=160	r=180	r=200
Average	128.6	99.4	142.8	172.2	180.4
Max	290	151	303	337	496

2.5 Discussion and Conclusions

We propose a Genetic algorithm median solver using DCJ distance measurement. Our GA-based method uses genomic sorting to generate initial population and find offspring by crossover and mutation procedures. Using GA, we have the ability to extend optimal median solution space with limited space and time, so it is not easy to stack at local optimal, even when the three genomes are very distant. Our experiments on simulated datasets shows that our GA method is very efficient and has better

speed and accuracy compared to existing methods. It also confirms the importance of sorting in solving the DCJ median problem, and our approaches can be adopted to include other events such as deletions and insertions, for which linear algorithms are available to compute the distance, to further improve the ancestral inference from genome rearrangements. However, this paper is a first attempt to use the approach of genetic algorithm in gene order analysis, it requires better strategies in selection and crossover. As we generally deal with many more genomes, we need to develop a genetic algorithm that can compute phylogenies and ancestors directly, without solving the median problem at all.

Chapter 3

Genetic Algorithm Median Solver with Insertions and Deletions

3.1 Introduction

Insertions and deletions are important components of genome evolution and should be considered in genome rearrangement algorithms. The double cut and join (DCJ) is an universal rearrangement operation introduced by Yancopoulos et al. In 2005 [66], that allows to represent most large scale evolutionary events, such as inversions, translocations, fusions and fissions occurred in genomes. If we want to measure the DCJ distance between two genomes the situation is they must have equal gene content, that means they must have exactly the same gene contents. The assumption is not in accord with true organisms' evolutionary history. Recently lots of work has been done to modify equal DCJ distance measurement to allow insertion or deletion events, or complex problems such as genome halving [67], genome liquating, and sorting an unequal content genome to the identity genome [68].

In order to deal with unequal genomes, EI-Mabrouk citeel2000recovery first addressed the edit distance for insertions and deletions by extending the results of Hannenhalli and Pevzner. In 2008, Yancopoulos and Friedberg [69] proposed an extension of the DCJ paradigm including operations performing insertions and deletions.

In 2010, Braga followed Yancopoulos's work and proposed a linear time approach to computer the DCJ-Indel distance between two genomes, in which the cost of an insertion or deletion is the same as that of a DCJ, where several consecutive markers

can be inserted or deleted in a single event [70, 71, 72].

Philip [73], in 2012, provided a simplified indel model which theoretically solve the problem in linear time directly from breakpoint graph. However, the method is not easy to realize and not practical for sorting.

3.2 Motivation

The methods for computing genomic distances and sorting operations between two genomes with unequal gene contents have attracted lots of attentions recently. Although median solvers and distance measurements with equal genomes are developed thoroughly (such as breakpoint distance, reversal distance and DCJ distance), Median solver using DCJ-Indel idea has little progressed. `GRAPPA` can only handle limited number of deletions with small gene rearrangement rate. And by now no other DCJ-Indel median solver has been addressed. In Chapter 2, we have proved the excellent performance of our genetic algorithm median solver (`GaDCJ`) in both accuracy and scale aspects theoretically and experimentally. So we want to extend our algorithm to handle deletions and insertions by using DCJ-Indel sorting algorithm. Our proposed GaDCJ-Indel algorithm can handle not only complex indel scenarios, but also large distance datasets in limited memory spaces and time.

3.3 Methods

In Chapter 2, I have in detail explained what kinds of algorithms we adopted and why we used them in each step in our GaDCJ median solver. We will extend it to handle unequal median genomes problem with insertions and deletions rearrangement events. So we will not go through all of our DCJ-Indel model again. I will describe what parts we need to modify in order to fit more complex situation with both DCJ and indels operations.

DCJ-Indel Model

Without loss generality, given a genome $A = \{a, e, x, c, d, y, b, z, w\}$ and a genome $B = \{a, b, c, d, e\}$, the two are unequal genomes. Based on the concept of adjacency graph we introduced in Chapter 2, Figure 3.1 gives us an example of an adjacency graph with two unequal genomes (A and B). In Bragas algorithm, an indel only affects the label of one adjacency by deleting or inserting contiguous markers in this label. That is when we sort A into B, the indel operations are executed by deleting all the markers only in A and inserting all the markers only in B. No classical DCJ operation is able to describe an insertion or a deletion event, so an operation in our DCJ-Indel median model is either a DCJ, an insertion or a deletion.

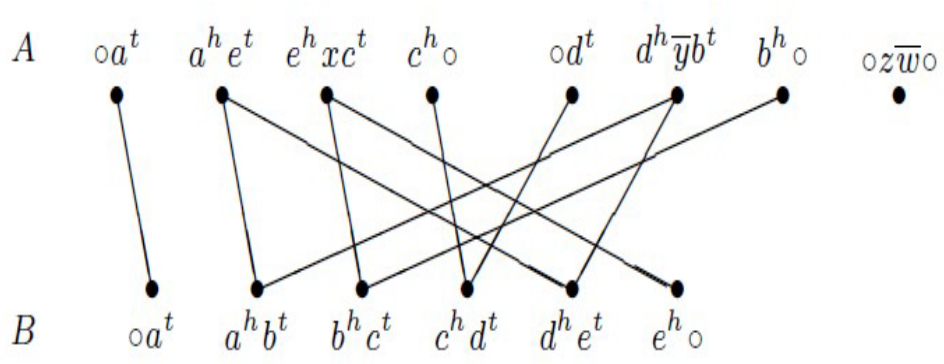

Figure 3.1 A adjacency graph of two unequal genomes: A and B

Given $l_3 \neq \epsilon$, deleting l_3 from the adjacency $r_1 l_1 l_2 l_3 r_2$ is represented as the operation

$$\rho_d = (r_1 l_1 l_2 | l_3 | r_2 \rightarrow r_1 l_1 l_2 | r_2)$$

meanwhile inserting l_3 is represented as the operation

$$\rho_i = (r_1 l_1 l_2 | r_2 \rightarrow r_1 l_1 l_2 | l_3 | r_2)$$

where l_3 stands for a substring of genomic materials in the operated genome and r_1 and r_2 are the telomeres or an extremity of a marker in the operated genome.

The DCJ-indel distance of genome A and genome B, signed as $d_{DCJ\_indel}(A, B)$, is described as the minimum number of steps of the sum of DCJ and indel operations required to sort A into B or B into A (the two values are equal).

In Bragas paper, he proposed there were two different directions can realize DCJ-indel sorting operation: one can minimize the number of DCJs and the other can minimize the number of Indels. Figure 3.2 shows an example of these two different sorting scenarios with unequal genomes. Within the same number of steps, the space of solutions of these two directions contains scenarios with different components.

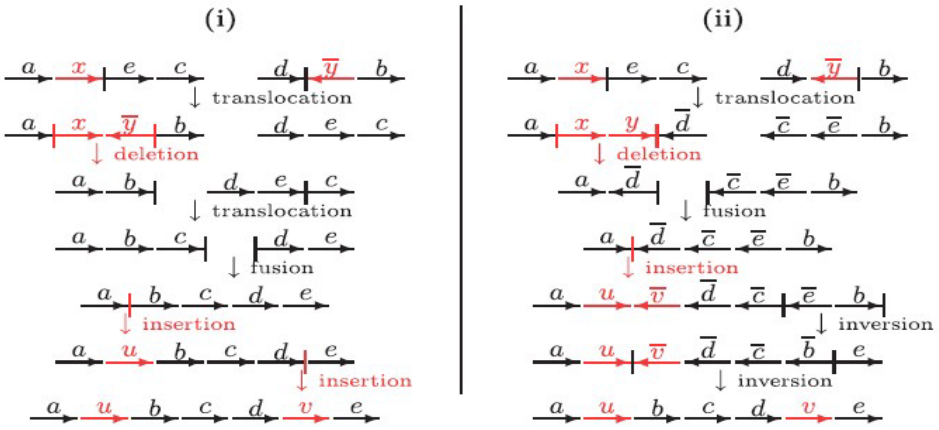

Figure 3.2 Two optimal scenarios if DCJ-Indel sorting: (i) Minimal DCJ operations. (ii) Minimal Indel operations

Fitness Function

For equal genomes, we can adopt triangle inequality formula to get the lower bound of their median score(Figure 2.1). So we can use the difference between current individual's median score with the lower bound score to adjust evolutionary direction.

$$d_{1M} = \frac{d_{12} + d_{13} - d_{23}}{2} \tag{3.1}$$

$$d_{2M} = \frac{d_{12} + d_{23} - d_{13}}{2} \tag{3.2}$$

$$d_{3M} = \frac{d_{23} + d_{13} - d_{12}}{2} \tag{3.3}$$

However, given any three unequal genomes A, B, C without duplication markers, there is no guarantee that the triangle inequality can still be held in this situation. Yancopoulos and Friedberg gave a simple example [74]. Giving three genomes: $A = a, b, c, d, e$, $B = a, c, d, b, e$ and $C = a, e$, the distance of $d_{dcj\_indel}(A\,, B) = 3$, $d_{dcj\_indel}(A\,, C) = 3$ and $d_{dcj\_indel}(B\,, C) = 1$. We can see it doesn't obey the triangle inequality formula.

