An Optimized Algorithm to Find Maximum Parsimonious Tree Using PrimeNucleotide Based Approach

Authors:

Rasif Ajwad (104410)
Syed Nayem Hossain (104413)

Supervisor:

Md. Abid Hasan
Lecturer
Department of Computer Science & Engineering (CSE)

A thesis submitted to the Department of CSE in partial fulfilment of the requirements for the degree of B.Sc. Engineering in CSE
Academic Year: 2013-2014
Semester: Summer

Department of Computer Science and Engineering (CSE)
Islamic University of Technology (IUT)
Gazipur, Dhaka, Bangladesh

Declaration of Authorship

This is to certify that the work presented in this thesis is the outcome of the analysis and investigation carried out by Rasif Ajwad and Syed Nayem Hossain under the supervision of Md. Abid Hasan in the Department of Computer Science and Engineering (CSE), Islamic University of Technology (IUT), Gazipur, Dhaka, Bangladesh. It is also declared that neither of this thesis nor any part of this thesis has been submitted anywhere else for any degree or diploma. Information derived from the published and unpublished work of others has been acknowledged in the text and a list of references is given.

Authors:

_____

Rasif Ajwad
Student ID: 104410

_____

Syed Nayem Hossain
Student ID: 104410

Supervisor:

_____

Md. Abid Hasan
Lecturer
Department of Computer Science and Engineering (CSE)
Islamic University of Technology (IUT)

Abstract

Molecular phylogeny based on the nucleotide or amino acid sequence comparison has become a widespread tool for general taxonomy and evolutionary analysis. Molecular phylogeny methods are often free of problems which arise while applying phenotypic phylogeny. So, we prefer phylogenetic methods for classifying the organisms in any evolutionary situation. Phylogenetic inference methods like Maximum parsimony perform exhaustive search strategy to extract evolutionary information from genomic sequences. It is a simple but popular technique used in cladistics to infer a phylogenetic tree for a set of taxa (commonly of species or reproductively isolated populations of a single species) on the basis of some observed data on the similarities and differences among taxa. The relationships among organisms or genes are studied by comparing the homologues of DNA and protein sequences. However, complexity arises when we increase the number of sequences involved, as the number of possible solutions increase exponentially alongside. In our paper, we have proposed an algorithm which identifies the highest repeating nucleotide (*PrimeNucleotide*) from the informative site efficiently to fix one *ParentNode* with the best fitted nucleotide using a predefined *WeightMatrix* to find the most parsimonious phylogenetic tree in linear time. The algorithm has been applied on the genome sequences of different bacteria and viruses to ensure its efficiency and universality. The results obtained were similar to the traditional Transverse parsimony method and a significant improvement in both time consumption and memory usage rate were achieved.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Phylogeny, derived from two Greek words, *phylon* (stem) and *genesis* (origin), is a process which gives an idea about the evolution or origin of an organism. Phylogeny is illustrated as a tree (Fig. 1). For a long time after people began trying to classify organisms in a systematic fashion, they used a variety of definitions of relationship. One definition was – things that look alike are more closely related to each other than things that look different. This is perhaps logical; but it is wrong, for the things resemble each other superficially. Apart from understanding the evolutionary relationships among the different groups of organisms, phylogeny also helps to understand the evolutionary history of organisms, map the pathogen strain diversity for vaccines, assist in the epidemiology of infectious diseases and genetic defects, biodiversity studies etc.

There are two types of phylogeny methods, namely, *phenotypic phylogeny* and *molecular phylogeny*. Phenotypic phylogeny is considered the traditional method of phylogeny as it is based upon phenotypic observations from the group of organisms. In due course of time, scientists found that in this method it was difficult to classify the micro-organisms because the phenotypic resemblance/dissimilarity may be superficial. All these paved the pathway for the arrival of the novel concept of molecular phylogeny. In this approach, the relationships among organisms or genes are studied by comparing the homologues of DNA and protein sequences. Thus, molecular phylogeny can be defined as the study of relationships among the organisms using molecular markers such as DNA or protein sequences.

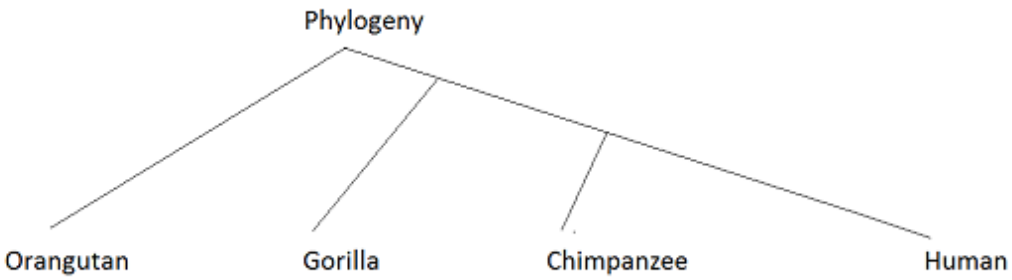

Figure 1: Phylogenetic tree of human beings and his ancestors

These two methods of phylogeny are related, since the genome strongly contributes to the phenotype of the organisms. In general, organisms with more similar genes are more closely related. However, phenotype based phylogeny or morphological phylogeny has many disadvantages over molecular phylogeny. For instance, there may be similar phenotypes in distantly related organisms due to the process called convergent evolution; phenotypic features for many organisms (e.g. bacteria) cannot be studied; it is difficult to compare the phenotypic traits with distantly related organisms (e.g. when comparing bacteria and mammals). Molecular phylogeny methods are often free of such problems and make possible the study of genes without a morphological expression.

Molecular phylogenetics, is the only means to establish a natural classification of micro-organisms, since their phenotypic traits are not always consistent with the genealogy or family pedigree. Though the structure and function of molecules change over time, there remains some similarity which suggests that the species descended from a common ancestor. As massive amount of genomic data are available to us through different sequence databases such as *GenBank*™ [1], molecular phylogenetics is continuing to grow and find new applications. [11, 12, 15, 16]

The most convenient way of visually presenting the evolutionary relationships among a group of organisms is through illustrations called *phylogenetic trees* (Fig. 2). A Tree is a mathematical structure which is used to model the actual evolutionary history of a group

of sequences or organisms. A phylogenetic tree is composed of *nodes*, each representing a taxonomic unit (species, populations, individuals), and *branches*, which defines the relationship between the taxonomic units in terms of descent and ancestry. The branching pattern of the tree is called the *topology*, and the branch length usually represents the number of changes that have occurred in the branch. Other terminologies include *root*, the common ancestor of all taxa; *Operational Taxonomic Unit (OTU)*, any group of organisms, populations, or sequences considered to be sufficiently distinct from each other and is treated as a separate unit etc.

