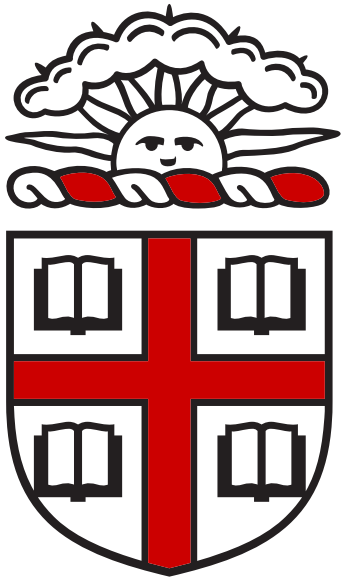
# Event-based Phylogeny Inference and Multiple Sequence Alignment



Phong Nguyen Duc
Computer Science Department
Brown University

Submitted in partial fulfillment of the requirements for the
Degree of Master of Science in the Department of Computer Science at Brown University

Providence, Rhode Island
May 2012

This thesis by Phong Nguyen Duc is accepted in its present form
by the Computer Science Department as satisfying
the thesis requirements for the degree of Master of Science

Date

Franco P. Preparata, Advisor

Approved by the Graduate Council

Date

Peter M. Weber, Dean of the Graduate School

# VITA

Phong Nguyen Duc was born in Haiphong city, Vietnam, on 17 August 1989. After completing his high school study at the High school for the Gifted (Hochiminh city) in 2007, he entered the National University of Singapore where he studied Computational Biology. In 2011, he entered the Graduate School at Brown University, Computer Science Department, under the concurrent degree agreement between Brown University and the National University of Singapore.

# Preface

Since the identification of DNA/RNA as genetic material, deciphering the code of life has been a major goal put forward by biologists. One approach particularly successful in studying DNA sequences is to compare related sequences from different organisms. Sequence alignment, specifically pairwise alignment, is among the earliest tool developed in bioinformatics. However, the generalization of pairwise alignment to multiple sequence alignment is not straightforward. The comparison of multiple sequences is expressed in two different but related problems: multiple sequence alignment finding shared homologous regions among input sequences, and phylogeny inference finding the order by which each sequence diverges from a common parent. These two problems have been under intensive research in the last three decades.

However, multiple sequence alignment and phylogeny inference are not completely solved problems, in the sense that there is no single best algorithm that stands out practically and theoretically for each of these problems.

My first encounter of the phylogeny inference problem was in 2010, when Prof. Ken Sung at the National University of Singapore gave us an assignment to infer the phylogeny of dengue viruses across the world. By then I noticed that not all regions in the sequences can be aligned reliably, due to heavy mutations and high degree of divergence. This problem is more serious with long input sequences.

Prof. Franco P. Preparata introduced the problem to me again in 2011, this time at Brown University. He was looking into how ancestor sequences can be constructed to help build the phylogeny. By the end of 2011, we had some idea of how to generate putative ancestor sequences for the internal nodes of the phylogeny, assuming there is no insertion/deletion.

In Spring 2012, I found a way to reliably identify insertion/deletion events. This is then used to extend our previous algorithm to handle insertion/deletion. The final algorithm is a novel tool that suggests a complete evolution hypothesis of input sequences, consisting of a phylogeny and of the placement of mutations on the edges of the resulting tree.

As described above, this thesis started with the initial insights from Prof. Franco. The discussions with him provided me with new insights, as well as support to my ideas. I can not thank him enough for these discussions, for the courses he recommended, and for his time proofreading and editing this thesis. He has been a great mentor to me.

Special thanks to previous teachers who nurtured my interest in genomics and bioinformatics: Prof. Ken Sung (NUS), Dr. José Dinneny (NUS), and Prof. Sorin Israil (Brown University).

This thesis would not have been possible without the financial support from the Singapore Government and SAS Institute, Singapore.

Last but not least, I would like to thank my beloved family and friends who have been a constant source of love and support. I am forever indebted to them.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

A persistent focus of biological research has been the detection of similarities and dissimilarities among living species. As the study of nature continues, human knowledge of similarities and dissimilarities among species has grown gradually both in depth and in breadth.

Our knowledge of similarities and dissimilarities is used to give names to species we see. The science of identifying, naming, and organizing species into groups, called *Taxonomy*, until 2010 has identified millions of species [Report, 2010], which demonstrates the breadth of our knowledge about species.

One may go further to ponder about the cause of the observed similarities and dissimilarities. Charles Darwin's seminal work "On the origin of species" and many other contributions have suggested that all living things share a universal ancestor, and the differences among species are partly caused by mutations accumulated over generations. *Phylogenetics* is the study of the evolutionary relatedness among species. Researchers have established the links among seemingly different life forms, from bacteria, fungi, to animals and plants [Maddison, 2007]. Deciphering such a distant past of species evolution requires a deep understanding of morphology, molecular

biology, and genomics of species.

Until the 19th century, much of the data available to taxonomy and phylogenetics referred to the geological distribution and morphology of wildlife and fossils. As a result, classical methodologies of taxonomy and phylogenetics have been developed to work on morphological features, e.g. birds are vertebrate animals with feathers and wings.

After the identification of DNA/RNA as the genetic material of living organisms, genome sequences become a new input to old sciences. Similar to the way morphological traits are compared in traditional phylogenetics, computational phylogenetics starts with comparing genomic sequences. Such comparison provides an unprecedented granularity in our ability to compare living organisms, such that the dissimilarity between parents and children can be detected and quantified. A notable example is the reconstruction of the history of human migration from Africa based on mitochondrial DNA, which contributes important evidence in addition to older evidence derived from archaeology and linguistics.

However, this new advantage is a serious scientific challenge. As single bases mutate at higher frequencies than morphological mutations are observed, traditional approaches in taxonomy and phylogenetics cannot be directly applied on genome sequences. For example, it is harder to find one or two bases at certain positions to define a group of species the way birds are classified by having feathers and wings. As the result, phylogeny inference algorithms become increasingly complicated. They make use of sophisticated mathematical models to combine the information obtained from the whole sequences to determine the phylogeny, in contrast to the traditional approach that only makes use of important macroscopic morphological features.

The gap between phylogeny inference algorithm and classical phylogenetics has practical implications:

1. Current phylogeny inference algorithms do not allow for independent verifications of phylogenies. Given two conflicting phylogenetic trees (phylogenies) obtained by two different algorithms, we cannot tell which part of which phylogeny is more biologically plausible. This is why current algorithms return different phylogenies for different portion of the genome, and cannot combine the phylogenies nor explain why different phylogenies can be arrived at.

2. The way many algorithms infer phylogenies is also disconnected from the intuition of an evolutionary process. We have no idea of where a mutation happens in the phylogeny (the phylogenetic tree). While selecting morphological features to study has been a common practice in classical taxonomy and phylogenetics, most current multiple sequence alignment do not take into account the specific volatility of different regions. They try to impose an alignment even in sequence regions with higher mutation rate and unclear alignments. A phylogeny inference algorithm that takes in such an alignment would have to proceed with that unreliable imposition.

By addressing the aforementioned problems, this thesis works toward a more reliable approach to study the evolution of sequences.