In Braga's paper [74], they used a surcharge parameter k in DCJ-Indel distance to solve this problem, denoted by surcharge-triangle inequality. The modified triangle inequality is describe as

$$m(A\,, B) \leq m(A\,, C) + m(B\,, C) \tag{3.4}$$

The formulas to calculate $m(A\,, B)$,$m(A\,, C)$ and $m(B\,, C)$ are defined as:

$$m(A\,, B) = d_{dcj\_indel}(A\,, B) + k(|G_1| + |G_2| + |G_3| + |G_4|) \tag{3.5}$$

$$m(A\,, C) = d_{dcj\_indel}(A\,, C) + k(|G_1| + |G_5| + |G_6| + |G_4|) \tag{3.6}$$

$$m(A\,, B) = d_{dcj\_indel}(A\,, B) + k(|G_3| + |G_5| + |G_6| + |G_2|) \tag{3.7}$$

where the three genomes can be divided into 6 sets (3.3): $G_1\,, G_2\,, G_3\,, G_4\,, G_5\,, G_6$. $G_1 \cap G_2$ is the set of gene orders that occur only in genome A but not in genome B and $G_3 \cap G_4$ is the set of gene orders that only occur in B not in A. Analogously, we know the meaning of $G_1 \cap G_5$, $G_6 \cap G_4$, $G_3 \cap G_5$ and $G_6 \cap G_2$. In general, the surcharge-triangle inequality holds if we take $k = 3/2$.

Based on the above description, in our `GaDCJ-indel` algorithm, the fitness function on longer use classic triangle inequality function but use the surcharge-triangle inequality. However the $k = 3/2$ can not provide a tight lower bound in DCJ-Indel median problem, we only use it in the first several generations. Generally we use this

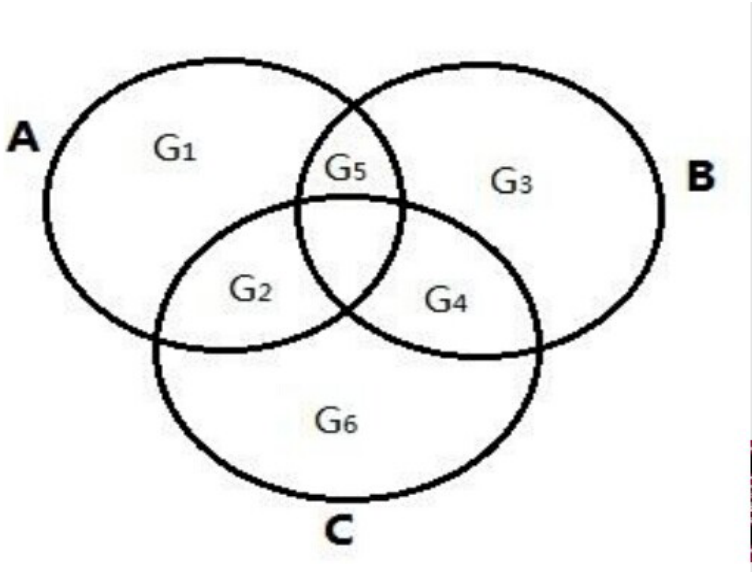

Figure 3.3 The 6 sets of G_A, G_B and G_C

formula:

$$0.25 * \#genes * diameter \tag{3.8}$$

to correct evolutionary direction in the first several generations. After that, we decrease the number of k step by step until 1 to approach optimal solution(s).

Initialization for GaDCJ-Indel algorithm

In the initialization procedure, we will generate a population pool by sorting along a path from one genome to the other. Because DCJ-Indel sorting has two different directions (minimal DCJ or minimal Indel), in different sorting directions, the structures of generated initial genomes would be various. In order to consider more combinations and globally improve the accuracy of results, we modify the initial method of GaDCJ median algorithm: we generate the initial population pool half by minimal DCJ method and the other half by minimal Indel method. Actually, we can tune the ratio of the two methods and to see which ratio is more suitable in which situation.

Another modified aspect is for unequal genomes sorting algorithm, we only sort the two unequal genomes (G_A and G_B) to their common segment markers (G_I) from different starting points. So along the sorting path (Figure 3.4 gives a demonstration), actually, we will have 12 different sorting steps with various genomes structures (meanwhile, for equal genomes, we may have some overlapped sorting points along the sorting path).

Based on the above reasons, in our DCJ-Indel median solver, we will enlarge the number of initial population pool in order to fit the complex insertion and deletion scenarios.

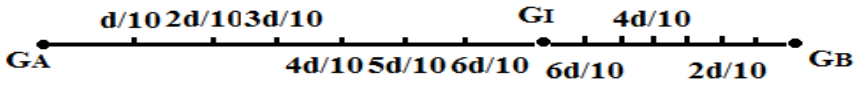

Figure 3.4 Sorting Two Unequal Genomes G_A and G_B to Their Common Segment Markers (G_I) From Different Starting Point

Crossover for GaDCJ-Indel algorithm

Based on our GaDCJ algorithm, the crossover method we adopt in DCJ-Indel model is also based on sorting genomes. Instead of using only DCJ sorting, we use both DCJ and Indel operations.

First, we pick two parents (P_1 and P_2) from the candidate pool and compare their fitness score F_1 and F_2, assuming P_2 has better fitness score than P_1. Then we will generate two children genomes C_1 and C_2 from the two parents. C_1 is generated by selecting a genome which is on the sorting path from P_1 to P_2 (from less fit one to the better one) and is m (randomly chosen) steps away from P_1. Here we need to

consider whether the value of m is larger then the total number of dcj operations or not. To realize sorting procedure, we always do dcj sorting first. So if m is larger than or equal to D_{dcj}, we will do the total number of DCJ sorting operations first and then do the rest number of indel operations; otherwise, m dcj sorting operations are only adopted. In other words, the new child obtains genetic material from both parents by applying DCJ and indel operations on one parent P_1, with respect to the one with better fitness P_2. We do not generate C_2 by sorting from the worse to the better. As from our experiments, this can easily destroy the good genetic structures and leads to bad solutions. Instead, we generate C_2 as the direct copy of P_2 (which has better fitness), given the better genome a higher chance to pass its good structures in next generations.

Mutation for GaDCJ-Indel algorithm

In the DCJ-Indel median solver, an individual can be mutated by applying a random number of DCJ and indel operations to sort to another individual. However, there are two questions to be answered: how many operations are required and which operations should we choose to apply?

Liking GaDCJ algorithm, we use sur-triangle inequality (explained above) to estimate the lower bound of median score. Although the actual distances may be different from those estimated values, it is a good indicator that how close a genome is to the true median. If one genome has its three edge lengths are too far away from their estimated lengths, this genome is likely to be bad and should be mutated toward a better one. For a genome G, we can compute its three edge lengths to the given genomes, and find the one which has the largest length difference between the generated and estimated one. We then sort G some random m steps closer to that given genome. We should also compare the value of m with D_{dcj} to decide the ratio of DCJ and Indel sorting operations.

Besides that, we may use adaptive mutation ratio with time passed by. At first several generations, the population has enough diversities, so it is not necessary to put high mutation rate(0.1). Later, with the number of generation increasing, there will be some domestic genomic structures, so we tune up the mutation rate to 0.2 in order to explore more search space and avoid stack in local optima.

We conduct the above procedure on the two children genomes (C_1 and C_2) obtained from the crossover procedure. As a result, we get two new genomes C_1' and C_2'. We then choose the two best from the four generated off-spring (C_1, C_2, C_1' and C_2') in order to maintain enough diversities and enhance the quality of individuals in the next generation.

Other Procedures

Based on the good performance of previous researches, we follow the same selection methods and other algorithm details in each procedure described in Chapter 2.

3.4 Experiments Results and Discussions

Datasets

Although existing DCJ median solvers can solve equal genomes median problem efficiently, rare of them can handle insertion and deletion operations, what's more no one can solve distant unequal genomes. In our experiments, the real gene order data is so hard to get and we don't know the true ancestors and topology evolutionary in history, so in this research field we always use simulation data. So we already know the correct solutions and can compare our results with them. So we compare our DCJ-Indel method results with the relative true median score and the true ancestor which are generated by a simulator.