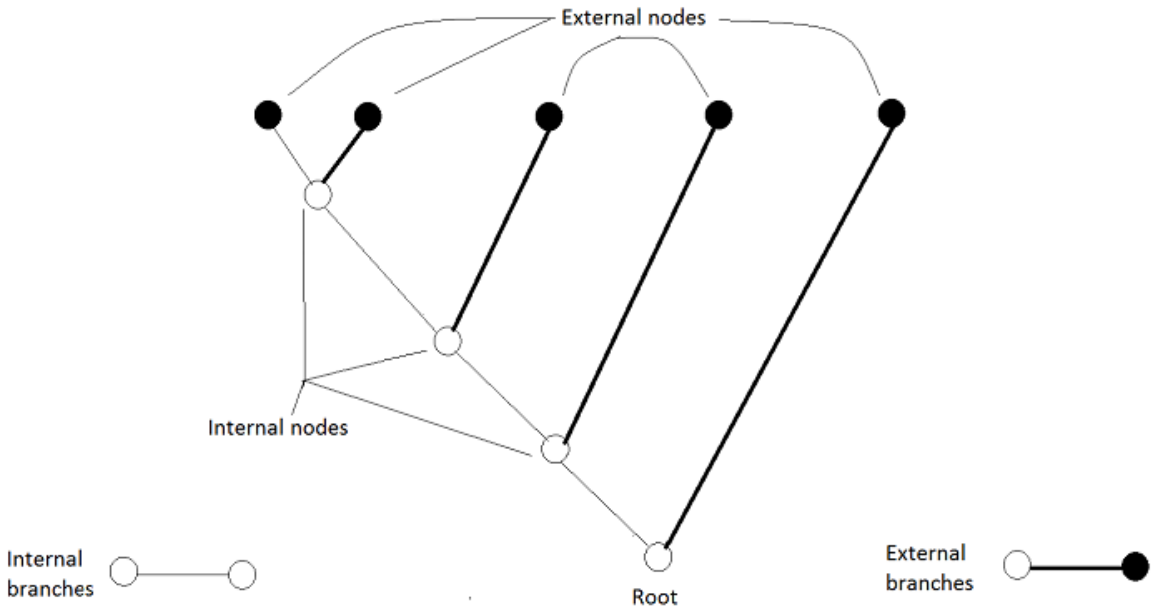

Figure 2: Different parts of a phylogenetic tree

There are some predefined steps for phylogenetic tree construction. In phylogenetics, different genes or combinations of genes or DNA regions may be used to infer phylogenetic trees while addressing groups of organisms. These are called *phylogenetic markers.* Then, sequencing of desired phylogenetic marker can be done which will be the input for the

various phylogenetic packages. After that, since descendants inherit traits from their ancestors through genes, the history of descent is recorded in the changes within the DNA sequences. The molecular data on sequences in the genes are a simple form of character data: the characters are positions in the sequences, and the characters' states are the nucleotides at those positions. This idea has been conceived and put in multiple alignment concepts. Next, we need to choose an evolutionary model, which includes several parameters such as base frequencies, substitution rate matrix, gamma distribution, proportion of invariable sites etc. The next step is to choose a method to construct phylogenetic trees. Different methods are discussed in the next chapter. Finally, we need to perform statistical evaluation of the obtained phylogenetic tree. The tree which is generated by the methods needs to be tested or evaluated statistically. The steps are shown in Fig 3.

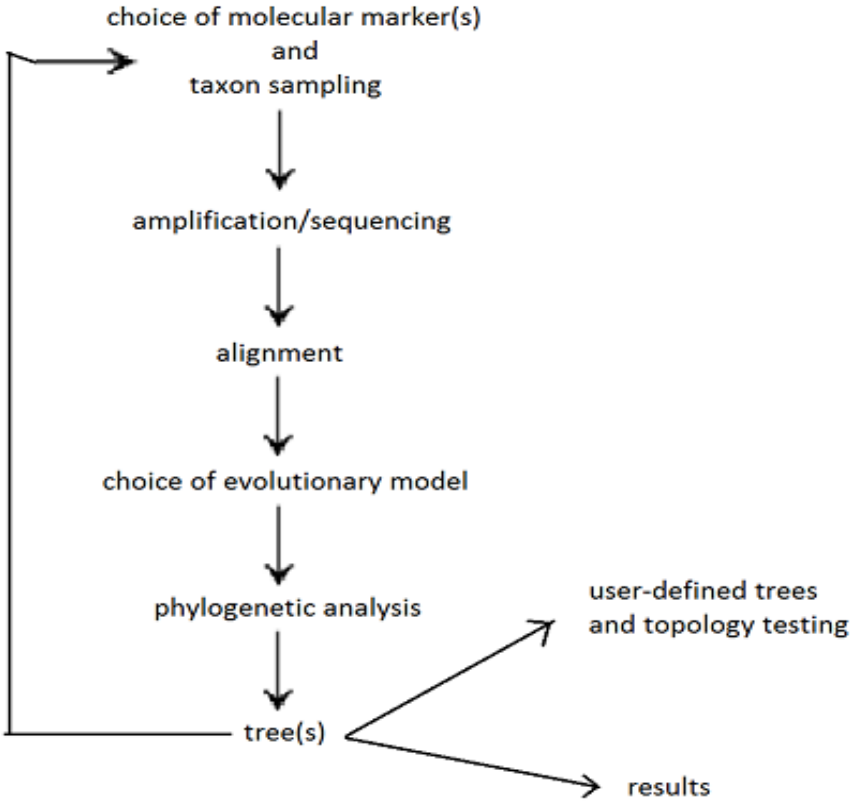

Figure 3: Basic steps for phylogenetic tree construction

Molecular phylogenetic studies aim to recover the order of evolutionary events. The events are represented graphically as evolutionary trees called phylogenetic trees. These trees depict relationships among species or genomes over time. The process is computationally complex, and there is no right way to approach the problems. Hundreds of different species comprise of phylogenetic data, with varying mutation rates and patterns that influence evolutionary change. As a result, there are numerous different evolutionary models and stochastic methods available. The correctness of the methods depends on the nature of the study and the data used. [12-14] Different methods of phylogenetic inference have been proposed upon examining the collected genomic data. Most of these methods are based on some *optimization principle*. [2-3] Under this principle, the phylogenetic tree inferred from the sequence of different genes is determined. Each of these tree topologies is assigned a particular score and the topology with the highest or lowest optimal score is chosen as the result.

However, the available methods can be classified into two categories. The *exhaustive-search* strategy examines a large number of possible trees and chooses the best one based on the optimal score. In contrast, the *stepwise clustering* method, examines local topological relationships of a tree and construct the best one step by step. Methods like Maximum-Parsimony (MP) [4-5], Fitch-Margoliash (FM) [6], Maximum-Likelihood (ML) [7] and Minimum-Evolution (ME) [8] belong to the first category whereas the Neighbor-Joining (NJ) method [9] and other distance methods [10] belong to the second.

Chapter 2

Literature Reviews

This chapter is divided into different sections. In each sections, we will discuss some proposed molecular phylogenetic methods under two different categories (exhaustive-search method and stepwise clustering method) mentioned previously. Along with their description, we will try to highlight the shortcomings of each of the discussed methods.

2.1 Maximum Parsimony (MP) Method

Maximum Parsimony (MP) [4-5] is a simple but popular technique used in cladistics to infer a phylogenetic tree for a set of taxa on the basis of some observed data on the similarities and differences among taxa. In other words, the principle of MP searches for a tree that requires *the smallest number of evolutionary changes* to explain the differences observed among OTUs.