## 1.1 Objectives

Our goal is to develop a unifying algorithm that takes as its input genomic sequences that are assumed to share a common ancestor, and output a hypothesis of their evolution, consisting of a phylogeny and of the placement of mutation on the edges of the resulting tree. While there are different types of mutations such as reversals and duplications, this thesis will focus on point substitution, insertions and deletions - mutations relevant to input sequences of several hundred base pairs in length. This

is different from many current algorithms that only give either the phylogeny from a set sequences, or the multiple alignment among sequences, but not an analysis of the actual mutations to explain the dissimilarity among sequences originated from a common ancestor. An algorithm that satisfies our goal would offer various benefits:

- The output is open to user validation, so that phylogenies from different algorithms and different genomic regions can be compared and combined.
- The algorithm generates putative ancestor sequences for the internal nodes of the phylogeny. This may be useful to evolutionary studies.
- The algorithm depends on more realistic assumptions, allowing more biological knowledge to be incorporated.

To achieve this goal, we use the following approach. We first use simulated data and aligned data to generate aligned input sequences with no insertion/deletion (indel). Simulated data with no indel is generated with a simplistic model of evolution: we start with a single sequence which represents the common ancestor; at each generation the available sequences will be duplicated with random substitutions to generate their offspring, similar to binary fission in bacteria. Aligned data is given in matrix form, where each row of the matrix corresponds to a sequence with gaps inserted in between. Gap-free sequences are generated from the matrix form by taking a sample of rows, and by removing columns with gaps (chapter 3).

By generating input sequences with no insertion/deletion, we can study phylogeny inference independently from multiple sequence alignment.

We first develop algorithms that suggest sequences at the internal nodes of the phylogeny (Section 5.4). The common parent of a pair of sequences is given as the consensus sequence, with possible ambiguity at positions where two children differ.

The ambiguity is later resolved using the principle of parsimony. Once all the ambiguities are resolved, each internal node of the inferred phylogeny is substantiated with a sequence. This is different from the usual approach of Neighbor-Joining that makes use of pseudo-distances. Our new approach is capable of locating point substitutions in the phylogeny.

We then develop an algorithm that detects gaps that represent insertions/deletions while doing multiple sequence alignment (Section 4.5). We looked for available multiple sequence alignment algorithms that suit our needs, but none was found. We transformed the problem of tracking gaps into the problem of tracking gap-free local alignments surrounding gaps. With this approach, we can reliably detect gaps resulting from the same insertion/deletion event. Existing algorithms have difficulties detecting insertion/deletion events because they represent gaps as single characters, in contrast to our representation of gaps as a whole.

Finally, we construct the final algorithm that infers the phylogeny, while simultaneously keeping track of point substitutions, insertions, and deletions (chapter 6). This algorithm makes use of the developed technique to detect insertion/deletion events. The maximum parsimony approach developed earlier can then be applied to find a plausible evolutionary hypothesis that takes the number of insertion/deletion events into account.

## 1.2 Organization

Section 1.3 sets up the terminology and notations used in the thesis. Chapter 2 discusses how different phylogenies and alignments are currently compared, and the assumptions upon which those comparisons are based. Scoring scheme and algorithms affect each other: failure to develop an algorithm that uses more realistic assumptions

would prevent strict scoring models to be used, while over-simplistic scoring models would lead to algorithms that optimize the wrong objective. In the problem of multiple sequence alignment and phylogeny inference, the currently used scoring models have problem keeping track of individual insertion/deletion events, even though such model exists for pairwise alignments (affine gap penalty, for example).

Chapter 3 describes how we obtain the data for our study with various useful statistics (for 16S RNA dataset) and default parameters (for simulated dataset).

Chapter 4 starts with a survey of current approaches to multiple sequence alignments, including progressive alignment and consistency approaches. The chapter concludes with our novel algorithm to detect insertion/deletion events, which borrows ideas from the consistency approach.

Chapter 5 starts with a survey of available phylogeny inference method, including maximum parsimony, maximum likelihood, and clustering methods. It then develops variants of Neighbor-Joining algorithm that construct putative sequences at internal nodes of the phylogenies. These novel algorithms borrow ideas from the maximum parsimony approach, Neighbor-Joining algorithm, and progressive alignment. The chapter concludes by comparing the accuracy of developed phylogeny inference algorithms using modifications of the Robinson-Foulds distance.

Chapter 6 introduces our main contribution, an algorithm that reconstructs the whole evolution process from input sequences. This chapter draws a lot of concepts and algorithmic ingredients from previous chapters. In details, it extends a Neighbor-Joining variant from chapter 5 to keep track of insertion/deletion events by operating on the output of the insertion/deletion detection algorithm from Chapter 4. The result algorithm completes the framework by solving the problems pointed out in Chapter 2.

Chapter 7 finally summarizes the conclusion and suggests interesting directions for further study.

## 1.3 Definitions

In this Section we introduce mathematical formulations of biological concepts. While these are all commonly used concepts, different assumptions with varying strength are still needed to justify the mathematical formulations.

*Sequences* are sequences of letters A, G, T or C. This is an abstraction of DNA sequences. We are interested in sequences of several hundreds bases.

Two sequences are called *homologous* if they have a shared ancestor sequence. We are interested in homologous pairs with similar function, hence undergoing the same evolutionary stress. In higher organisms such as animals and plants, these homologous pairs originate from one sequence that went on to evolve independently after *speciation* in two reproductively isolated species (*orthologs*).

Two sequences may have regions that are homologous to each other. A *pairwise alignment* arranges the regions to match them base by base. Given two sequences $S_1$, $S_2$ with some homologous regions, a *pairwise alignment* of $(S_1, S_2)$ is a two rows matrix with entries of either A, G, T, C or single gaps, such that if gaps are removed, the first row is the same as $S_1$ and the second row is the same as $S_2$. We want the aligned positions of $S_1$ and $S_2$ to be homologous of each other. Note that this construction cannot reveal mutations such as reversals, duplications, translocations...

The common practice to detect homology is to search for similar sequences using sequence alignment algorithms, sometimes with biological function validation. Since the history of most sequences is unknown, we would fail to detect homology if too

many mutations have happened between the two sequences.

A *species* is a group of individuals that can interbreed, and reproductively isolated from other such groups. We assume that the set of ancestors of a species can be listed as one chain, e.g. there is an inheritance relationship between any two ancestors of a given species. The assumption fails in occasions when individuals from different species can still interbreed. This situation is more common in bacteria and viruses.

While the genomes of individuals in a species contain differences, we take the sequence of a species to be a consensus sequence over those variants. This assumption suffices for most phylogeny analysis, as the intra-species variation is negligible when we are working on inter-species variation. Such an assumption would need to be relaxed if we want to model the continuous change of allele frequencies in the course of evolution (affected by natural selection, genetic drift and gene flow).