We tested our DCJ-Indel methods on simulation datasets of three genomes with

200 gene orders for each one. We generated trees with three leaves and one internal node, assigned the identity permutation on the internal node and generated the three leaves by applying rearrangement events along each edge respectively. The number of events on each edge is controlled by a birth-death process which is viewed as a good model to fit evolutionary trees. We set the insertion and deletion event percentages: one is 2% and the other is 4%. The datasets are grouped by the average edge lengths (r), which are 50, to 200 events per edge in our experiments, with the $\frac{r}{N}$ rates of 0.5, 1.0, 1.5,2.0, ranging from easy to extremely difficult. For each r, we generated 10 datasets and averaged the results.

The maximum number of iterations for our GA method was set at 100 but will stop earlier if the perfect median score is met. The one genome with the lowest median score will be reported as the final result. In our experiments, this maximum number is big enough that all instances have their best genome appeared within fewer than 100 iterations.

Comparison with True Ancestor and True Median Score

Currently there is rarely available indel median problem solver, so we use simulation datasets and compare our GaDCJ-Indel algorithm's results (G_M) with true ancestor (G_T) and true median score summed by the DCJ distance between true ancestor to each of the three given leaves.

Table 3.1 shows the average results for indel median scores, measured by DCJ-Indel distance. The minimal different percentage is 0.62% and the largest one occurred in the hardest datasets is only 7.5%. We can see that our GaDCJ-Indel algorithm can achieve very close or even the same indel median scores in easy cases, meanwhile for large distance datasets, our algorithm can still give reasonable indel median scores.

For an unrooted tree defined by three given unequal genomes, the indel median

genome can be used to estimate the gene order of the internal node, which we use to infer the missing ancestor. Thus the distance to the true ancestor (known in simulations) is an additional measurement to verify the quality of indel median solvers. Table 3.2 gives the average DCJ-Indel distance to the known authentic ancestor for our methods. From the data, we can see that the median genomes inferred by our GaDCJ-Indel method are indeed very close to their true ancestor and the fractions between the differences with total relative evolutionary event numbers are very small. The minimal one is 0.9% in the easier case and the largest one is 5.15%. Even for $r = 200$, the DCJ-Indel distance between the inferred and true ancestor is still less than 62 (10%).

We also test more distant and difficult datasets which have 4% insertion and deletion events with diameter from $r = 50$ to $r = 200$. Table 3.3 and Table 3.4 show the average results and averaged different percentages of these experimental results to their true scores. It is surprised to see that they are almost as good as easy ones. It indicates that our GaDCJ-Indel algorithm has very large computational scale, which not only can handle small distance with little indel events datasets, also have good performance on large and difficult datasets.

This suggests that the sorting-based mutation and crossover procedures are very effective and preserve important genomic structures, so our indel median solver can provide very close ancestor genomes from easy case to very harder one in both distance and structural aspects.

Table 3.1 Comparison to the true median score with indel rate of 0.02

	r=50	r=100	r=150	r=200
Our Average Score	121.4	221.2	280.2	303.2
True Average Score	119.4	222.6	295.8	327.8
Diff Percentage	1.67%	0.62%	5.27%	7.5%

Table 3.2 Comparison to the true ancestors with indel rate of 0.02

	r=50	r=100	r=150	r=200
$d_{G_T} - d_{G_M}$	2.7	27.7	44.0	61.8
$\frac{d_{G_T} - d_{G_M}}{3*r}$	0.9%	4.61%	4.89%	5.15%

Table 3.3 Comparison to the true median score with indel rate of 0.04

	r=50	r=100	r=150	r=200
Our Average Score	121.0	220.2	260.8	303.4
True Average Score	115.6	215.0	273.7	336.9
Diff Percentage	5.0%	2.42%	4.71%	9.94%

Table 3.4 Comparison to the true ancestors with indel rate of 0.04

	r=50	r=100	r=150	r=200
$d_{G_T} - d_{G_M}$	2.1	17.8	41.5	67.7
$\frac{d_{G_T} - d_{G_M}}{3*r}$	0.7%	2.97%	4.61%	5.64%

Convergence

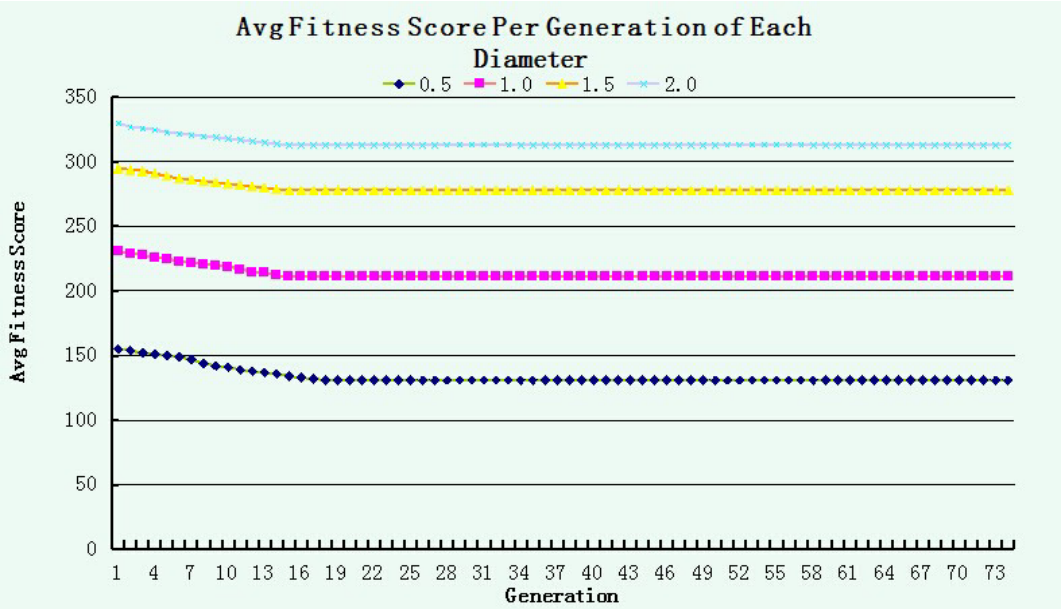

Figure 3.5 Average Fitness Score with Generation Number Increasing

Let's look at the above 3.5, we can see that our algorithm has little premature issues and can converge to an optimal solution generation by generation. In future, we

may test different terminate criteria in order to limit computational time consuming and improve our algorithm efficiency.

Time and Space Performance

Based on experiments, the consuming time for our GaDcj-Indel algorithm is no exceed 20 minutes for each $r = 200$ datasets and for small event rate, our algorithm can finish all data in less than several minutes. Lots of current equal genomes median solvers use huge of memory spaces in order to store temporary information. Although they may use less time for solving small datasets, they cannot finish anyone in distant genomes. If following their methods in solving indel median problem, we will also meet the short of storage problem. Our algorithm only needs little memory to record each population, so it can easily handle large distant datasets.

3.5 Conclusion

We propose the first genetic algorithm based median solver with unequal content genomes, but without duplications, taking into consideration of DCJ and indel operations. By DCJ-Indel sorting operations, following the four steps of classic genetic algorithm, we develop our GaDCJ-Indel algorithm. We use various dcj and indel operation ratios to generate initial population pool with enough diversities. So we can reach sufficient search space. Then we adopt dcj operation and indel operation in crossover and mutation procedures. For testing our algorithm, we use two group of datasets with different indel event rate. All experimental results indicate that our GaDCJ-Indel algorithm has very large computational scale, which not only can handle small distance with litter indel events datasets, also have good performance on large and difficult datasets. Besides that, our algorithm only takes relative limited memory space no matter what kind of data being used.

Chapter 4

Reconstructing Ancestral Genomic Orders Using Genetic Algorithm Model

4.1 Introduction

Phylogenetic analysis focuses on the study of evolutionary relationships among groups of organisms (e.g. species, populations), based upon similarities and differences in their physical and/or genetic characteristics [20, 19]. A phylogenetic tree or evolutionary tree is often described as a binary tree: its leaves are the given set of descendant organisms and internal nodes stand for extinct ancestors connected by edges to indicate evolutionary relationships [75].

To date, phylogenetic reconstruction generally deals with two types of data: DNA or protein sequences or gene orders. Although sequence data still dominate, gene order was acknowledged early on as a valuable phylogenetic character [76, 77, 78, 79]. There are several tools for gene order analysis, most of those require the scoring of trees and by comparing the scores, pick the best-scored one as the phylogeny. In the scoring procedure, one by-product is the inference of ancestral genomes by labeling internal nodes with gene orders. Over the past few years, ancestral gene-order inference has brought profound predictions of protein functional shift and positive selection [80].

Methods for scoring and inferring ancestral genomes assume a given tree topology and assignment of genomes on leaf nodes, known as the small phylogeny problem (`SPP`). There are currently two types of methods, maximum parsimony methods and

maximum likelihood methods, which have various of limitations. For example, parsimony methods rely on iteratively solving the median problems, which are NP hard and very difficult to compute. Current methods are also easily stuck into local optima.