The input data used in a maximum parsimony analysis is in the form of 'characters' for a range of taxa. A character is a partitioning of the taxa into distinct character states with respect to some feature. Differences in the character states are explained by the evolutionary changes. In mathematical terms, from the set of possible trees, finding all trees $\boldsymbol{\tau}$ such that $L_{\boldsymbol{\tau}}$ is minimal:

$$L_{\boldsymbol{\tau}} = \sum_{k=1}^{B} \sum_{j=1}^{N} \omega_j \, . \, diff(x_{k'j}, x_{k''j})$$

Equation 1: Finding the minimal tree in MP method

Here, $L_{\boldsymbol{\tau}}$ is the length of the tree, B is the number of branches, N is the number of characters, k' and k'' are the two nodes incident to each branch k, $x_{k'j}$ and $x_{k''j}$ represent either element of the input data matrix or optimal character-state assignments made to internal nodes, and $diff(y, z)$ is a function specifying the cost of a transformation from state y to state z along any branch. The coefficient ω_j assigns a weight to each character.

The trees used in maximum parsimony analysis are, in a general way, unrooted trees (there is no indication of time in the tree, only the relations between taxa). All the taxa used in the analysis are leaf nodes (often called tips or terminal taxa) in the (leaf-labelled) tree (so they have only one edge). Internal nodes are inserted into the tree to represent the inferred ancestral species. Each internal node has at least three edges into it. The transitions between character states are associated with the edges on the tree.

All inferences in comparative biology depend on accurate estimates of evolutionary relationships. Recent phylogenetic analyses have turned away from maximum parsimony towards the probabilistic techniques of maximum likelihood and Bayesian Markov Chain Monte Carlo (BMCMC) [21]. These probabilistic techniques represent a parametric approach to statistical phylogenetics, because their criterion for evaluating a topology — the probability of the data, given the tree — is calculated with reference to an explicit evolutionary model from which the data are assumed to be identically distributed.

2.2 Unweighted Pair Group Method with Arithmetic Mean (UPGMA)

The UPGMA is the simplest method of tree construction. It is a cluster analysis derived from the clustering algorithms popularized by Sokal and Sneath (1973). It was originally developed for constructing taxonomic phonograms, i.e., trees that reflect the phenotypic similarities between the OTUs, but it can also be used to construct phylogenetic trees if the rates of evolution are approximately constant among the different lineages. For this purpose the number of observed nucleotide or amino-acid substitutions can be used. UPGMA employs a sequential clustering algorithm, in which the local topological relationships are identified in the order of similarity, and the phylogenetic tree is built in a step-wise manner. We first identify from among all the OTUs the two OTUs that are most similar to each other and then treat these as a new single OTU. Such an OTU is referred to as a composite OUT. Subsequently from among the new group of OTUs we identify the pair with the highest similarity, and so on, until we are left with only two OTUs. This method is the least accurate but widely used.

2.3 Fitch-Margoliash (FM) Method

Fitch-Margoliash (FM) [6] method follows a common pair-wise clustering algorithm which yields an unrooted tree and unlike other clustering methods it does not proceed by adding taxa one at a time to a growing tree. Rather it has an optimum criterion that must be met. This method attempts to find the tree which minimizes the following sum:

$$\sum \frac{(d - d')^2}{d^2}$$

Equation 2: Finding the tree with minimal sum in FM method

Here d is the observed distance and d' is the expected distance given some phylogeny and assuming additivity between all the branch lengths.

FM method is similar of UPGMA, a popular phylogenetic method except that it compares taxa, distances in groups of three using the given distance relationships and uses composite OTUs, which are averages of all OTUs present in doing this. Among the advantages of this method, the primary one is that it can use empirical substitution scoring methods. Moreover global optimization of trees are done by statistical criteria.

Though FM method requires simple calculation, it requires longer execution time. This method does not consider intermediate ancestors, meaning that there is no requirement for an internally-consistent evolutionary model. Furthermore, this method misses *homoplasies* (independent derivation of a character state in two lineages), especially over long-distances, long evolutionary distances will be underestimated.

2.4 Maximum Likelihood (ML) Method

Maximum Likelihood (ML) [7] method creates all the possible trees containing the set of organisms considered, and then use the statistics to evaluate the most likely tree. From a small number of organisms, this is possible. For as large number of organisms, the task cannot be accomplished as the number of generated trees is very large. Therefore, *heuristics* (methods which produce an answer in a computable length of time, but for which the answer may not be optimal) are used to select a subset of trees to create. In this method, the bases (nucleotides or amino acids) of all the sequences at each site are considered separately, and the *log-likelihood* of having these bases are computed for a given topology by using a particular probability model.

Few main features of ML method include statistical method for inferring the phylogenies (substitution model is chosen for the sequence data, topology that gives the highest likelihood is chosen as the best tree), the method is very dependent on the model

of substitution used, and the method estimates the branch lengths not topology, so it may give the wrong topology.

Though ML method uses all the sequence information and usually consistent, it is very CPU intensive and thus extremely slow. Moreover, it needs long computation time to construct a tree. The result depends on the model of evolution used, which may not best-fitted always.

2.5 Minimum Evolution (ME) Method

Given an unrooted metric tree for n sequences there are $(2n-3)$ branches, each with length e_i. The sum of these branch lengths is the length L of the tree:

$$L = \sum_{i=1}^{2n-3} e_i$$

Equation 3: Finding the length of an unrooted tree

Minimum Evolution (ME) [8] method produces a tree which minimizes L. This method is similar in spirit to parsimony, however, the length in this case is computed from the pair-wise distances between the sequences rather than from the fit of individual nucleotide sites to a tree. The trees obtained by parsimony and ME method are identical in topology and branch lengths.

ME method is easy to perform, calculation are quicker and fit for sequences having high similarity rates. But the sequences are not considered as such and so there is loss of information. All sites are generally equally treated though there are differences in substitution rates. Furthermore, the method is not applicable to distantly divergent sequences.

2.6 Neighbour-Joining (NJ) Method

Neighbour-Joining [9] belongs to the stepwise clustering methods of phylogenetic inference. This method attempts to correct the UPGMA method for its strong assumption that the same rate of evolution applies to each branch. Hence, this method yields an unrooted tree. A modified distance matrix is constructed to adjust the differences in the rate of evolution of each taxon. Similar to the UPGMA method, the least distant pairs of nodes are linked and their common ancestral node is added to the tree; their terminal nodes are pruned from the tree. This continues until only two nodes remain.

NJ method is fast, suited for large datasets and for bootstrap analysis. It permits lineage with largely different branch lengths and permits correction for multiple substitutions. But in this method the sequence information is reduced. It gives only one possible tree and strongly dependent on the model of evolution used.

Chapter 3

Our Motivation

The Maximum-parsimony (MP) method can be treated as the most widely used sequence-based tree reconstruction method as it uses the simplest, most parsimonious explanation from the observed sequences, until new sequences arise to adopt a more complex result. In this method, the correct topology is computed as tree length (TL) which denotes the minimum number of evolutionary changes. The topology showing the smallest TL value is finally selected as the preferred tree (*MP tree*).