Suppose we know that a set of sequences $\mathcal{S}$ are homologous and want to obtain their evolution history. We denote their latest common ancestor $R(S)$, or $R$ if the reference to

$$\mathcal{S}$$

is implicit, to be the latest sequence that each sequence $S_i$ in $\mathcal{S}$ can trace back to. For each sequence $S_i$ there is a single chain of ancestors that trace back to $R(S)$ (assumption above), where each node represents an ancestor species. The union of all those chains is a tree $T_0(S)$. For simplicity and practical reasons, we contract edges when there are internal vertices with a single child. The tree $t(\mathcal{S})$ obtained after contraction is called the *phylogeny* relating $\mathcal{S}$. Chromosomal crossover and other types of genetic recombination cannot be described by a phylogenetic tree.

In genetics, *mutation rate* is the probability that mutation occurs in a cell division. The concept also works in the context of a single gene or a single base. *Mutation*

*frequency* is the generalization of mutation rate in a time unit. While mutation frequency is easier to measure, there are more determining factors that come into the picture, for example selection forces.

There are different kinds of mutations. Nucleotide substitution is the most frequent mutation, as well as the easiest one to detect and quantify. *Substitution rate* and *substitution frequency* are defined likewise.

Once we start to quantify the relationship between sequences, different metrics come into the picture. Suppose we are comparing two sequences $S_x$ and $S_y$. The *edit distance* is the smallest number of mutational events (insert/delete/substitution) that converts $S_x$ into $S_y$ (or vice versa). If we assume no insertions and deletions, the edit distance becomes the *Hamming distance*. There are also other distance metrics that look at insertions, deletions, GC contents...

With a chosen metric, we can then add weights to edges in $T_0(S)$ and $t(\mathcal{S})$: for an edge $e$, $w(e)$ is the distance between the sequences at its two end points. Ideally, the distance between the same pairs of nodes in $T_0(S)$ and $t(\mathcal{S})$ should not differ too much.

Because every mutation has a mutation that reverses it, the *root* $R$ can be placed anywhere in $t(\mathcal{S})$ unless we have some reference to time. In particular, if the mutation frequency is similar among all sequences in the course of evolution, the edit distances between $R$ and each of the leaves would be roughly the same, reflecting comparable evolution time from $R$. This property helps us guess the position of $R$ in an unrooted tree.

Note that this constant molecular clock hypothesis rarely holds due to different factors: life span variation, function changes, environment variation... Therefore, the position of $R$ in $t(\mathcal{S})$ is often undetermined, and can only be resolved with the

existence of an *outgroup* - a sequence not belonging to $\mathcal{S}$ but still sharing a traceable common ancestor. Such a sequence would be connected to $t(\mathcal{S})$ at $R$.

Given $\mathcal{S}$, we want to reconstruct a tree $t(\mathcal{S})$ that approximates $t(\mathcal{S})$. The main aim is to maximize topological similarity. This problem is called *phylogeny inference*. Depending on the application of $t(\mathcal{S})$, we may be required to obtain more information that accompanies the phylogeny. For example, evolutionary studies may involve estimating edge weights, root node, and/or sequences filled in the internal nodes, in addition to the tree topology.

Some phylogeny inference method for $n$ species refers to a *distance matrix* $d_{n\times n}$ where $d_{x,y}$ is a distance between $S_x$ and $S_y$ in a chosen metric. We would also want to apply the same metric used in constructing $d$ to weight the edges in $t(\mathcal{S})$. Intuitively, the divergence between $S_x$ and $S_y$ can be seen as the accumulation of multiple intermediate steps. Mathematically, if $t(\mathcal{S})$ contains a path $(S_x, u_1, u_2, ..., u_{n-1}, S_y)$, it is ideal to have

$$d_{x,y} = w(S_x, u_1) + w(u_1, u_2) + ... + w(u_{n-1}, S_y) \tag{1.1}$$

where $w$'s are the weights of edges in $t(\mathcal{S})$. If a metric satisfies 1.1 it is called *tree additive*. Note that the absolute equality rarely happens in biological data, so we usually accept small differences when we call a metric *'tree additive'*. Under tree additive metrics, tree distances are also preserved between $T_0(S)$ and $t(\mathcal{S})$.

When the substitution rate is high or the branches are long, some mutation is reversed. A letter A is mutated to G, and then mutated back to A again. This is called *homoplasy*. In general, a series of point mutation happening in the same position would appear as a single mutation. With high degree of *homoplasy*, or generally overlapping, the Hamming distance increasingly deviates from being tree

additive.

*Multiple sequence alignment (MSA)* is a straightforward generalization of pairwise alignment. Assuming no change in the order of homologous regions, we want to add gaps so that those regions align base by base. However, the objective function used for pairwise alignments cannot be generalized easily, and it is NP-hard to optimize for most objective functions.

MSA is useful for phylogeny inference because it helps detect homologous regions in a set of sequences. On the other hand, many MSA algorithms need a guiding tree to define their objective function or reduce the problem to pairwise alignments. The knowledge of sequence history would require both MSA and phylogeny inference to be solved.

# Chapter 2

# Scoring model

The relationship between sequences is a description of how regions of different sequences evolved from some common ancestor. Suppose two different algorithms give two different results, the user then needs a method to pick out the better result. Qualitatively, the result that provides a clearer picture on how mutational events happen would be the better result. Quantitatively, we need a scoring model to compare results.

We first look into scoring models with two sequences, and then move on to scoring models of multiple sequence alignment.

## 2.1 Scoring of Pairwise Alignment

### 2.1.1 Hamming distance

We first motivate with a simple scoring model, *Hamming distance*. Given two sequences $S_1$, $S_2$, where $S_1[i]$ is the $i$-th character of sequence $S_1$ and so on, their

Hamming distance is the number of positions where they differ.

$$d(s_1, s_2) = |\{i : s_1[i] \neq s_2[i]\}|$$

This model does not account for insertions/deletions, as well as higher order mutations such as duplications, reversals... However, the model accounts for substitution, beside the caveat that multiple substitutions occurring at the same position cannot be detected.

As different nucleotides have different chance of being mutated into another nucleotide, different scores can be assigned to different matches/mismatches. Time may also be added as a parameter in the model. Models generalized in this direction has been studied extensively [Jukes, 1969], and are incorporated into several phylogeny inference algorithms, despite the fact that insertions/deletions are neglected.

### 2.1.2 Levenshtein distance

Without modelling insertions/deletions, it is impossible to explain the evolution between two sequences with different lengths. *Levenshtein distance*, often called *edit distance*, is the minimum number of edits needed to transform one sequence to another, with only insertion/deletion, and substitution of single characters taken into account. For example, the distance between "abcde" and "bbce" is 2 (subtitute "a" to "b", and delete "d").

By assigning different cost for different edit actions (insertion and deletion are less frequent events, so they are assigned higher cost), this model becomes useful enough that it has been incorporated into Smith-Waterman algorithm [Smith and Waterman, 1981], which is still commonly used.

Both Hamming distance and Levenshtein distance were originally defined in the context of pairwise distances. When we have multiple sequences, they become the basic building blocks of the new model.

### 2.1.3 General gap penalty

Levenshtein distance cannot model insertions/deletions of multiple characters. They approximate this cost by the number of characters being deleted/inserted. In the real situation, a gap of length 10 is much more likely than 10 separate single gaps. This suggests that we find a better model for insertions/deletions. One way to do this is to assign a penalty score for each single insertion/deletion event that is detected.