To avoid the problems of existing methods, in this paper, we present a cooperative co-evolutionary genetic algorithm that provides a method that globally scores trees and infers ancestors. Co-evolution is defined as the process of reciprocal evolutionary change that occurs between species when they interact with each other. There are two types: competition or cooperation [81, 82]. Cooperative co-evolutionary genetic algorithm (`CCGA`) divides a problem into smaller sub-problems which are connected by some links. Our method takes each internal node as one species, which has certain degree of interactions with others depending on the connections between them. Our `CCGA` algorithm uses fitness score designed to consider co-evolution and initializes and evolves each population based on genomic sorting. Our extensive experiments on simulated datasets show that compared with other methods, it not only can find relative accurate tree scores, but also can infer ancestors that are much closer to true ancestors.

4.2 Basic Notations and Preliminaries

Ancestral Genome Inference

Maximum parsimony methods typically iterate over each internal node to solve for the median genomes until the sum of rearrangement events over all edges is minimized, which is reported as the tree score. The median problem can be formalized as follows: given a set of 3 genomes with permutations $\{G_i\}_{1\leq i\leq m}$ and a distance measurement d, find another permutation G_m such that the median score defined as $\sum_{i=1}^{3} d(G_i, G_t)$ is minimized, which is NP-hard for most distance measurements [83].

One of the best parsimony method is `GASTS` [84]. As shown in Figure 4.1, each

internal node initialization can define as a median problem. `GASTS` uses generalized weighted adequate sub-graphs to initial each internal node, however the method do not have an optimality guarantee: they form the basis for heuristic assignment of the median in the initialization phase [84]. Although it speeds up the difficult median computation, the above procedure captures optimal substructures and can be easily trapped into local optima.

The most recent maximum likelihood method is `PMAG` [85], which encodes gene orders into binary sequences and finds ancestral genomes based on a probabilistic framework. As it avoids the expensive bottleneck of median computation, it is fast and has better performance when genomes are distant. When PMAG computes the ancestor on an internal node, it first re-roots the tree to make it the root, then finds the probability of each adjacency. As a result, each internal node is considered independently and does not consider results from other internal nodes, thus may also be trapped in local optima.

Since existing methods have various problems on scoring and inferring ancestors, a global method is always desirable, which is the main motivation of our genetic algorithm method.

Genetic Algorithm in Phylogenetic Inference

Genetic Algorithms (GA) [86, 87] are inspired by evolution and governed by Darwins theory of natural selection: the fittest one will have the higher chance to survive. With the evolution progresses, "good" gene segments will propagate throughout the population and two good parents will have higher chance to produce better offspring than bad parents. As a result, each successive generation will become better adapted to their living environment. A genetic algorithm will iterate until a solution is found or the maximum number of iterations is reached.

Genetic algorithms are widely used in solving hard optimization problems, includ-

ing those in computational biology [87, 88]. Although genome rearrangement deals with chromosomes, evolutions and mutations, setting up a proper GA method is not trivial and the first GA-based median solver was not available until last year [89]. This GA-based median solver relies on sorting, i.e. when two individuals (genomes) are picked to produce offspring, it will sort the bad genome some steps toward the good one, thus creating offspring with genomic structures of the good genome. Simulation results showed that this median solver is very accurate and finds median genomes that are closer to true ancestors, compared with the best exact DCJ solvers.

Extending the median solver to globally score a tree and infer ancestral genomes is not trivial as a tree essentially defines an ecosystem with populations on internal nodes that interact with each other. To cope with this restriction, our new method is a cooperative co-evolutionary genetic algorithm. Given a tree with fixed topology and its leave genomes, we can treat each internal node as a species (with a population of individual genomes). As there are multiple internal nodes, changing genomes on one node will impact its neighboring nodes (Figure 4.1). Our method should proceed with the goal to make the whole ecological system (i.e. all internal nodes of the given tree) best fit. To achieve this, we use the following procedures to initialize the populations and select best fit individuals to reproduce, which is based on a carefully designed fitness function and sorting-based crossover and mutation procedures.

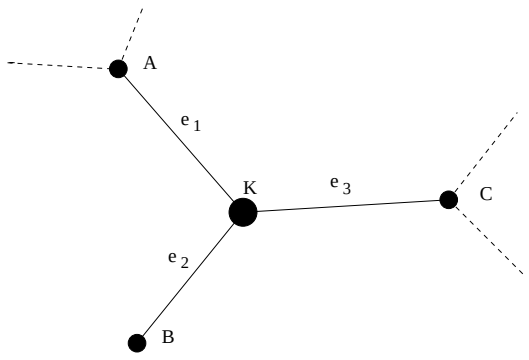

Figure 4.1 An example of tree topology around internal node K, nodes A and C are also internal, while B is a leaf which has its genome defined by the input.

4.3 Motivation

Co-evolution is defined as the process of reciprocal evolutionary change that occurs between species when they interact with each other. Mainly, there are two types of it: competition or cooperation [81, 82]. Cooperative co-evolutionary genetic algorithm (`CCGA`) divides a problem into smaller sub-problems which are connected by some links. After doing some exploration in GA-based phylogenetic solver, we propose a `CCGA-Tree` algorithm. We can take each internal node as one species, which has a certain degree of interaction with others depending on the genetic distances between them.

Although current phylogenetic reference solvers, like `GRAPPA` and `GASTS` using branch-and-bound as well as heuristic search methods to get fast and accurate ancestor genomes, have good performances on small datasets. However it restricts the solution search spaces and diversities, especially when giving long distant leave genomes data sets. Besides that, they all use distance matrixes to record medial step results and iteratively update those medial results. With increasing the number of leaves and their gene order numbers, the temporal storage spaces it needs grow quickly and exponentially. It limits current solvers' computational capacity.

Wanting to improve the above limitations, we propose our `CCGA` algorithm. Our DCJ-Median algorithm (using DCJ distance method has several advantages:

1. can explore wider solution spaces and provide more accurate ancestor genomes;
2. just needs relatively small and stable storage spaces for initial population pool instead of huge temporal space for medial matrixes;
3. uses the idea of cooperative co-evolutionary and meta-population method in order to jump out local optimal issue and extend solutions searching space for ancestor inference.

Distance Definitions

Given a tree, if we know its topology and internal genomes, we can estimate the length of an edge by computing the distance using the corresponding two end genomes. However, since we do not know internal genomes in real data analysis, we use the linear programming method developed by Tang and Moret, which is fast to compute and provides a good approximation of each edge length of any given tree [90]. For an edge with two end nodes A and B, we call its edge length obtained by this linear programming method the expected edge length $d_{exp}(A, B)$.

If the labeling (assignment) of genomes on two nodes A and B are known, e.g. genome G_A is assigned to node A and G_B is assigned to B, we can define the distance between these two genomes $d(G_A, G_B)$ to be their DCJ distance.

Since in our algorithm, each internal node is a species which has its own population, thus we must consider cooperative co-evolution between species and reflect this in our distance definition. Assume the ith individual of an internal node K has gene order G_K^i, and node K is connected to another internal node A (Fig. 4.1), the distance between genome G_K^i and node A must be defined as a distance between an individual and a population. As we can compute the DCJ distance between G_K^i and every individual in A, the simplest is of course to use the smallest DCJ distance or to use the average of all these distances. The former gives too many influences of one individual, while the later takes into account some bad individuals, thus both can not guarantee to keep the best genomic structures. In our algorithm, we use the smallest 50% distance between G_K^i and individuals in A's population, and average their distances as $d_{avg}(A, G_K^i)$. If a node is leaf, as its genome is known and unique, d_{avg} is simply the DCJ distance between the individual and the assigned leaf genome. For example, in Fig. 4.1, node B is a leaf, thus $d_{avg}(B, G_K^i) = d(G_B, G_K^i)$.

Initializing Populations on Internal Nodes

Since genetic algorithms are based on the evolution of a population, the distribution and choices of initial population have huge influences on the search space impacts its efficiency and effectiveness. Indeed, from our experience, how to identify a proper initial population is the most important problem here. After several experiments and comparisons, we decide to use the following approach (Fig. 4.1):

1. As there are several methods (such as GASTS [84] and PMAG [85]) that can provide a solution to ancestral genomes, we will borrow their solutions as seeds to populate every internal node;

2. For each internal node K and its seed genome, we then start to generate its population. There are three neighboring nodes (A, B, C) known from the given tree topology, each with its own assigned (seed or leaf) genome. For example in Fig. 4.1, G_A, G_C use seed genomes while G_B is a leaf genome determined by the input. If A is the parent node, evolving from species A to B will pass through K, thus we can assume K is on the sorting path from G_A to G_B. Based on this observation, we can generate some candidate genomes for internal node K by sorting along the path from G_A to G_B with $\frac{1}{10}d(G_A, G_B), \frac{2}{10}d(G_A, G_B), \cdots, \frac{6}{10}d(G_A, G_B)$ steps away from the starting point G_A. By switching the starting and end points, each pair of the three neighboring nodes can generate 12 different genomes, giving 36 different initial genomes for node K.