Maximum parsimony can be considered *nonparametric*, because trees are evaluated on the basis of a general metric—the minimum number of character state changes required to generate the data on a given tree—without assuming a specific distribution. The shift to parametric methods was spurred, in large part, by studies showing that although both approaches perform well most of the time, maximum parsimony is strongly biased towards recovering an incorrect tree under certain combinations of branch lengths, whereas maximum likelihood is not.

Like most phylogenetic methods, MP produces unrooted trees. This is because they detect the differences between the sequences, but shows only the evolutionary relationships between organisms in the tree. It cannot actually infer the placement of a common ancestor in the structure of the evolutionary path used to obtain the current relationships, the direction of the evolutionary process is not given. However, the number of unrooted trees that have to be analyzed rapidly increases with the number of Operational Taxonomic Units (OTUs). (Table 1)

Table 1: Complexity of Maximum Parsimony (MP) method

Number of OTUs	Number of unrooted trees
2	1
3	1
4	3
5	15
6	105
7	954
8	10,395
9	135,135
10	34,459,425
15	2.13E15

The number of unrooted trees (N_u) for n OTUs follows the equation below:

$$N_u = \frac{(2n-5)!}{2^{n-3}\,(n-3)!}$$

Equation 4: The number of unrooted trees for n OTUs

This rapid increase in number of trees to be analyzed may make it very difficult to apply the method to very large datasets. In that case the MP method may become very time consuming, even on very fast computers.

The goal of this paper is to develop an improved algorithm for MP method where the number of trees will be much fewer, where instead of all possible trees only some selected trees will be considered as potential best tree. To observe the efficiency of our proposed algorithm, DNA sequences of different lengths were collected from NCBI (National Center for Biotechnology Information) databases. The system's response time was also calculated and compared with the traditional exhaustive MP method.

Chapter 4

Proposed Methodology

Maximum-parsimony (MP) method search for the phylogenetic tree that has undergone the least number of evolutionary changes. This optimal tree explains the differences observed among various Operational Taxonomic Units (OTUs). Different types of weighing schemes are used to calculate the optimal score. In the *Transversion Parsimony* (TP) scheme, a cost is assigned for each transition from one nucleotide base (A, C, G or T) to another. Transition to the same nucleotide state (such as A↔A, T↔T, C↔C or G↔G) has a cost of zero (0). The cost is different for other transactions: It is four (4) for *puRine↔pYrimidine* (A↔C, A↔T, G↔C or G↔T) transactions and, one (1) for *puRine→puRine* (A↔G) or *pYrimidine→pYrimidine* (C↔T) transaction. The transaction costs can be represented by the following *WeightMatrix* (Fig. 11):

	A	C	G	T
A	0	4	1	4
C	4	0	4	1
G	1	4	0	4
T	4	1	4	0

Figure 4: WeightMatrix for finding maximum parsimonious tree

The next task is to choose the informative site among the OTUs. Informative site denotes the position in the relevant set of nucleotide sequences at which there are at least

two different transaction states at that point in the sequences, and each of those states occurs in at least two of the sequences. The informative site is used to create all possible tree topologies. The score of each topology can be calculated on the basis of the given weight matrix. The tree containing the minimum weight score is selected as the desired TP tree. However, the number of trees increases rapidly if the informative site is too long. As a result, the method becomes both complex and time consuming, regardless of the computational power of the machine used.

We have developed a WeightMatrix based Tree Search algorithm (WTS) which emphasizes on developing a heuristic method to find the optimal TP tree and it will be less time consuming as well as memory efficient. The proposed algorithm does not need to create all possible combinations of the phylogenetic trees to find out the tree with the minimum evolutionary changes. Taking the characteristics of the weight matrix into account, our algorithm emphasizes on reducing the tree combinations. As we are calculating weight score for considerably fewer combinations, the desired tree can be found more efficiently.

4.1 Materials and Methods

In order to find the desired Transversion Parsimony (TP) tree, first we need to detect the informative site among the OTUs. Once the informative site has been chosen, we apply our proposed algorithm (Algorithm 1). At first we detect the maximum occurring nucleotide in the informative site (*PrimeNucleotide*). Keeping this *PrimeNucleotide* fixed as one of the *ParentNode* (either *LeftNode* or *RightNode*) of our desired tree, we will take all possible nucleotides (A, C, G, and T) as the other one. We then calculate the TP score for each combination separately which indicates the final weight score of the tree. The cost assigned for each transition from one nucleotide base to another is named *SeqValue*, which is the weight value obtained from the weight matrix for each transactions. The

corresponding *SeqValue*s for *LeftNode* ↔ *RightNode* transition is calculated and added to the corresponding TP score.

As we have one of our *ParentNode*s fixed, we take each nucleotide *N* from the previously chosen informative site and decide whether to make *N* a child of *LeftNode* or *RightNode*. This decision is dynamic and tends to give the lowest possible TP score for the selected topologies. We make *N* a child of *LeftNode* if *SeqValue (LeftNode,N)* < *SeqValue (RightNode,N)* or a child of *RightNode* otherwise. After joining the nucleotide (*N*) with its *ParentNode*, the *SeqValue (ParentNode, N)* is calculated from the *WeightMatrix* and added to the total TP score. The selected tree topologies are then compared based on their corresponding TP score. The topology with the minimum TP score is our desired topology. Unlike the traditional MP method, we don't need to consider every possible topology. For example, in MP method, for an informative site of length four (*WXYZ*), the unrooted trees can be constructed in three different fashions such as *((W, X), (Y, Z)); ((W, Z), (Y, X))* or *((W, Y), (X, Z))*. But in case of our algorithm, a specific pattern is followed while constructing trees: each nucleotide is added from the informative site to one of the *ParentNode*s where it is best fitted.

4.2 Algorithm 1: WeightMatrix-based Tree Search for MP method

In order to find the desired *Transversion Parsimony* (TP) tree, first we need to generate the *WeightMatrix* in our JAVA simulation Program. We are considering all the possible pairs among the nucleotides and assign the respective value to the nucleotide pair. The value which is taken as consideration is directly imported from the *WeightMatrix*.

We have also built a class ParentCategory for tracing for initializing the system which actually representing a ParentNode of the system. It has the needed variables and the setter and getter method for accessing the values whenever it is necessary for calculation.

In maximum parsimony, an informative site is a position in the relevant set of sequences at which there are at least two different character states at that point in the sequences, and each of those states occurs in at least two of the sequences. In othe words, a site is informative only when there are at least two different kinds of nucleotides at the site, each of which is represented in at least two of the sequences under study.

For example, if we have four inputs DNA sequences such as:

SeqW: ACA**G**GAT

SeqX: ACA**C**GTC

SeqY: GTA**A**GGT

SeqZ: GCA**C**GAC

The informative site is **GCAC** (in bold font) as in this specific position (4th for this example) of the sequences the number of nucleotide substitution is maximum. According to our proposed algorithm, the *PrimeNucleotide* is **C** as it is repeated highest number of times (two in this example) in the informative site.

Determining informative site is the first step of finding maximum parsimonious tree. To do this, at first, we check each index of the available *Operational Taxonomic Units* (OTUs). And when we detect the change among to adjacent OTU nucleotide of same index, we increment the counter value. After scanning all nucleotides among the different OTUs in a given index, if it occurs that the counter value is greater than the previous OTU nucleotides, and then we detect it as an informative site. This process will continue until the end of all OTUs has been scanned.