The penalty score should reflect the distribution of insertion/deletion length. An ideal model would even take the local information into account: whether it is in the loop region of a protein/RNA sequences, its exon encoding frame, etc. However, it is hard to model all these factors in a general scoring model.

To build a gap penalty that works reasonably with different kinds of sequences, we have to rely on some general observation. The gap penalty should be monotonic: a short gap happens more often than a longer gap. The gap penalty is also conveniently modelled as being convex: the penalty per base of a long gap is smaller than that of a short gap. The event of an insertion/deletion itself is more important than the length of the insertion/deletion. With all these observations in mind, *affine gap penalty* tries to approximate a reasonable general gap penalty by assigning a penalty for opening a gap, and a smaller penalty for every character a gap extends. This is an algorithmically convenient model, and has found its way into the most popular search algorithm, BLAST.

## 2.2 Multiple Sequence Alignment

### 2.2.1 Star graph approximation and sum-of-pairs

Suppose the underlying phylogeny is a star graph (reference to figure). The total weight of all edges is then proportional to the sum of all pairwise distances among leaves. This sum is a commonly used score over multiple sequence alignments, called *sum-of-pairs* score [Lipman et al., 1989].

Since most phylogenies are not star graphs, the sum-of-pair is not a good scoring model. It introduces biases if a large portion of input sequences cluster together. Some realized this problem, but they introduced ad-hoc fix instead of changing the scoring model itself [Thompson et al., 1994].

Other scoring models such as maximum parsimony and maximum likelihood takes the phylogeny into account when computing the score. However, finding the phylogeny that maximizes such a score is NP-complete even for Hamming distance [Felsenstein, 2003], not to mention more complicated models.

### 2.2.2 Affine gap in multiple sequence alignment

If affine gap penalty can be applied in multiple sequence alignment, it would offer the same benefit that made it useful for pairwise alignment: insertions and deletions can be detected as events rather than artificial gap characters. However, few multiple alignment algorithms use affine gap penalty, because it is harder to generalize the concept. Most alignment algorithms assume some independence between how subsequent characters from different sequences match, so that the algorithm can rely on dynamic programming to reduce the space of alignments to be checked. The alignment of two affine gaps cannot be fit into this framework.

For one to write algorithms that operate on insertions/deletions as single events, one has to obtain a holistic view of sequence regions and the gaps in between. Such algorithms will have to deal with major as well as minor insertions/deletions; and the pathological cases where those events overlap. Clearly, this is much more complicated than the conventional approach that takes in one character at a time, but it is also more informative.

# Chapter 3

# Datasets

We use two datasets to independently evaluate computational methods. The first dataset is a collection of aligned 16S rRNA with phylogeny estimated by the All-species Living Tree project (LTP) [Munoz et al., 2011]. The dataset consists of more than 8000 SSU (small subunit) ribosomal sequences about 1500bp each. The sequences are aligned and organized into a phylogeny. To obtain the ancestor sequences, we run Fitch's algorithm [Fitch, 1971] to find a maximum parsimony solution. A more involved approach would be to maximize the likelihood with branch lengths taken into account.

The second dataset is generated by simulation. Our simulation takes in 6 parameters: $n$, the approximated length of all the sequences; $maxp$, the maximum substitution rate in each site of the sequence; $pIns$, the probability of insertion/deletion in each generation; $insertSize$, the maximum size of each insertion/deletion; $nSeq$, the approximated number of leaves in the generated phylogeny; and $pSurvive$, the probability that a leaf is chosen from a full binary tree as described below.

1. Generate the common ancestor $R$ as a sequence of $n$ i.i.d. character, each

drawn uniformly from $\{A, C, T, G\}$, and is assigned a substitution probability uniformly drawn from the range $[0, maxp]$.

2. For each leaf, mutate it twice to generate two new children connected to it. The mutation process is described below.

    - For each position, mutate it according to the assigned substitution probability. Given that a substitution event happens, the new base is chosen uniformly with probability 1/3.
    - With probability $pIns$, the mutation includes a single insertion or a single deletion (each with conditional probability of 1/2). The position of the indel is picked uniformly across the whole sequence, and the length of the insertion is also picked uniformly from $[1, insertSize]$. If new bases are introduced to the sequence, they also have their substitution probability assigned as described above.

3. Repat (2) until $pSurvive$ times the number of current leaves is greater than $nSeq$.

4. Each leaf of the current full binary tree is chosen for the final phylogeny with probability $pSurvive$. The returned phylogeny is the current full binary tree restricted on the survived leaves.

With this simulation scheme, we can keep track of the true phylogeny relating the observed leaves, as well as the whole mutation process. Each internal node of the model phylogeny is an ancestor sequence obtained during the simulation. We also know the alignment of sequences, since the history of each single base is kept.

This simulation protocol implicitly assumes constant molecular clock. When we want to remove that assumption, instead of generating two branches for each leaf, we may only take one random leaf and expand each iteration.

We can reduce the size of the datasets. For the LTP dataset, we can pick a random subtree of a roughly fixed size. For the simulation dataset, we can vary the number of iterations to adjust the number of selected leaves. During the first stage of studies, a small input size is crucial to the development speed.

We can also create a dataset that only have point substitution as mutation. This is done with simulated data by fixing $pIns = 0$. For real biological data, since the sequences are aligned, we can remove the columns with gaps. The remaining sequences would be gap-free, and we take them to be aligned without having to make any deletion or insertion. By ignoring sequence alignment, we can study phylogeny inference independently from multiple sequence alignment.

Unless otherwise noted, we pick $maxp = 0.1$, $pSurvive = 0.5$, $insertSize = 3$. If the study assumes no gap, $pIns = 0$. Otherwise $pIns = 0.03$. $n$ and $nSeq$ are two key parameters, and vary during our study. A realistic default would be $nSeq = 50$, $n = 200$.

# Chapter 4

# Multiple Sequence Alignment approaches

Given $m$ sequences $S_1$, $S_2$,... $S_m$ with some homologous regions, a *multiple alignment* of $(S_1, S_2, ...S_m)$ is a $m$-row matrix with entries of either A, G, T, C or single gaps, such that if gaps are removed, the first row is the same as $S_1$ and the second row is the same as $S_2$, and so on. A good multiple alignment suggests that positions aligned in the same column are homologous to each other.

Multiple sequence alignment methods do not separate into disjoint classes. Each method is built on previous methods, coupled with a few observations and improvements. The following Chapter describes some common themes and how they evolved through time.

| . | A | C | G | T | gap |
|---|---|---|---|---|---|
| A | 1 | -1 | -1 | -1 | -3 |
| C | -1 | 1 | -1 | -1 | -3 |
| G | -1 | -1 | 1 | -1 | -3 |
| T | -1 | -1 | -1 | 1 | -3 |
| gap | -3 | -3 | -3 | -3 | $-\infty$ |

Table 4.1: Similarity matrix with each mismatch penalty -1, gap penalty -3 and each match score 1

## 4.1 Dynamic Programming Approach

When the number of sequences is 2, the commonly used algorithm for pairwise alignment is the Needleman-Wunsch algorithm or its variants. It defines an objective function as follows.