3. To get more representative population, we need to repeat the above procedure N times and get an initial population pool for node K with $36 \times N$ various genomes.

The repetition number of N is obviously influenced by the number of genes and the distances among genomes. In our prior GA-based median solver, we set N to be

a constant (50), which is too large for closed genomes, but too small when genomes are distant. Given a tree, we can define its diameter as the longest path between two leaves. As larger diameter generally indicates that genomes are more distant, we need larger populations to cover the huge search space. In this paper, we use the following setting: $N = \frac{diameter}{10}$. Since d is unknown, we can estimate it based on the seed and leaf genomes, as its exact value is not that critical.

Computing Individual's Fitness Function

In GA algorithms, the expression of fitness function depends on the optimization goals. To score and infer ancestors, we want to find ancestral genomes that are close to the true ancestors; as these true ancestors are unknown in real analysis, we want to find those that minimize the evolutionary events. The later requirement can be considered to find genomes that minimize the summation of all edge lengths (i.e. tree score), while the former is a bit difficult to judge. Given the true tree T with true ancestors, and an inferred tree T' with the same topology but different genomes, we can compare the two trees by examining edge lengths: if T' has the same genomes as T, these two trees should have the same edge lengths. In other words, if T' has edges that with skewed lengths, the inferred ancestors may be far away from true ancestors. With this observation, for each internal node K, we can define the fitness score for its ith individual (G_K^i) as the following (Fig. 4.1):

$$F = \sum_{j=1}^{3} abs(e'_j - e_j) + \sum_{j=1}^{3} e'_j \tag{4.1}$$

where e'_j is the average length of the jth edge ($j = 1, 2, 3$) between G_K^i and its neighboring nodes, e_j is the expected length of the jth edge in the true tree. In Fig. 4.1, $e'_1 = d_{avg}(A, G_K^i), e'_2 = d_{avg}(B, G_K^i)$ and $e'_3 = d_{avg}(C, G_K^i)$, while $e_1 = d_{exp}(A, K)$, $e_2 = d_{exp}(B, K)$ and $e_3 = d_{exp}(C, K)$.

The first term is actually very important as we not only want to find trees with

minimum scores, we also want to maintain the tree shape and avoid edges being skewed.

Selection

The above fitness function indicates that for a given internal node (and its species), the one with smaller score makes the whole ecology system better fit. Thus it should be given a higher chance to pass its good genomic structure into the next generation. Following our experimental results and previous researches, we adopt a widely used hybrid approach of some traditional selection methods. For each internal node and its species, the population size will be kept the same from generation to generation. At each generation, we keep its top 10% individuals and copy them (without change) into the next generation in order to pass their good structures. We then randomly select two individuals from the remaining 90% population to produce two offspring (using methods discussed next); this procedure is repeated until the population limit is reached.

Crossover

After two individuals are selected, we will use crossover to exchange their genetic materials and produce offspring. As we are dealing with gene orders, we can not just adopt the simple two points exchange as done by most genetic algorithms. Similar to our GA-based median solver, the crossover procedure is based on genomic sorting. For two parents (P_1 and P_2), we compute their fitness scores which reflect their relationship with other species. Assume P_1 has a lower (better) fitness score, we produce the first child C_1 by copying P_1, given the better genome a higher chance to survive. We then produce the other child C_2 by selecting a genome on the DCJ sorting path from P_2 to P_1 (from the worse score one to the better one) with m (randomly chosen) steps away from P_2. In other words, the new child C_2 has both

parents' genetic material by applying DCJ operations, with the better fit one to have higher chance to decide their child's evolutionary directions.

Mutation

Mutation happens randomly to provide diversity, which is useful to avoid local optimal by preventing the population from converging rapidly and evolving to similar genomic structures. Here we still apply DCJ sorting operations to realize the mutation procedure.

With the probability threshold of 0.2, we randomly select one individual G_K^i in internal node K's current population pool to mutate it by applying some DCJ sorting operations. If G_K^i is far from the true, its average edge lengths may also be very different from the expected edge lengths, thus we want to correct the most skewed edge by sorting G_K^i towards the neighboring node corresponding to that skewed edge, using the following steps (Fig. 4.1):

1. By using the linear programming method, we know the expected three edge lengths $d_{exp}(A, K), d_{exp}(B, K), d_{exp}(C, K)$ of the true internal node K to its three neighbors A, B and C.

2. As we also need to consider cooperation evolution conditions, we will get the average distances between G_K^i and its three neighbors (i.e. $d_{avg}(A, G_K^i)$, $d_{avg}(B, G_K^i)$, and $d_{avg}(C, G_K^i)$), using only the closet 50% individuals from each of the three neighbors as we mentioned before.

3. We then identify which neighbor has the largest absolute difference between the expected and average edge lengths, and sort G_K^i along that edge to get a new genome, using number of steps determined randomly.

The crossover and mutation algorithms described above not only consider the evolutionary pressure between related species, but also ensure that individuals with better

fit in the ecology system have a better chance to preserve its genomic structure by DCJ sorting.

Iterating through Generations

In our CCGA algorithm, as we are dealing with meta-populations, we need to evolve each internal population, and propagate this evolution through its neighboring nodes. For each generation of the ecosystem, starting from internal nodes closest to leaves and moving inward, we allow the population of each internal node to evolve I_1 generations, using its last generation to cooperate with other nodes. Once we finish a generation of the whole ecosystem, we restart the above procedure until I_2 generations are reached. In other words, there are two iterations: I_1 lets each species to evolve separately, while I_2 keeps the "whole ecological system" to evolve and update.

Based on our experimental results, also considering running speed and efficiency, the maximal numbers of both I_1 and I_2 only need to be set at 20 to reach convergence and obtain satisfactory results (see the next section). Once the program stops, it will scan each internal node and report the one with the best fitness score (using our predefined fitness function) as the ancestral genome for that internal node; it then computes each edge's DCJ distance, using genomes assigned to its two end nodes; the final tree score is then simply the summation of all edges' DCJ distances.

4.2 gives us a clear interpretation for the whole algorithm procedures.

4.5 Experimental Results and Discussions

We used C++ to implement our `CCGA` method and conducted various experiments on simulated datasets to assess its accuracy. Simulation is the main approach to evaluate the quality of a phylogeny method, as its evolutionary history is known. We used `GASTS` and `PMAG` to provide the initial population for `CCGA`, and compared it with these two methods directly. As `GASTS` requires very large amount of storage space when

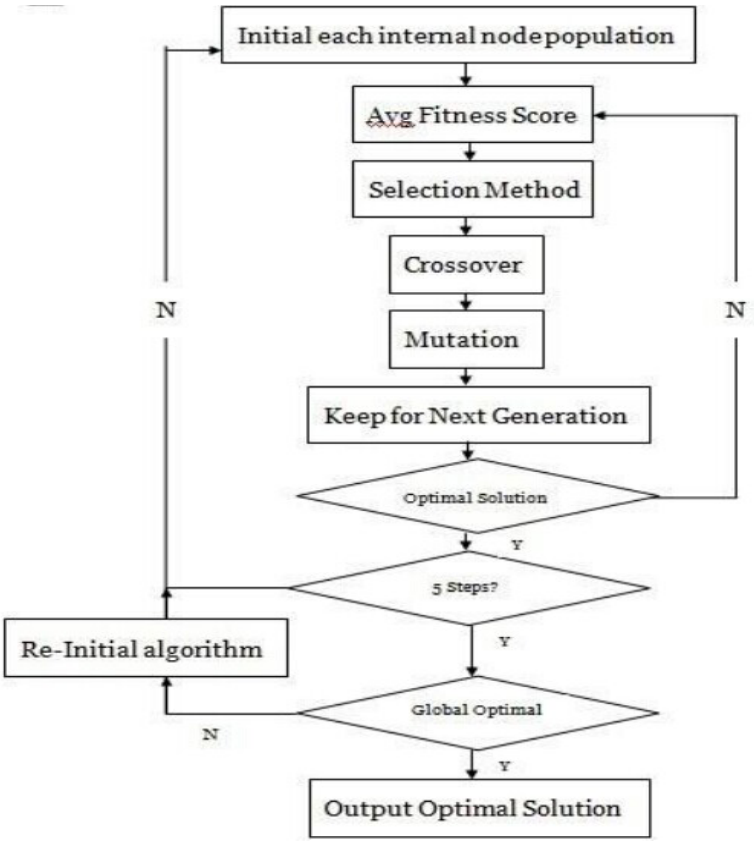

Figure 4.2 A flow chart of CCGA algorithm

the genomes are distant, we used a shared-memory computer with 256GB memory and 1T hard disk space to run the experiments.