The next step for our proposed method is finding the highest occurring nucleotide seen or appeared in the informative site. The main reason of finding the highest occurring nucleotide is to fix that nucleotide as one of our parent node. The good reason behind this step is, as we can see, in the *WeightMatrix*, the weight between two same nucleotides is zero (0). Considering this property into account, if we create the tree with more edge containing zero, then the least weight tree will be formed.

For now, we have only have determined one *ParentNode* of our desired parsimonious tree which is the highest occurring nucleotide in the informative site. Now, we have to choose another parent node. As we have to overall scope on the datasets, so we are considering four possible combinations of nucleotides (**A, C, T,** and **G**) as the other *ParentNode*. It is to be mentioned that, when we choose all possible combination of nucleotide as other parent node, then we need to add the weight between the *ParentNode*s to the total TP score as it is a part of the tree. Up to now, we are forming four trees whose one *ParentNode* is the highest occurring nucleotide in the informative site (as we call it *PrimeNucleotide)* and other *ParentNode* is all possible combinations of nucleotides (**A, C, T, G**) and add the weight to the total TP score of the respective trees .

Now, we have to form the tree in an optimized manner. We want to add a child of the tree where it is best fitted. As we have one of our *ParentNode*s fixed, we take each nucleotide *N* from the previously chosen informative site and decide whether to make N a child of *LeftNode* or *RightNode*. This decision is dynamic and tends to give the lowest possible TP score for the selected topologies. We make *N* a child of *LeftNode* if *SeqValue (LeftNode,N) < SeqValue (RightNode,N)* or a child of *RightNode* otherwise. After joining the nucleotide (*N*) with its *ParentNode*, the *SeqValue (ParentNode,N)* is calculated from the weight matrix and added to the total TP score.

We have created four trees (whose one *ParentNode* is the *PrimeNucleotide* and the other parent nodes are **A, C, T, G**) and add the children (nucleotides in the informative site) in an optimized manner, now we have to choose the best maximum parsimonious (MP) tree among the four trees. Clearly, the tree which has the lowest TP score is our best maximum parsimonious (MP) tree.

The formal algorithm for finding maximum parsimonious tree using *PrimeNucleotide* based approach is given in Table 2. The input of our proposed algorithm is a set of test sequences (OTUs) to choose informative sites and the output is the desired minimum length *Transversion Parsimony* (TP) tree.

We have used 20 experimental dataset of bacteria and viruses for testing our proposed algorithm. Table 3 will show the training data sets' scientific names, their accession numbers. All these data has been collected from the NCBI (**N**ation **C**entre for **B**iotechnology **I**nformation) database. The corresponding accession numbers are universal for each and every sequence.

Table 2: WeightMatrix-based Tree Search algorithm for MP using PrimeNucleotide based approach

Algorithm 1

Input: A set of test sequences (OTUs) to choose informative sites.
Output: Minimum length Transversion Parsimony (TP) tree.

1: Identify the most informative site form the given input sequences.

2: Get the highest occurring nucleotide (PrimeNucleotide) from the chosen informative site.

3: For all possible tree topologies, set PrimeNucleotide as ParentNode to do:
- **3.1**: Consider all possible nucleotide (**A, C, G,** and **T**) as the other ParentNode.
- **3.2**: Add the corresponding SeqValue to the total TP Score.

4: For each nucleotide N from the informative site to do:
- **4.1**: For each possible combination keeping PrimeNucleotide as ParentNode to do:
  - **4.1.1**: Add N to the LeftNode if SeqValue (LeftNode, N) < SeqValue (RightNode, N).
  - **4.1.2**: Else add N to the RightNode.
- **4.2**: Add the SeqValue to the TP score of their corresponding combination.

5: The tree Containing lowest TP score is the Minimum length Transversion Parsimony (TP) tree.

Table 3: Training datasets for the experiment

Experimental Dataset	NCBI Reference Sequence:	Data
Acetohalobium arabaticum DSM 5501 chromosome, complete genome	NC_014378.1	ATTATTA... TGTT
Acidothermus cellulolyticus 11B chromosome, complete genome	NC_008578.1	GATTCCTA…CAGT
Acidimicrobium ferrooxidans DSM 10331 chromosome, complete genome	NC_013124.1	GACTCGTC…TGAT
Acidaminococcus fermentans DSM 20731 chromosome, complete genome	NC_013740.1	AAAAAATC…ATGA
Acidovorax avenae ATCC 19860 chromosome, complete genome	NC_015138.1	TTATCACA...GCGA
Acidovorax ebreus TPSY chromosome, complete genome	NC_011992.1	TAACTCCT...TGGA
Acetobacterium woodii DSM 1030 chromosome, complete genome	NC_016894.1	TTATTTGG...TGAT
Acaryochloris marina MBIC11017 chromosome, complete genome	NC_009925.1	AATAAATA...ATTT
Actinobacillusn suis H91-0380 chromosome, complete genome	NC_018690.1	GTATTGAC….GTT
Actinobacillus succinogenes 130Z chromosome, complete genome	NC_009655.1	TTAGGAAC…AAA
Acidaminococcus intestini RyC-MR95 chromosome, complete genome	NC_016077.1	CAAAGTCA…TGCA

Acholeplasma laidlawii PG-8A chromosome, complete genome	NC_010163.1	TATTTGAT… TTAT
Acidithioba cilluscaldus SM-1 chromosome, complete genome	NC_015850.1	ATGAGTA...CAAC
Achromobacter xylosoxidans A8 chromosome, complete genome	NC_014640.1	ATGAAAGA...GCGT
Achromobacter xylosoxidans NBRC 15126 = ATCC 27061, complete genome	NC_023061.1	ATGAAAGA...GCGT
*Acetobacter pasteurianus*386B, complete genome	NC_021991.1	AATGGGTA…ACGT
Acetobacter pasteurianus IFO 3283-01, complete genome	NC_013209.1	ACTGCAGG…GTGT
Acetobacter pasteurianus IFO 3283-01-42C, complete genome	NC_017150.1	ACTGCAGG…GTGT

Chapter 5

Performance Evaluation

5.1 Obtaining Results

Various DNA sequences were taken as our experimental dataset and the collected dataset were divided into four parts to conduct four separate experiments. The DNA sequences were all taken from **NCBI** database (**N**ational **C**entre for **B**iotechnology **I**nformation). We used NCBI reference sequence to uniquely identify various genome sequences for different species of bacteria and viruses. We tried to determine the correct phylogenetic topology for both TP scheme and our proposed algorithm. After applying our proposed algorithm, we applied the traditional Transverse Parsimony (TP) algorithm to the collected datasets and seen that both of these methods produce same results for the same dataset. This ensures the validity of our proposed algorithm. We have grouped these datasets into four, five and six OTUs so that uniformity is maintained and the performance evaluation can be seen more clearly as the length of the informative site increases, our proposed algorithm will work more efficiently. The results of the experiments are shown below in Table III. It has been observed that for every set of experimental data we have achieved exact same results while comparing our proposed method with the traditional one.