Given a pairwise alignment as a 2-row matrix, each column can be scored according to a similarity matrix. The similarity between two identical bases should be higher than the similarity between two different bases (usually negative, perceived as a penalty). As there are 4 possible bases in addition to the single-position gap, the similarity matrix is a $5 \times 5$ matrix (Table 4.1).

The score of the whole alignment is the sum of individual scores of each column.

This objective function assumes columns can be scored independently. Therefore a Dynamic Programming algorithm can be used to find an alignment that optimizes the objective function. The Needleman-Wunsch algorithm follows a dynamic programming framework that defines a state $d_{ij}$ to be the best score that can be obtained by aligning substrings $S_1[0, i]$ to $S_2[0, j]$ (Table 4.2).

For Needleman-Wunsch algorithm to extend to the alignment of $m$ sequences, we need to somehow generalize the scoring of a column of 2 bases into the scoring of a column of $m$ bases. *Sum-of-pairs* is one such generalization, defined as follows.

| . | A | C | C | T | G |
|---|---|---|---|---|---|
| A | 1 | -2 | -5 | -8 | -11 |
| C | -2 | 2 | -1 | -4 | -7 |
| T | -5 | -1 | 1 | 0 | -3 |
| G | -6 | -4 | -2 | 0 | 1 |

Table 4.2: Dynamic Programming table for aligning ACCTG to ACTG, using similarity matrix in Table 4.1. The corresponding alignment is (ACCTG/AC-TG)

The *sum-of-pairs* score (*SP score*) of a column is the sum of scores from each pair of characters in the column.

The generalization of the algorithm then follows naturally [Lipman et al., 1989]. However, the naive algorithm is impractical because the number of dynamic programming (DP) states is now exponential with respect to $m$ (Figure 4.1 for $m = 3$).

$$dimension(d) = (|S_1|, |S_2|, ..., |S_m|)$$

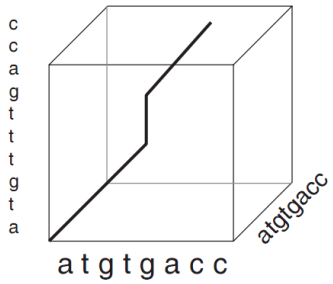



Figure 4.1: Alignment path for 3 sequences [Lee et al., 2002]

Initially, most effort was spent on decreasing the number of DP states to be computed to improve the speed of the algorithm [Lipman et al., 1989]. However, another drawback lies in the sum-of-pair scoring function. How gaps are introduced into a sequence depends on all other sequences equally, while it should have given more weights to similar sequences than distant sequences [Feng and Doolittle, 1987]. Feng and Doolittle proposed that similar sequences should be aligned first, then more

distant sequences are incorporated into the alignment. This idea gives rise to the main body of practical algorithms for multiple sequence alignment, *progressive alignment*.

## 4.2 Progressive Approach

*Progressive alignment* is an approach to construct the multiple alignment from a series of pairwise alignment steps, each tries to align the results of previous alignment steps.

For example, we want to construct the multiple alignment of three sequences CAAAGGGT, CAAAT, and CGGGT.

First, we align CAAAGGGT with CAAAT to get

```
CAAAGGGT
CAAA---T
```

Then, we align CGGGT with the previous result to get

```
CAAAGGGT
CAAA---T
C---GGGT
```

Note how gaps are introduced to sequences so that they align with each other. Since only gaps are introduced in each alignment step, the approach is labeled *"once a gap, always a gap"*.

We now make two observations that suggest how we should formalize the previous process.

First, the order of alignment steps matters. For example, if we align CAAAT and CGGGT first, we will have a different alignment:

```
CAAAT
CGGGT
```

This alignment would then be aligned with CAAAGGGT.

```
CAAA---T
CGGG---T
CAAAGGGT
```

The multiple alignment obtained in this order is less (biologically) plausible than the previous multiple alignment, since "GGG" had a perfect match that it is not aligned to.

Since each step combines two previous results, the alignment order corresponds to a binary tree, with initial sequences at its leaves. As the tree is used to guide the pairwise alignment steps, it is labeled a *guide tree*. (figure references for the example here)

Second, our pairwise alignment should be able to take in the output of previous alignment steps. For example, to be able to align (CAAA—T/CAAAGGGT) with CGGGT. Clearly the inputs are not DNA sequences anymore. They are more complicated to be able to describe the alignments that have been made in previous steps. We call these structures *profiles*, defined as follows.

Given a set of sequences $\mathcal{S}$, a profile of $\mathcal{S}$ is a structure that summarizes the multiple alignment of sequences in $\mathcal{S}$. The representation of the profile is designed to support its use in multiple alignment:

- A profile can be generated from an individual sequence. Here we call profiles generated from individual sequence *singleton profiles*.

- Any two profiles $A$ and $B$ can be aligned to yield another profile, such that the new profile keeps track of which part of $A$ is aligned with which part of $B$, and which part of $A$ and $B$ cannot be aligned.

Different profile representations are described in Section 4.2.1.

Having defined *guide trees* and *profiles*, we can formalize the progressive alignment approach as follows.

Input: a set of sequences $\mathcal{S}$.

1. Calculate the guide tree from $\mathcal{S}$
2. Replace each sequence $S_i$ in $\mathcal{S}$ by its singleton profile $P_i$
3. While there are more than one profile in $\mathcal{S}$:
   (a) Select two profiles $P_x$,$P_y$ from $\mathcal{S}$ according to the computed guide tree
   (b) Align $P_x$ and $P_y$ to obtain $P_z$
   (c) Remove $P_x$ and $P_y$ and add $P_z$ to $\mathcal{S}$

Now we have formalized what the progressive alignment approach is, we can go back and address the two observations we made before.

The first observation was about the importance of guide trees. Similar sequences can be aligned with confidence, while distant sequences cannot. The relationship between sequences is captured in phylogenetic trees, therefore it is natural that phylogenetic inference algorithms such as Neighbor Joining [Thompson et al., 1994] and UPGMA [Edgar, 2004] be used to produce guide trees.

The second observation was about the role of profiles. We will discuss different representations of profiles in Section 4.2.1.

However, the way profiles are use also poses additional problems. Because each alignment only uses the information from two profiles, it ignores the information from other sequences. How do we utilize other sequences at the same time to prevent mistakes in initial alignments? (Section 4.3). Suppose we have made mistakes during the first few alignment steps, and they are propagated to later steps, how do we fix those mistakes? (Section 4.4).

### 4.2.1 Profile representation

A profile should summarize the information of sequences in its subtree, and allow for alignment with another profile.

The simplest and most commonly used representation of a profile is the *fractional count*, albeit often only briefly mentioned as averaging over the whole column [Notredame et al., 2000] [Do et al., 2005].