We utilized the simulator proposed by Lin et al. [91] to produce birth-death tree topologies. With a model tree, we can produce genomes of any size and difficulty by simply adjusting three main parameters: the number of genomes N, the number of genes n, and the tree diameter. In this paper, we used $N = 10, 20$ and $n = 100, 200, 1000, 2000$. We also used 4 different evolutionary diameters which are $0.5n$, $1.0n$, $1.5n$ and $2.0n$, representing data from easy to extremely difficult. For each combination of parameters, we generated 10 datasets and gave the averaged final results.

From table 4.1, we can see that for each datasets (with small diameters), GASTS gives tree scores that are very close or even exactly the same as true tree scores,

Table 4.1 Differences of Tree Scores Compared with True Trees

	n=100			
diameter	0.5n	1n	1.5n	2n
CCGA_PMAG	1.2	12.2	18.4	50.7
PMAG	10.4	19.8	24.6	54.8
CCGA_GASTS	1.0	10.8	16.4	48.2
GASTS	0.4	10	16.2	50.4
	n=1000			
diameter	0.5n	1n	1.5n	2n
CCGA_PMAG	32.2	57.3	79.6	177.5
PMAG	33.6	76.2	124.2	180.2
CCGA_GASTS	29.8	37.0	59.2	179.2
GASTS	29.0	32.0	59.0	259.3

(a) Results from datasets with $N = 10$ leaves. CCGA_PMAG and CCGA_GASTS use PMAG and GASTS to initialize internal nodes respectively.

	n=100			
diameter	0.5n	1n	1.5n	2n
CCGA_PMAG	2.2	3.2	16.6	42.5
PMAG	20.4	22.8	37.8	43.0
CCGA_GASTS	0.7	2.8	4.8	43.0
GASTS	0.4	2.8	4.8	49.3
	n=1000			
diameter	0.5n	1n	1.5n	2n
CCGA_PMAG	6.2	14.6	108.1	122.8
PMAG	34.2	50.4	126.6	122.8
CCGA_GASTS	3.0	7.0	100.2	124.2
GASTS	3.0	6.8	110.3	125.5

(b) Results from datasets with $N = 20$ leaves. CCGA_PMAG and CCGA_GASTS use PMAG and GASTS to initialize internal nodes respectively.

while our CCGA method using either GASTS or PMAG as initialization method also can provide very close tree scores. For datasets with larger parameters, our CCGA method almost always improves upon the initialization results, achieving better tree scores than the other two methods.

For each internal node, the distance between the inferred and ancestral genomes is an additional measurement for the quality of our method. Table 4.2 shows the average DCJ distances between the inferred and ancestral genomes, summed over

Table 4.2 DCJ Differences from Inferred Genomes to True Ancestors

	n=100			
diameter	0.5n	1n	1.5n	2n
CCGA_PMAG	0.6	10.2	16.8	42.8
PMAG	3.4	13.2	17.2	48.4
CCGA_GASTS	0.4	9.6	18.2	41.6
GASTS	0.4	9.8	21.4	47.6
	n=1000			
diameter	0.5n	1n	1.5n	2n
CCGA_PMAG	20.2	36.8	66.2	144.2
PMAG	20.0	69.2	183	166.6
CCGA_GASTS	20.0	32.6	49.6	168.2
GASTS	22.4	28.4	51.8	378.6

(a) Results from datasets with $N = 10$ leaves.

	n=100			
diameter	0.5n	1n	1.5n	2n
CCGA_PMAG	2.4	3.4	9.3	29.7
PMAG	2.8	6.4	18.2	32.0
CCGA_GASTS	0.4	3.2	7.6	33.2
GASTS	0.4	3.0	7.6	39.5
	n=1000			
diameter	0.5n	1n	1.5n	2n
CCGA_PMAG	4.8	6.5	113	128.2
PMAG	15.2	46.6	111.6	137.6
CCGA_GASTS	2.4	2.4	120.8	137.8
GASTS	2.2	2.4	128.6	139.6

(b) Results from datasets with $N = 20$ leaves.

every internal node. It is very surprised to see that for almost all datasets, no matter if they are difficult or easy, internal nodes inferred by our CCGA method are indeed much closer or even equal to the true ancestors compared to those inferred by GASTS and PMAG. This suggests that the sorting-based mutation and crossover procedures are very effective and preserve important genomic structures.

From Tables 4.1 and 4.2, we can also see that although we used different initialization methods, their final average results are very close. It indicates that when by using suitable initial seed methods, no matter which one is used, our GA-based phy-

logenetic inference algorithm will provide good results. The slight differences also tell us that GA based algorithm is indeed a population-based algorithm, which performance is impacted by its number of population and how to initial its population. In the future, we will explore more initial seed methods to further improve our method.

GASTS is an efficient heuristic phylogenetic tool which itself relies on initially labeled internal nodes. In GASTS, for an internal node to be initialized, two of its three neighbors must be already initialized or known (such as leaves). The third node, while typically not be initialized yet, GASTS wants to gather as much information as possible, so it summarizes the data available in the third subtree into a set of weighted adjacencies. Thus the information it uses to initialize a node consists of two 0-1 sets of adjacencies from the two initialized neighbors and one weighted set of adjacencies from the third neighbor. The weight w_x for each adjacency x is given by

4.6 Conclusions

We propose a new method to score and infer ancestor genome structure with a fixed tree topology. Median-based approaches are widely used to solve phylogenetic reconstruction and ancestral inference problems and provide good performance for small and closely related genomes. However, using these methods is computationally very expensive and can easily get stuck in local optima, thus it is not suitable for real genomes which are larger and more complex. Our genetic algorithm is a cooperative co-evolutionary method based on meta-population. From our experimental results, we find that by using good initialization methods and sorting based crossover and mutation procedures, as well as careful consideration of co-evolution, our GA-based method not only converges to relatively small tree scores, but also provides good estimation of ancestral genomes. It also confirms the importance of sorting in solving the ancestral inference problem, and our approaches can be adopted to include other events such as deletions, insertions and duplications, for which new efficient sorting

algorithms are available, enabling us to avoid the difficult quest of finding median solvers. However, this paper is our first attempt to use the approach of cooperative co-evolutionary algorithm in gene order analysis, thus lots can be improved to make it more efficient and accurate by finding better initialization, selection and reproduction procedures.

Chapter 5

Conclusions

Our work focuses on using genetic algorithm (GA) framework to solve median problem and infer phylogenetic ancestor genomes and develop the first genetic algorithm median solver and phylogenetic inference tool.

For median problem, We propose a genetic algorithm median solver using DCJ distance and sorting methods. It is the first attempt to use the approach of genetic algorithm in gene order analysis. In the frame of classic genetic algorithm, in order to fulfill different demands, we develop our own algorithms for each procedure adopting DCJ sorting operations in order to generate initial population and pass "good" genetic materials by crossover and mutation procedures. Our algorithm has the ability to extend optimal median solution space in limited space and time, so it is not easy to stack at local optimal, especially in large scale and distant datasets. Our experiments on simulated datasets shows that our GA method is very efficient and has better speed and accuracy compared to existing methods such as GRAPPA and AsMedian. The excellent performance of our algorithm indicate that sorting operation is very useful in solving DCJ median problem. Besides that, our approaches can be adopted to include other events such as deletions and insertions, for which linear algorithms are available to compute the distance.

For more complex situation, we develop the first unequal genomes median solver using genetic algorithm, but without duplications, taking into consideration of DCJ and indel operations. In the frame of classic genetic algorithm, we use various DCJ and indel operation ratios to generate initial population pool with enough diversities.

So we can reach enough search space and avoid stack into local optima. We also adopt DCJ operation and indel operation into crossover and mutation procedures. Two groups of datasets with different indel event rate are used to verify the performance of our GaDCJ-Indel algorithm. All experimental results indicate that our GaDCJ-Indel algorithm not only can handle small distance datasets with little indel events, but also has good performance on large and difficult datasets. Besides that, our algorithm only take relative limited memory space no matter what kind of data being used and has very large computational scale.