Table 4: Obtaining result both in TP method and our proposed method

Experiment No	Dataset used	TP Scheme	Our Proposed Method
1	NC_014378.1, NC_008578.1, NC_013124.1, NC_013740.1	Left Node: A Right Node: C TP Score: 6	Left Node: A Right Node: C TP Score: 6
2	NC_015138.1, NC_008752.1, NC_011992.1, NC_016894.1, NC_009925.1	Left Node: A Right Node: G TP Score: 5	Left Node: A Right Node: G TP Score: 5
3	NC_018690.1, NC_009655.1, NC_016077.1, NC_010163.1	Left Node: T Right Node: C TP Score: 5	Left Node: T Right Node: C TP Score: 5
4	NC_015850.1, NC_014640.1, NC_023061.1, NC_021991.1, NC_013209.1, NC_017150.1	Left Node: A Right Node: C TP Score: 6	Left Node: A Right Node: C TP Score: 6

5.2 Performance Evaluation

To measure the performance of the proposed algorithm, we compared the memory usage of TP scheme with it. The experiments were carried on an Intel Core™ i3 processor, 4GB RAM machine. The comparison shows that if the number of datasets is higher, the memory usage of WTS significantly decreases in contrast with the traditional TP scheme. The comparison results are shown in Fig. 5.

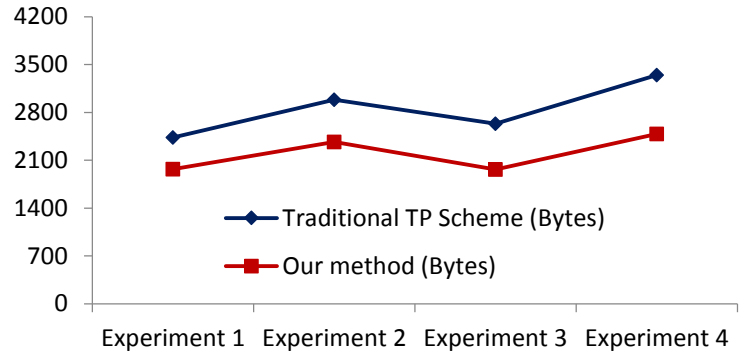

Figure 5: Comparison between TP scheme and proposed algorithm

(In terms of memory usage)

We also compared the proposed algorithm with TP scheme in the basis of time consumption. These experiments were also carried out in the machines with same configuration and in an idle state. It has been observed that, the time consumption of our method is always much less than that of TP scheme. The comparison results are shown in Fig. 6.

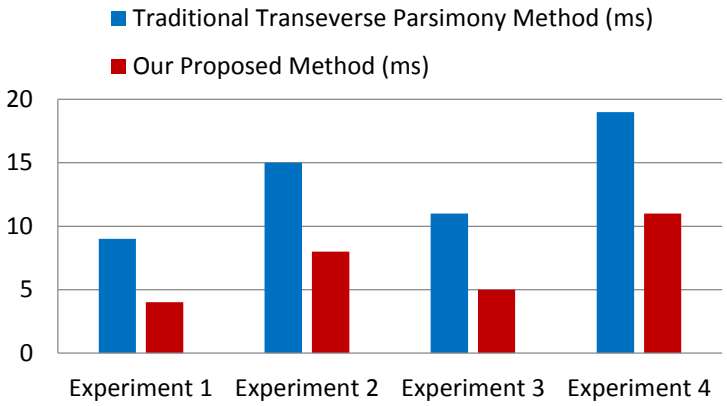

Figure 6: Comparison between TP scheme and proposed algorithm

(in terms of time consumption)

In TP method, all possible combination of tree topologies is needed to be calculated. For example, if the informative site is of length four (WXYZ), it does not necessarily mean

that trees should always be constructed in ((W, Y), (X, Z)) fashion. Other than that, the topologies can also be made by ((W, X), (Y, Z)) or ((W, Z), (Y, X)) fashion. Taking all these possible construction fashions into consideration, if the length of the informative site is n, then the number of phylogenic trees P for TP scheme is:

$$P = \frac{(2n-5)!}{2^{n-3}(n-3)!} \times {}^{4}C_{2}$$

Equation 5: Finding the number of phylogenetic trees generated in TP scheme

:

Our proposed method greatly reduces the number of phylogenetic trees need to be constructed compared to the traditional TP scheme as in our algorithm as one of the *ParentNode*s are kept fixed on the basis of informative site rather than going for an exhaustive strategy by taking all possible tree combinations into consideration. Our algorithm also places each nucleotide of informative site in such a position where it seems to be best fitted. So, if the length of the informative site is n, then the proposed method will produce much less number of phylogenic trees from which the best topology is then selected. In more details, by fixing *PrimeNucleotide* as one of the *ParentNode*, we are actually reducing the number of sample trees from 16 to 4. Again, by placing the child to an optimized position, we are reducing the computation greatly. . Because, the number of unrooted trees that have to be analyzed increases with the number of Operational Taxonomic Units (OTUs).

Chapter 6

Conclusion

6.1 Summary of the research

This thesis proposed an optimized algorithm for finding the maximum parsimonious tree with the highest information regarding evolutionary changes.

Our algorithm takes input a sequence of OTUs to choose informative sites among them. After choosing the most informative site (the site with the highest number of changes through the entire evolutionary process), we select the *PrimeNucelotide* from the chosen informative site. The *PrimeNucleotide* serves as the key contributor of our proposed algorithm, as according to this *PrimeNucleotide* we choose one of our *ParentNodes* (*LeftParent* or *RightParent*).

After choosing one of the *ParentNodes* we construct all possible tree combinations according to that particular structure. We ten calculate the total weight values (*SeqValues*) for each of them to find the most parsimonious tree (the tree with the minimum weight). The obtained tree(s) is/are the minimum length Transversion Parsimony (TP) tree.

This system could be used for finding the maximum parsimonious tree from various sizes of DNA sequences. DNA sequences of different lengths were used to calculate the accuracy of our algorithm. These sequences varied in size from very small to very large. Our proposed system has some significant advantages over the methods discussed previously in Chapter 1 and 2. The key features of our proposed algorithm are:

- Requires less memory than the traditional Transverse Parsimony (TP) method.
- The time consumption rate is significantly less.
- Our algorithm is dynamic as we do not need to consider all possible topologies into account.
- The complexity of our algorithm is not exponential, so the number of unrooted trees that have to be analyzed does not rapidly increase with the number of Operational Taxonomic Units (OTUs).

6.2 Future works

In this thesis, we proposed a simple algorithm for finding the maximum parsimonious tree altering a popular phylogenetic inference method (Maximum Parsimony).

In future, we are interested to work on examining the efficiencies of various search algorithms for MP, ME, and ML trees when the number of sequences used is large and to find simple algorithms for inferring the true tree with a reasonably high level of accuracy. We want to study these methods by computer simulation and by examining the relationships between the efficiencies of inferring the optimal and the true trees. We also want to examine the efficiencies of inferring the true tree when wrong nucleotide substitution models are used.