As sequences under one single profile have been aligned, we can write them down as an $m$-row matrix, where $m$ is the number of sequences. Let the number of columns be $N$, then the profile $P$ is then a sequence of length $N$, with the $i$-th element $P_i$ keeping track of the A-C-G-T content in column $i$.

$$P_{i,c} = \frac{count_{i,c}}{m}, c \in \{A, C, T, G, gap\}$$

$count_{i,c}$ is the number of occurrences of $c$ in column $i$. Note that

$$\sum_c count_{i,c} = m \; \forall i = 1, ..., N$$

When two columns $A_i$ and $B_j$ of two profiles $A$ and $B$ are aligned, the score is

given as the weighted average of base-base similarity score $\delta$.

$$score = \sum_{c_1,c_2} A_{i,c_1} * B_{j,c_2} * \delta(c_1, c_2)$$

For example, if we use the similarity matrix from Table 4.1 as $\delta$, and the columns to be aligned are ...

```
Ai = A        Bj = A
     A             A
     A             G
     G             G
```

... then the similarity of these two columns is

$$A_{i,A}B_{j,A}\delta(A,A) + A_{i,A}B_{j,G}\delta(A,G) + A_{i,G}B_{j,A}\delta(G,A) + A_{i,G}B_{j,G}\delta(G,G)$$
$$= \frac{3}{4}.\frac{2}{4}.1 + \frac{3}{4}.\frac{2}{4}.(-1) + \frac{1}{4}.\frac{2}{4}.(-1) + \frac{1}{4}.\frac{2}{4}.1 = 0$$

While this representation is simple, how do we know if the alignments it gives are biologically plausible?

We can use Occam's razor as a criterion to guide our alignment selection. A biologically plausible hypothesis is one that requires fewest assumptions to explain the observed sequences.

Each multiple alignment is a hypothesis: it hypothesize that some positions are homologous to each other, while others are not. The gaps introduced and the mismatches are assumptions: we assume that those are the real mutations to explain how a common ancestor evolved into observed sequences.

The number of assumptions (or likelihood) can be measured if all ancestor se-

quences are known. However, it is more involved to infer those ancestor sequences, and the *fractional count* profile representation is a reasonable approximation. At a position $i$, the ancestor sequence is fixed if all characters in the corresponding column are the same, that is, $\exists c : P_{i,c} = 1$. Otherwise, our uncertainty scales with the number of other characters that we observed.

However, this approximation does not come without caveats.

One problem is that a biased sampling of sequences would lead to biased column representation. For example, if instead of an alignment of 10 sequences, we have another 10000 extra copies of one sequence to have 10010 sequences in total, then any pair of profiles would be very similar to each other, biasing any similarity scoring. The actual situation is not so extreme, but the way people collect DNA sequences from species do introduce some biases into the databases. One way to reduce the effect of duplicated information is to give each sequence a different weight [Thompson et al., 1994]. Similar sequences would be down-weighted, because they are over-represented in the sampling pool.

Another problem is that *fractional count* tends to penalize insertion more than deletion. An insertion introduces an extra column with the same penalty calculated over and over again, while a deletion is just a gap in an existing column. To overcome this problem, one can keep track of existing gaps, and avoid penalizing them again [Löytynoja and Goldman, 2005].

Representing a profile as a sequence also poses another problem, demonstrated by the following example.

Consider 5 domains S, T, X, Y, Z and the following 3 sequences: XYT, XZT, and XST. If the profile for the first 2 sequences is (XY-T/X-ZT), S would be aligned to YZ. The situation would be completely different if by chance we produced a different

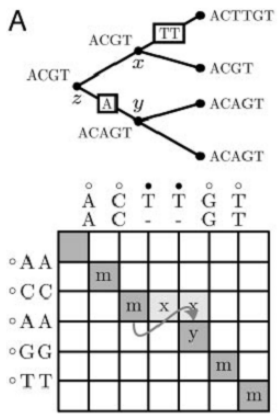



Figure 4.2: The insertion "TT" is counted twice when profiles $x$ and $y$ are compared. It introduces two additional columns when compared with a similar deletion of size 2. The algorithm uses the arrow to skip the gaps that have already been penalized [Löytynoja and Goldman, 2005]

profile (X-YT/XZ-T). Then S would be aligned to ZY. The difference here is merely an artifact of the forced order of unaligned domains.

In general, when there are two domains that have never appeared in the same sequence, a greedy algorithm will have to impose an order on two unrelated domains in the multiple sequence alignment, with no reason why one order is preferred over another.

The Partial Order Graph (POA) algorithm [Lee et al., 2002] seeks to remedy this problem by representing a profile as a Directed Acyclic Graph. The alignment of XYT and XZT would then produce the following DAG.

Using Directed Acyclic Graph as a profile representation adds some complexity. The authors could not align two profiles, so they incorporated sequences into a growing profile, one by one. This in turn makes the algorithm sensitive to the order of incorporated sequences. Another difficulty is to detect domains in a sequence. The authors chose to incorporate only the best local alignment into the growing profile, ignoring other domains disjoint from that local alignment.

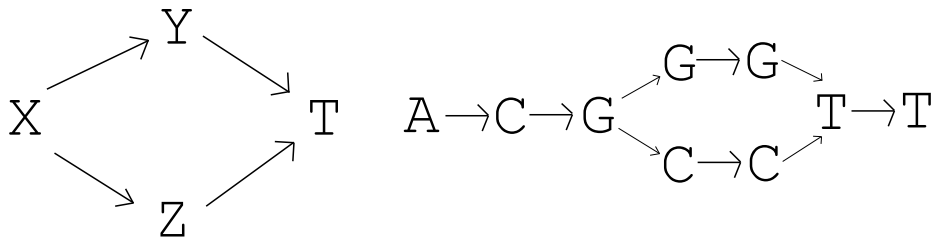



Figure 4.3: DAG resulted from aligning XYT and XZT. The actual graph is on the right, as we transform each domain into its corresponding sequence.

However, this approach reveals some interesting ideas. First, the *fractional count* representation is not the only possible way, and other alternatives are worth exploring. Second, when many sequences are aligned (up to thousands of sequences), distant pairs of sequences appear, and in many cases their differences cannot be explained by substitution and short indels. For pairwise alignments there are global and local alignments, so similarly it might be interesting to examine the idea of local alignment in the multiple sequence setting.

## 4.3 Consistency Approach

Sequence alignment can be seen as a signal detection problem: we need more than one signal to obtain information from data with confidence. Given two sequences $a$ and $b$, if $a_i = b_j$, $a_{i+1} = b_{j+1}$, ... $a_{i+l-1} = b_{j+l-1}$ with large enough $l$, then we are more confident to say that $a[i, i+l-1]$ matches $b[j, j+l-1]$. The fact that the indices of the matches are consecutive makes it possible to combine the signals and report the match confidently. If we look at an $m \times N$ alignment matrix, then this combination of signals is a string of columns in the alignment matrix. Is there another way to combine signals in the alignment matrix?

One of the advantages that multiple alignment has over pairwise alignment is that we have more support for the alignment: if substring X is aligned to Y, and

Y to Z, then this supports that X aligns to Z. We call this combination of signals *consistency*. Consistency has been a very important tool to incorporate information from all sequences, even in pairwise alignment steps of progressive alignment.