For phylogeny inference aspect, we first attempt to use the approach of cooperative co-evolutionary algorithm in gene order analysis. We proposed a GA-Tree algorithm which adapts meta-population, co-evolution and repopulation pool methods with a fixed tree topology. Currently median-based approaches are widely used to solve phylogenetic reconstruction and ancestral inference problems for small and closely related genomes. However, these methods are very time consuming and can easily get stuck into local optima, thus it is not suitable for real genomes which are usually larger and more complex then we generally assumed. Our genetic algorithm, based on previous researches which are using sorting based crossover and mutation procedures, explored a new way to solve this problem. From our experimental results, by using good and reasonable initialization methods, as well as careful consideration of co-evolution, our GA-based method not only converges to relatively small tree scores, but also provides good estimation of ancestral genomes. It also confirms one of our previous conclusions: the important role of sorting, and our approaches can be adopted into other events such as deletions, insertions and duplications, enabling us to avoid the difficult question of finding median solvers. This is a exploration in phylogenetic area, thus in future lots can be improved to make it more efficient and accurate by finding better initialization, selection and reproduction procedures.

Bibliography

[1] Guillaume Bourque and Pavel A Pevzner. Genome-scale evolution: reconstructing gene orders in the ancestral species. *Genome research*, 12(1):26–36, 2002.

[2] B. M.E. Moret, D.A. Bader, T. Warnow, S.K. Wyman, and M. Yan. GRAPPA: a high-performance computational tool for phylogeny reconstruction from gene-order data. In *Proc. Botany*, Albuquerque, NM, August 2001.

[3] Maryam Haghighi and David Sankoff. Medians seek the corners, and other conjectures. *BMC bioinformatics*, 13(Suppl 19):S5, 2012.

[4] Zorica Stanimirović. A genetic algorithm approach for the capacitated single allocation p-hub median problem. *Computing and Informatics*, 29(1):117–132, 2012.

[5] D.S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18(6):341–343, 1975.

[6] J. Hein. A new method that simultaneously aligns and reconstructs ancestral sequences for any number of homologous sequences, when the phylogeny is given. *Molecular Biological and Evolution*, 6:649–668, 1989.

[7] *Drosophila* genome project, baylor college of medicine. 2003.

[8] AH Sturtevant and Th Dobzhansky. Inversions in the third chromosome of wild races of drosophila pseudoobscura, and their use in the study of the history of the species. *Proceedings of the National Academy of Sciences of the United States of America*, 22(7):448, 1936.

[9] Th Dobzhansky and AH Sturtevant. Inversions in the chromosomes of drosophila pseudoobscura. *Genetics*, 23(1):28, 1938.

[10] Mathieu Blanchette, Guillaume Bourque, and David Sankoff. Breakpoint phylogenies. *Genome Informatics*, 1997:25–34, 1997.

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

[12] S. Hannenhalli. Polynomial algorithm for computing translocation distance between genomes. *Discrete Appl. Math.*, 71:137–151, 1996.

[13] S. Hannenhalli. Polynomial-time algorithm for computing translocation distance between genomes. In Z. Galil and E. Ukkonen, editors, *6th Ann. Symp. Combinatorial Pattern Matching*, volume 937 of *Lecture Notes in Computer Science*, pages 162–176, Espoo/Helsinki, Finland, 1995. Springer-Verlag.

[14] D. Sankoff and P. Trinh. Chromosomal breakpoint reuse in genome sequence rearrangement. *Journal of Computational Biology*, 12(6):812–821, 2005.

[15] S. Hannenhalli. *Software for computing inversion distances between signed gene orders*. Department of Mathematics, University of Southern California, URL. http://www-hto.usc.edu/plain/people/Hannenhalli.html.

[16] S. Hannenhalli and P.A. Pevzner. Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals). In *Proc. 27th Ann. Symp. Theory of Computing (STOC95)*, pages 178–189, Las Vegas, NV, 1995. ACM.

[17] I. Rogozin, K. Makarova, Y. Wolf, J. Murvai, E. Czabarka, L. Szekely, R. Tatusov, and E. Koonin. Connected gene neighborhoods in prokaryotic genomes. *Nucleic Acids Res.*, 30:2212–2223, 2002.

[18] J. Venter, K. Remington, J. Heidelberg, A. Halpern, D. Rusch, J. Eisen, D. Wu, I. Paulsen, K. Nelson, W. Nelson, D. Fouts, S., A. Knap, M. Lomas, K. Nealson, O. White, J. Peterson, J. Hoffman, R. Parsons, H. Baden-Tillson, C. Pfannkoch, Y. Rogers, and H. Smith. Environmental genome shotgun sequencing of the sargasso sea. *Science*, 304:66–74, 2004.

[19] Walter M Fitch, Emanuel Margoliash, et al. Construction of phylogenetic trees. *Science*, 155(760):279–284, 1967.

[20] DF Robinson and Leslie R Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53(1):131–147, 1981.

[21] H. Levene. Genetic equilibrium when more than one ecological niche is available. *American Naturalist*, 87:311–313, 1953.

[22] Wayne P Maddison. Gene trees in species trees. *Systematic biology*, 46(3):523–536, 1997.

[23] Roderic DM Page and Michael A Charleston. From gene to organismal phylogeny: reconciled trees and the gene tree/species tree problem. *Molecular phylogenetics and evolution*, 7(2):231–240, 1997.

[24] U. Roshan, B. Moret, T. Warnow, and T.L. Williams. Performance of supertree methods on various dataset decompositions. In O.R.P. Bininda-Edmonds, editor, *Phylogenetic Supertrees: Combining information to reveal the Tree of Life*, pages 301–328. Kluwer Academic Press, 2004.

[25] R.C. Lewontin. Directions in evolutionary biology. *Annual Review of Genetics*, 36:1–18, 2002.

[26] W.M. Fitch. A non-sequential method for constructing trees and hierarchical classifications. *J. Mol. Evol.*, 18:30–37, 1981.

[27] D. Sankoff and P. Trinh. Chromosomal breakpoint reuse in genome sequence rearrangemen. *Journal of Computational Biology*, 12:812–821, 2005.

[28] William J Bruno, Nicholas D Socci, and Aaron L Halpern. Weighted neighbor joining: a likelihood-based approach to distance-based phylogeny reconstruction. *Molecular Biology and Evolution*, 17(1):189–197, 2000.

[29] Eric Tannier, Chunfang Zheng, and David Sankoff. Multichromosomal genome median and halving problems. In *Algorithms in Bioinformatics*, pages 1–13. Springer, 2008.

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

[31] Alberto 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.

[32] Mike Galles and Eric Williams. Performance optimizations, implementation, and verification of the sgi challenge multiprocessor. In *System Sciences, 1994. Proceedings of the Twenty-Seventh Hawaii International Conference on*, volume 1, pages 134–143. IEEE, 1994.

[33] A. Caprara. On the practical solution of the reversal median problem. In O. Gascuel and B.M.E. Moret, editors, *Proc. 1st Workshop Algs. in Bioinformatics (WABI'01)*, volume 2149 of *Lecture Notes in Computer Science*, pages 238–251, Århus, Denmark, 2001. Springer-Verlag.

[34] Adam C Siepel and Bernard ME Moret. Finding an optimal inversion median: experimental results. In *Algorithms in Bioinformatics*, pages 189–203. Springer, 2001.

[35] Andrew Wei Xu and David Sankoff. Decompositions of multiple breakpoint graphs and rapid exact solutions to the median problem. In *Algorithms in Bioinformatics*, pages 25–37. Springer, 2008.

[36] Randy L Haupt and Sue Ellen Haupt. *Practical genetic algorithms*. John Wiley & Sons, 2004.

[37] Lawrence Davis et al. *Handbook of genetic algorithms*, volume 115. Van Nostrand Reinhold New York, 1991.

[38] Siddhartha Bhattacharyya and Gary J Koehler. An analysis of non-binary genetic algorithms with cardinality 2 v). *Complex Systems*, 8(4):227–256, 1994.

[39] Y Leung, ZP Chen, ZB Xu, and KS Leung. Convergence rate for non-binary genetic algorithms with different crossover operators. *The Chinese University of Hong Kong*, 1998.

[40] Paul O Lewis. A genetic algorithm for maximum-likelihood phylogeny inference using nucleotide sequence data. *Molecular Biology and Evolution*, 15(3):277–283, 1998.

[41] Stephane Guindon, Franck Lethiec, Patrice Duroux, and Olivier Gascuel. Phyml onlinea web server for fast maximum likelihood-based phylogenetic inference. *Nucleic acids research*, 33(suppl 2):W557–W559, 2005.

[42] Alexandros Stamatakis. Raxml-vi-hpc: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics*, 22(21):2688–2690, 2006.

[43] DJ Zwickl. Garli: genetic algorithm for rapid likelihood inference. *See http://www. bio. utexas. edu/faculty/antisense/garli/Garli. html*, 2006.

[44] Matthew J Brauer, Mark T Holder, Laurie A Dries, Derrick J Zwickl, Paul O Lewis, and David M Hillis. Genetic algorithms and parallel processing in maximum-likelihood phylogeny inference. *Molecular Biology and Evolution*, 19(10):1717–1726, 2002.