Appendix

JAVA Simulation Codes of Proposed Method

The JAVA simulation codes of our proposed algorithm are given below:

File: ParentCategory.java

```
1  /*
2   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5  package tree;
6
7  /**
8   *
9   * @author ManHunt
10  */
11 public class ParentCategory {
12     int leftValue;
13     int rightValue;
14     int centerValue;
15     int totalValue;
16
17     char leftNode;
18     char rightNode;
19
20     public ParentCategory() {
21         this.leftValue = 0;
22         this.rightValue = 0;
23         this.centerValue = 0;
24         this.totalValue = 0;
25     }
26
27
```

```
28
29
30      public void setLeftNode(char leftNode) {
31          this.leftNode = leftNode;
32      }
33
34      public void setRightNode(char rightNode) {
35          this.rightNode = rightNode;
36      }
37
38      public char getLeftNode() {
39          return leftNode;
40      }
41
42      public char getRightNode() {
43          return rightNode;
44      }
45
46
47
48      public int getCenterValue() {
49          return centerValue;
50      }
51
52      public int getLeftValue() {
53          return leftValue;
54      }
55
56      public int getRightValue() {
57          return rightValue;
58      }
```

```
59
60      public int getTotalValue() {
61          return leftValue + rightValue + centerValue;
62      }
63
64      public void setCenterValue(int centerValue) {
65          this.centerValue = centerValue;
66      }
67
68      public void setLeftValue(int leftValue) {
69          this.leftValue = leftValue;
70      }
71
72      public void setRightValue(int rightValue) {
73          this.rightValue = rightValue;
74      }
75
76
```

```
77      //ASSIGNING TO MIN...
78
79      void addAndCalculate(char givenChar) {
80          SequenceValue secVal = new SequenceValue();
81
82          if(secVal.getSequenceValue(leftNode, givenChar) < secVal.getSequenceValue(rightNode, givenChar)){
83                  this.setLeftValue(this.getLeftValue() + secVal.getSequenceValue(leftNode, givenChar));
84          }
85          else this.setRightValue(this.getRightValue() + secVal.getSequenceValue(rightNode, givenChar));
86      }
87
88
89
90
91      void addAndCalculate(char givenChar, int serial){
92          SequenceValue secVal = new SequenceValue();
93  if(serial % 2 == 0)this.setLeftValue(this.getLeftValue() + secVal.getSequenceValue(leftNode, givenChar));
94          else this.setRightValue(this.getRightValue() + secVal.getSequenceValue(rightNode, givenChar));
95      }
96
97
98
99
100
101 }
102
```

File: SequenceValue.java

```
1   /*
2    * To change this template, choose Tools | Templates
3    * and open the template in the editor.
4    */
5   package tree;
6
7   /**
8    *
9    * @author ManHunt
10   */
11  public class SequenceValue {
12
13
14      int sequenceValue;
15
16      public int getSequenceValue(char leftNode, char rightNode) {
17
18          if(leftNode=='A' && rightNode=='A') return 0;
19          else if(leftNode=='A' && rightNode=='C') return 4;
20          else if(leftNode=='A' && rightNode=='G') return 1;
21          else if(leftNode=='A' && rightNode=='T') return 4;
22
```

```
23            else if(leftNode=='C' && rightNode=='A') return 4;
24            else if(leftNode=='C' && rightNode=='C') return 0;
25            else if(leftNode=='C' && rightNode=='G') return 4;
26            else if(leftNode=='C' && rightNode=='T') return 1;
27
28            else if(leftNode=='G' && rightNode=='A') return 1;
29            else if(leftNode=='G' && rightNode=='C') return 4;
30            else if(leftNode=='G' && rightNode=='G') return 0;
31            else if(leftNode=='G' && rightNode=='T') return 4;
32
33            else if(leftNode=='T' && rightNode=='A') return 4;
34            else if(leftNode=='T' && rightNode=='C') return 1;
35            else if(leftNode=='T' && rightNode=='G') return 4;
36            else if(leftNode=='T' && rightNode=='T') return 0;
37
38
39            return 0;
40
41        }
42
43
```