DALIGN is among the first multiple sequence aligner to implement consistency [Morgenstern et al., 1998]. Given $m$ sequences, they perform all $\frac{m(m-1)}{2}$ possible pairwise alignments. For each pairwise alignment between sequences $a$ and $b$, a pair $(i, j)$ such that $a_i$ is aligned with $b_j$ is called a *diagonal* (which is different from the conventional diagonals in alignment matrices). All those diagonals are collected, sorted according to their own weights and how much they overlap with other diagonals, and then added to the multiple alignment one by one.

T-Coffee is a widely used aligner that follows a similar approach [Notredame et al., 2000]. It generates a library of alignments consisting of pairwise global and local alignments from input sequences. Each alignment is assigned a score, which is the fraction of matches over the length of the alignment. This is also called the *identity* of the alignment. Each pair of aligned bases is then assigned an initial weight: the identity of the alignment those aligned bases come from. Suppose $A$, $B$, $C$ are the different positions in three different sequences, and $W(A, B)$, $W(A, C)$, $W(B, C)$ are the assigned weights to the aligned pairs. We then iteratively update the weight according to how other sequences confirm the alignment: $W'(A, B) = W(A, B) + min(W(A, C), W(C, B))$ in a process called the *library extension*.


Figure 4.4: The initial weights, Example from [Notredame et al., 2000]

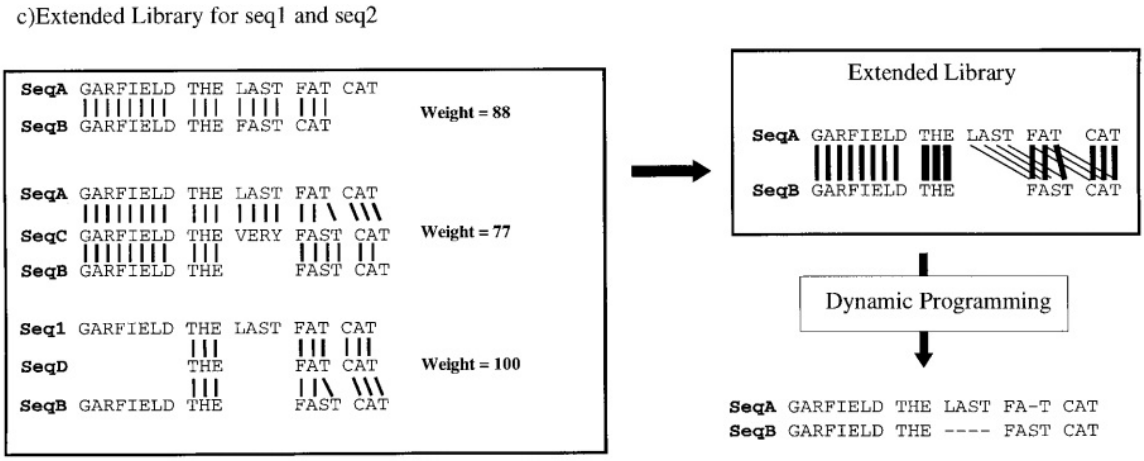



Figure 4.5: The updated weights, example from [Notredame et al., 2000]

For example, in figure 4.4, the alignment of SeqA and SeqB has 9/11 matches, so each aligned pair is assigned an initial weight of 88%. Similar weights are calculated for the alignment between SeqA and SeqC to give 77%, and between SeqB and SeqC to give 100%. When the library extension process uses seqC to update the weights of diagonals between seqA and seqB, the additional weight is $min(77, 100) = 77$. The final updated weights are represented by the thickness of the lines in the extended library.

The larger the number of sequences confirming a pair of positions, the higher weight the pair receives . Those weights are then used for the pairwise alignment steps in progressive alignment. During the pairwise alignment steps, gap penalties are set to zero. The consistency scores are strong enough to make them insensitive to gap penalties.

PROBCONS also implement a similar strategy, but the proposed formulas are designed with more probabilistic justification [Do et al., 2005]. The weight $W(A, B)$ in T-Coffee is now calculated as the posterior probability that $A$ and $B$ aligns. Then instead of updating weights by adding other weights, they perform a more sophisticated *probabilistic consistency transformation* that updates the probability of $A$ and

$B$ being aligned by the product of the probability of $A$ and $C$ being aligned and the probability of $C$ and $B$ being aligned:

$$\mathbf{P}'(x_i \sim y_j \in a^*|x,y) \leftarrow \frac{1}{|S|}\sum_{z\in S}\sum_{z_k}\mathbf{P}(x_i \sim z_k \in a^*|x,z)\mathbf{P}(z_k \sim y_j \in a^*|z,y)$$

Figure 4.6: Probabilistic consistency transformation [Do et al., 2005]. $\mathcal{S}$ is the set of input sequences, with $x, y, z \in S$. $x_i\ y_j \in a^*$ is the event that position $i$ of sequence $x$ is aligned with position $j$ of sequence $y$ in the unknown MSA $a^*$; $x_i$ corresponds to $A$, $y_j$ corresponds to $B$, and $z_k$ corresponds to $C$ in the previous paragraph

The probabilistic consistency transformation can be done multiple times. The obtained weights can be used for pairwise alignment in a way similarly to T-Coffee.

## 4.4 Iterative refinement

Iterative refinement works as follows. We start with a guide tree and a multiple alignment. In each iteration, we can pick some subtrees and realign sequences in each of those subtrees independently. The updated subtrees can then be merged to update the whole multiple alignment. At the same time, we may also try to make local changes on how subtrees are connected to each other. If the new alignment scores better than the old alignment, we start the next iteration with the new one. Otherwise, we continue with the old alignment.

While the idea is generally the same, different algorithms have different implementation of iterative refinement. A multiple sequence alignment method can ignore iterative refinement altogether because they do not define a scoring scheme for an alignment [Thompson et al., 1994]. They may define a simple criterion for the alignment such as the sum-of-pair score, and use that score to search for a better alignment while keeping the guide tree intact [Edgar, 2004] [Do et al., 2005]. They can also go to the other extreme where there is a likelihood measure for a guide tree together

with its associated multiple alignment, and the iterations are used to optimize the guide tree and the multiple alignment at the same time to maximize the likelihood [Liu et al., 2012].

## 4.5 Anchor based alignment

As described above, multiple sequence alignment is a hard problem with many approaches, which are usually computationally intensive. However, when we focus on a single conserved region across sequences, multiple sequence alignment becomes much easier.