[45] Robin R Gutell and Robert K Jansen. Genetic algorithm approaches for the phylogenetic analysis of large biological sequence datasets under the maximum likelihood criterion. 2006.

[46] L. Raubeson and R. Jansen. Chloroplast DNA evidence on the ancient evolutionary split in vascular land plants. *Science*, 255:1697–1699, 1992.

[47] P. Pevzner and G. Tesler. Human and mouse genomic sequences reveal extensive breakpoint reuse in mammalian evolution. *Proceedings of the National Academy of Sciences USA*, 100:7672–7677, 2003.

[48] S. Richards. Comparative genome sequencing of *Drosophila pseudoobscura*: Chromosomal, gene and cis-element evolution. *Genome Research*, 15:1–18, 2005.

[49] A. Rokas and P.W.H. Holland. Rare genomic changes as a tool for phylogenetics. *Trends in Ecology and Evolution*, 15:454–459, 2000.

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

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

[52] Anne Bergeron, Julia Mixtacki, and Jens Stoye. A unifying view of genome rearrangements. In *Algorithms in Bioinformatics*, pages 163–173. Springer, 2006.

[53] 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, Hawaii, 2001.

[54] G. Bourque and P. Pevzner. Genome-scale evolution: reconstructing gene orders in the ancestral species. *Genome Research*, 12:26–36, 2002.

[55] T. Hill, A. Lundgren, R. Fredriksson, and H. Schioth. Genetic algorithm for large-scale maximum parsimony phylogenetic analysis of proteins. *Biochimica et Biophysica Acta*, 1725:19–29, 2005.

[56] A. Mitra, A. Almal, B. George, D. Fry, P. Lenehan, V. Pagliarulo, R. Cote, R. Datar, and W. Worzel. The use of genetic programming in the analysis of quantitative gene expression profiles for identification of nodal status in bladder cancer. *BMC Cancer*, 6:159, 2006.

[57] R. Unger and J. Moult. A genetic algorithm for 3d protein folding simulations. *The 5th International Conference on Genetic Algorithms*, 1993.

[58] M Srinivas and Lalit M Patnaik. Adaptive probabilities of crossover and mutation in genetic algorithms. *Systems, Man and Cybernetics, IEEE Transactions on*, 24(4):656–667, 1994.

[59] Yee Leung, Yong Gao, and Zong-Ben Xu. Degree of population diversity-a perspective on premature convergence in genetic algorithms and its markov chain analysis. *Neural Networks, IEEE Transactions on*, 8(5):1165–1176, 1997.

[60] J Andre, P Siarry, and T Dognon. An improvement of the standard genetic algorithm fighting premature convergence in continuous optimization. *Advances in engineering software*, 32(1):49–60, 2001.

[61] Yaochu Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft computing*, 9(1):3–12, 2005.

[62] A. Caprara. On the practical solution of the reversal median problem. In *Proc. 1st Workshop Algs. in Bioinformatics (WABI'01)*, volume 2149 of *Lecture Notes in Computer Science*, pages 238–251, 2001.

[63] J.H. Holland. Adaptation in natural and artificial systems, university of michigan press. *Ann Arbor, MI*, 1(97):5, 1975.

[64] D.E. Goldberg. Genetic algorithms in search, optimization, and machine learning. 1989.

[65] Günter Rudolph. Convergence analysis of canonical genetic algorithms. *Neural Networks, IEEE Transactions on*, 5(1):96–101, 1994.

[66] Sophia Yancopoulos, Oliver Attie, and Richard Friedberg. Efficient sorting of genomic permutations by translocation, inversion and block interchange. *Bioinformatics*, 21(16):3340–3346, 2005.

[67] Julia Mixtacki. Genome halving under dcj revisited. In *Computing and Combinatorics*, pages 276–286. Springer, 2008.

[68] Martin Bader. Genome rearrangements with duplications. *BMC bioinformatics*, 11(Suppl 1):S27, 2010.

[69] Sophia Yancopoulos and Richard Friedberg. Sorting genomes with insertions, deletions and duplications by dcj. In *Comparative Genomics*, pages 170–183. Springer, 2008.

[70] Marília DV Braga, Eyla Willing, and Jens Stoye. Genomic distance with dcj and indels. In *Algorithms in Bioinformatics*, pages 90–101. Springer, 2010.

[71] Marília DV Braga and Jens Stoye. The solution space of sorting by dcj. *Journal of Computational Biology*, 17(9):1145–1165, 2010.

[72] Eyla Willing12, Simone Zaccaria, Marília DV Braga, and Jens Stoye. On the inversion–indel distance.

[73] Phillip EC Compeau. A simplified view of dcj-indel distance. In *Algorithms in Bioinformatics*, pages 365–377. Springer, 2012.

[74] Sophia Yancopoulos and Richard Friedberg. Dcj path formulation for genome transformations which include insertions, deletions, and duplications. *Journal of Computational Biology*, 16(10):1311–1338, 2009.

[75] Sharon R Browning and Brian L Browning. Rapid and accurate haplotype phasing and missing-data inference for whole-genome association studies by use of localized haplotype clustering. *The American Journal of Human Genetics*, 81(5):1084–1097, 2007.

[76] Jaime E Blair, Kazuho Ikeo, Takashi Gojobori, and S Blair Hedges. The evolutionary position of nematodes. *BMC evolutionary biology*, 2(1):7, 2002.

[77] Emmanuelle Lerat, Vincent Daubin, and Nancy A Moran. From gene trees to organismal phylogeny in prokaryotes: the case of the γ-proteobacteria. *PLoS biology*, 1(1):e19, 2003.

[78] Antonis Rokas, Barry L Williams, Nicole King, and Sean B Carroll. Genome-scale approaches to resolving incongruence in molecular phylogenies. *Nature*, 425(6960):798–804, 2003.

[79] David Sankoff, Guillame Leduc, Natalie Antoine, Bruno Paquin, B Franz Lang, and Robert Cedergren. Gene order comparisons for phylogenetic inference: Evolution of the mitochondrial genome. *Proceedings of the National Academy of Sciences*, 89(14):6575–6579, 1992.

[80] K Müller, T Borsch, L Legendre, S Porembski, I Theisen, and W Barthlott. Evolution of carnivory in lentibulariaceae and the lamiales. *Plant Biology*, 6(4):477–490, 2004.

[81] José M Chaves-González, Miguel A Vega-Rodríguez, and José M Granado-Criado. A multiobjective swarm intelligence approach based on artificial bee colony for reliable dna sequence design. *Engineering Applications of Artificial Intelligence*, 2013.

[82] Sergio Santander-Jiménez and Miguel A Vega-Rodríguez. Applying a multiobjective metaheuristic inspired by honey bees to phylogenetic inference. *Biosystems*, 114(1):39–55, 2013.

[83] Eric Tannier, Chunfang Zheng, and David Sankoff. Multichromosomal median and halving problems under different genomic distances. *BMC bioinformatics*, 10(1):120, 2009.

[84] Andrew Wei Xu and Bernard ME Moret. Gasts: parsimony scoring under rearrangements. In *Algorithms in Bioinformatics*, pages 351–363. Springer, 2011.

[85] Fei Hu, Lingxi Zhou, and Jijun Tang. Reconstructing ancestral genomic orders using binary encoding and probabilistic models. In *Bioinformatics Research and Applications*, pages 17–27. Springer, 2013.

[86] Tobias Hill, Andor Lundgren, Robert Fredriksson, and Helgi B Schiöth. Genetic algorithm for large-scale maximum parsimony phylogenetic analysis of proteins. *Biochimica et Biophysica Acta (BBA)-General Subjects*, 1725(1):19–29, 2005.

[87] Anirban P Mitra, Arpit A Almal, Ben George, David W Fry, Peter F Lenehan, Vincenzo Pagliarulo, Richard J Cote, Ram H Datar, and William P Worzel. The use of genetic programming in the analysis of quantitative gene expression

profiles for identification of nodal status in bladder cancer. *BMC cancer*, 6(1):159, 2006.

[88] Ulrich HE Hansmann and Yuko Okamoto. New monte carlo algorithms for protein folding. *Current Opinion in Structural Biology*, 9(2):177–183, 1999.

[89] Nan Gao, Ning Yang, and Jijun Tang. Ancestral genome inference using a genetic algorithm approach. *PloS one*, 8(5):e62156, 2013.

[90] Jijun Tang and Bernard ME Moret. Linear programming for phylogenetic reconstruction based on gene rearrangements. In *Combinatorial Pattern Matching*, pages 406–416. Springer, 2005.

[91] Yu Lin, Vaibhav Rajan, and Bernard ME Moret. Fast and accurate phylogenetic reconstruction from high-resolution whole-genome data and a novel robustness estimator. *Journal of Computational Biology*, 18(9):1131–1139, 2011.