File: Tree.java

```
1  /*
2   * To change this template, choose Tools | Templates
3   * and open the template in the editor.
4   */
5  package tree;
6
8  import java.util.ArrayList;
8  import java.util.Scanner;
9  import java.io.*;
10
11 /**
12  *
13  * @author ManHunt
14  */
15
16
17 public class Tree {
18
19     /**
20      * @param args the command line arguments
21      */
22
23    //MEMORY CONSUMPTION
24    private static final long MEGABYTE = 1024L * 1024L;
25    public static long bytesToMegabytes(long bytes) {
26      return bytes / MEGABYTE;
27    }
28
29
```

```
30
31      static Scanner scan = new Scanner (System.in);
32      public static void main(String[] args) {
33
34          //Informative Site Counting
35           try{
36
37
38                  FileReader inputStream1 = null;
39                  inputStream1 = new FileReader("a.txt");
40
41                  FileReader inputStream2 = null;
42                  inputStream2 = new FileReader("b.txt");
43
44                  FileReader inputStream3 = null;
45                  inputStream3 = new FileReader("c.txt");
46
47                  FileReader inputStream4 = null;
48                  inputStream4 = new FileReader("d.txt");
49
50
51
52
```

```
53
54              char a,b,c,d;
55              int a1,b1,c1,d1;
56              int nam, sak =0;
57
58              StringBuilder seq = new StringBuilder();
59              String Sequence = "";
60
61                  while ((a1 = inputStream1.read()) != -1 && (b1 = inputStream2.read()) != -1
62                          nam=0;
63                          a = (char) a1;
64                          b = (char) b1;
65                          c = (char) c1;
66                          d = (char) d1;
67
68
69
70                          if(a!= b) nam++;
71                          if(b!= c) nam++;
72                          if(c!= d) nam++;
73
74
```

```
                            if(nam>sak) {
                                sak = nam;
                                Sequence = null;
                                seq = new StringBuilder();
                                seq.append(a);
                                seq.append(b);
                                seq.append(c);
                                seq.append(d);


                                Sequence = seq.toString();
                            }


                    }

                     //System.out.format("The value of i is: %d%n", sak);


                    System.out.println(Sequence);
```

```
            //TIME TRACING
            long startTime = System.currentTimeMillis();

        //MEMORY CONSUMPTION FOR COMMON INFORMATIVE SITE
        // Get the Java runtime
    Runtime runtime = Runtime.getRuntime();
    // Run the garbage collector
    runtime.gc();
    // Calculate the used memory
    long memory = runtime.totalMemory() - runtime.freeMemory();
    //System.out.println("Used memory is bytes: " + memory);
    //System.out.println("Used memory is megabytes: "
       // + bytesToMegabytes(memory));


        //ASSIGNING GIVEN SEQUENCE TO ALL POSSIBLE CATEGORIES
        String givenSequence = Sequence;
```

```
124
125         //getting highest nucleotide
126         int gCount = 0;
127         int cCount = 0;
128         int aCount = 0;
129         int tCount = 0;
130
131         for (int i=0; i < givenSequence.length(); i++){
132             if(givenSequence.charAt(i) == 'G')gCount++;
133             if(givenSequence.charAt(i) == 'C')cCount++;
134             if(givenSequence.charAt(i) == 'A')aCount++;
135             if(givenSequence.charAt(i) == 'T')tCount++;
136         }
137
138
139         int num;
140         int max=0;//define Maximum value and save it in variable max = 0;
141
142         for (int count=0 ; count<4 ; count++) // start loop with (for)
143         {
144             if( count == 0){
145             num = aCount;//user will enter number it will be repeated 5 times .
146             }
147             else if( count == 1){
148             num = cCount;//user will enter number it will be repeated 5 times .
149             }
150             else if( count == 2){
151             num = gCount;//user will enter number it will be repeated 5 times .
152             }
153             else{
154             num = tCount;//user will enter number it will be repeated 5 times .
155             }
```

```
156
157              if(num>max ){//Another condition to find the maximum number
158              max = num;//if so , num will be saved in (max)
159              }
160          }//End loop for determining number
161
162
163
164          char max1;
165          if(aCount == max ) max1 = 'A';
166          else if(cCount == max ) max1 = 'C';
167          else if(gCount == max ) max1 = 'G';
168          else max1 = 'T';
169
170
171
```

```
        String left = "";
        //String left= "AAAACCCCGGGGTTTT";
        StringBuilder sb = new StringBuilder();
        sb.append(max1);
        sb.append(max1);
        sb.append(max1);
        sb.append(max1);
        left = sb.toString();
        String right="ACGT";



        //ALL POSSIBLE PARENT NODE PAIRS
        ArrayList<ParentCategory> allCategories = new ArrayList<ParentCategory>();
        for(int i=0; i < left.length(); i++){
            SequenceValue myVal = new SequenceValue();
            ParentCategory p = new ParentCategory();

            p.setLeftNode(left.charAt(i));
            p.setRightNode(right.charAt(i));
            p.setCenterValue(myVal.getSequenceValue(left.charAt(i),right.charAt(i)));

            allCategories.add(p);

        }



    for(int i=0; i < givenSequence.length(); i++){
        for(int j=0; j < allCategories.size();j++){
            allCategories.get(j).addAndCalculate(givenSequence.charAt(i));
        }
    }


    //FINDING OUT THE MINIMUM VALUED CATEGORY
    int minVal = 99999;
    int minIndex = 0;
    for(int j=0; j < allCategories.size();j++){
        System.out.println("Left Node : " + allCategories.get(j).getLeftNode() + " Right Node
       System.out.println("Total Value : " + allCategories.get(j).getTotalValue());

            if( allCategories.get(j).getTotalValue() < minVal){
                minVal = allCategories.get(j).getTotalValue();
                minIndex = j;
            }


    }
```

```
223          //PRINTING DETAILS...
224          System.out.println("Minimum Category: ");
225          System.out.println("Left Node : " + allCategories.get(minIndex).getLeftNode() + " Right Node
226          System.out.println("Total Value : " + allCategories.get(minIndex).getTotalValue());
227          System.out.println("done");
228
229
230          //TIME CONSUMPTION
231          long stopTime = System.currentTimeMillis();
232          long elapsedTime = stopTime - startTime;
233          System.out.println(elapsedTime);
234
235
236          //MEMORY CONSUMPTION
237          // Get the Java runtime
238      Runtime runtime1 = Runtime.getRuntime();
239      // Run the garbage collector
240      runtime1.gc();
241      // Calculate the used memory
242      long memory1 = runtime1.totalMemory() - runtime1.freeMemory();
243      //System.out.println("Used memory is bytes: " + memory1);
244      //System.out.println("Used memory is megabytes: "
245          //+ bytesToMegabytes(memory1));
246
247      System.out.println("memory Usage: " + (memory1-memory));
248
249
250
```

```
251
252
253
254              }catch(Exception E){
255                 System.out.println("ERROR in ranking");
256              }
257
258
259          }
260      }
261
```

Bibliography

[1] D.A. Benson, M.S. Boguski, D.J. Lipman, J. Ostell, and B.F. Ouellette,"Genbank.", Nucleic Acids Research, 26(1):1–7, 1998.

[2] M. Nei, "Phylogenetic analysis in molecular evolutionary genetics", Annu. Rev. Genet, 30:371–403, 1996.

[3] D. L. Swofford, G. J. Olsen, P. J. Waddell and D. M. Hillis, "Phylogenetic inference", 407–514, Molecular Systematics 2nd edition. Sinauer, Sunderland, Mass, 1996.

[4] R. V. Eck and M. O. Dayhoff, "Atlas of protein sequence and structure", National Biomedical Research Foundation, Silver Springs, Md, 1966.

[5] W. M. Fitch, "Toward defining the course of evolution:minimum change for a specific tree topology", Syst. Zool, 20:406–416, 1971.

[6] W. M. Fitch. and E. Margoliash, "Construction of phylogenetic trees", Science 155: 279-284, 1967.

[7] J. Felsenstein, "Evolutionary trees from DNA sequences: a maximum likelihood approach", J. Mol. Evol, 17:368–376, 1981.

[8] A. Rzhetsky and M. Nei, "A simple method for estimating and testing minimum-evolution trees", Mol. Biol. Evol, 9:945–967, 1992.

[9] N. Saitou and M. Nei, "The neighbor-joining method: A new method for reconstructing phylogenetic trees", Mol. Biol. Evol, 4:406-425, 1987.

[10] M. Nei, "Molecular evolutionary genetics", Columbia University Press, New York, 1987.

[11] B. G. Hall, "Phylogenetic Trees Made Easy: A How-To Manual, 2nded", Sinauer Associates, Inc.: Sunderland, June 2004.

[12] P. Liò, N. Goldman, "Models of Molecular Evolution and Phylogeny", Genome Research, 8:1233-1244, 1998.

[13] W. J. Ewens, R. Grant, "Statistical Methods in Bioinformatics", Springer Science and Business Media: New York, 2005.

[14] C. R. Linder, T. Warnow, "An overview of phylogeny reconstruction", Handbook of Computational Molecular Biology, Chapman and Hall/CRC Computer & Information Science, 2005.

[15] W. H. Li, "Molecular Evolution", Sinauer Associates: Sunderland, MA, 1997.

[16] R. Durbin, S. Eddy, A. Krogh and G. Mitchison, "Biological Sequence Analysis", Cambridge University Press: Cambridge, 1998.

[17] K. Dowell, "Molecular Phylogenetics: An introduction to computational methods and tools for analyzing evolutionary relationships", Math 500, 2008.

[18] K. Takahashi and M. Nei, "Efficiencies of Fast Algorithms of Phylogenetic Inference under the Criteria of Maximum Parsimony, Minimum Evolution, and Maximum Likelihood When a Large Number of Sequences Are Used", Mol. Biol. Evol, 17(8):1251–1258, 2000.

[19] N. Saitou and T. Imanishi, "Relative Efficiencies of the Fitch-Margoliash, Maximum- Parsimony, Maximum-Likelihood, Minimum-Evolution, and Neighbor-joining Methods of Phylogenetic Tree Construction in Obtaining the Correct Tree", Mol. Biol. Evol, 6(5):514-525, 1989.

[20] Z. Ghosh and B. Mallik, "Bioinformatics: Principles and Applications", ISBN13: 9780195692303, Oxford University Press, 2008.

[21] B. Kolaczkowski and J. W. Thornton, "Performance of maximum parsimony and likelihood phylogenetics when evolution is heterogeneous", NATURE, Vol 431, October 2004.