For example, we are interested in the region of 16S rRNA of length 312 given below

```
TGGGCTACACACGTGCTACAATGGATGGAACAAAGGGCAGCGAAGCCGTGAGGCCAAGCAAATCCCACAAAA
CCATTCTCAGTTCGGATTGCAGGCTGCAACTCGCCTGCATGAAGCCGGAATCGCTAGTAATCGCGGATCAGC
ATGCCGCGGTGAATACGTTCCCGGGTCTTGTACACACCGCCCGTCACACCACGAGAGTTGGTAACACCCGAA
GTCGGTGAGGTAACCGTAAGGAGCCAGCCGCCGAAGGTGGGACCAATGATTGGGGTGAAGTCGTAACAAGGT
ACCGTATCGGAA
```

Let's name this region the *anchor string*, for reasons we will explain later. We can now take the anchor string and search for it in our set of sequences, which are sampled randomly from the database of 16S rNAs: AM980986, AY859682, DQ442546, AY613990, AB184869, Y17234. The following example is obtained by searching each of the input sequence for the anchor string using BLAST. The anchor string is labeled *Sbjct*. The *identities* (Section 4.3) of an alignment is the percentage of matches over the number of aligned positions.

```
AM980986 Actinocatenispora_rupis
```

```
 Identities = 176/217 (81%), Gaps = 1/217 (0%)

Query  1153  GGGCTTCACGCATGCTACAATGGCCGGTACAGAGGGCTGCGATACCGCAAGGTGGAGCGA  1212
              |||||  |||  |  ||||||||||  ||  |||  |||||  ||||  |||  |||   ||| |
Sbjct  2      GGGCTACACACGTGCTACAATGGATGGAACAAAGGGCAGCGAAGCCGTGAGGCCAAGCAA  61

Query  1213  ATCCCTAAAAGCCGGTCTCAGTTCGGATCGGGGTCTGCAACTCGACCCCGTGAAGTCGGA  1272
              |||||  |||  ||  ||||||||||||  |  | ||||||||||  |  | |||||  ||||
Sbjct  62     ATCCCACAAAACCATTCTCAGTTCGGATTGCAGGCTGCAACTCGCCTGCATGAAGCCGGA  121

Query  1273  GTCGCTAGTAATCGCAGATCAGCAACGGTGCGGTGAATACGTTCCCGGGCCTTGTACACA  1332
               ||||||||||||||  |||||||  | |  ||||||||||||||||||||  ||||||||||
Sbjct  122   ATCGCTAGTAATCGCGGATCAGC-ATGCCGCGGTGAATACGTTCCCGGGTCTTGTACACA  180

Query  1333  CCGCCCGTCACGTCACGAAAGTCGGTAACACCCGAAG  1369
              ||||||||||  |||||  |||  ||||||||||||||
Sbjct  181   CCGCCCGTCACACCACGAGAGTTGGTAACACCCGAAG  217


AY859682 Mycobacterium_phocaicum
 Identities = 242/297 (81%), Gaps = 4/297 (1%)

Query  1186  GGGCTTCACACATGCTACAATGGCCGGTACAAAGGGCTGCGATGCCGTGAGGTGGAGCGA  1245
              |||||  |||||  ||||||||||  ||  |||||||||  ||||  |||||||||   ||| |
Sbjct  2      GGGCTACACACGTGCTACAATGGATGGAACAAAGGGCAGCGAAGCCGTGAGGCCAAGCAA  61

Query  1246  ATCCTTTCAAAGCCGGTCTCAGTTCGGATCGGGGTCTGCAACTCGACCCCGTGAAGTCGG  1305
              ||||   ||||  ||  ||||||||||||  |  | ||||||||||  |  | |||||  |||
Sbjct  62     ATCCCA-CAAAACCATTCTCAGTTCGGATTGCAGGCTGCAACTCGCCTGCATGAAGCCGG  120

Query  1306  AGTCGCTAGTAATCGCAGATCAGCAACGCTGCGGTGAATACGTTCCCGGGCCTTGTACAC  1365
              | ||||||||||||||  ||||||||  ||  ||||||||||||||||||||  |||||||||
Sbjct  121   AATCGCTAGTAATCGCGGATCAGCAT-GCCGCGGTGAATACGTTCCCGGGTCTTGTACAC  179

Query  1366  ACCGCCCGTCACGTCATGAAAGTCGGTAACACCCGAAGCCGGTGGCCTAACCCTTGTGGA  1425
              ||||||||||||  ||  ||  |||  ||||||||||||||  |||||   |||  ||  |  |||
Sbjct  180   ACCGCCCGTCACACCACGAGAGTTGGTAACACCCGAAGTCGGTGAGGTAA-CCGTAAGGA  238

Query  1426  GGGAGCCGTCGAAGGTGGGATCGGCGATTGGGACGAAGTCGTAACAAGGTAGCCGTA  1482
              |  |||||  ||||||||||  |   |||||||  ||||||||||||||||  |||||
Sbjct  239   GCCAGCCGCCGAAGGTGGGACCAATGATTGGGGTGAAGTCGTAACAAGGTA-CCGTA  294

DQ442546 Streptomyces_sulphureus
 Identities = 191/231 (83%), Gaps = 1/231 (0%)
```

```
Query  1181  TGGGCTGCACACGTGCTACAATGGCCGGTACAATGAGAGGCGAGGCCGTGAGGTGGAGCG  1240
              ||||||  ||||||||||||||||  || ||||  | |  ||||  |||||||||   |||
Sbjct  1      TGGGCTACACACGTGCTACAATGGATGGAACAAAGGGCAGCGAAGCCGTGAGGCCAAGCA  60

Query  1241  AATCTCAAAAAGCCGGTCTCAGTTCGGATTGGGGTCTGCAACTCGACCCCATGAAGTCGG  1300
              ||||  ||  |||  ||  ||||||||||||||  | ||||||||||  |  |||||||  |||
Sbjct  61    AATCCCACAAAACCATTCTCAGTTCGGATTGCAGGCTGCAACTCGCCTGCATGAAGCCGG  120

Query  1301  AGTCGCTAGTAATCGCAGATCAGCATTGCTCGGTGAATACGTTCCCGGGCCTTGTACACA  1360
              | ||||||||||||||  |||||||||  | ||||||||||||||||||||  ||||||||||
Sbjct  121   AATCGCTAGTAATCGCGGATCAGCATGCCGCGGTGAATACGTTCCCGGGTCTTGTACACA  180

Query  1361  CCGCCCGTCACGTCACGAAAGTCGGTAACACCC-AAGCCGGTGGCCTAACC  1410
              ||||||||||  |||||  |||  ||||||||||  |||  |||||   |||||
Sbjct  181   CCGCCCGTCACACCACGAGAGTTGGTAACACCCGAAGTCGGTGAGGTAACC  231


AY613990 Kitasatospora_viridis
 Identities = 229/278 (82%), Gaps = 3/278 (1%)

Query  1144  TGGGCTGCACACGTGCTACAATGGCCGGTACAAAGGGCTGCGATACCGTGAGGTGGAGCG  1203
              ||||||  ||||||||||||||||  || |||||||||  ||||  ||||||||   |||
Sbjct  1      TGGGCTACACACGTGCTACAATGGATGGAACAAAGGGCAGCGAAGCCGTGAGGCCAAGCA  60

Query  1204  AATCCCAAAAAGCCGGTCTCAGTTCGGATTGGGGTCTGCAACTCGACCCCATGAAGTTGG  1263
              |||||||  |||  ||  ||||||||||||||  | ||||||||||  |  |||||||  ||
Sbjct  61    AATCCCACAAAACCATTCTCAGTTCGGATTGCAGGCTGCAACTCGCCTGCATGAAGCCGG  120

Query  1264  AGTTGCTAGTAATCGCAGATCAGCATG-TGCGG-GAATA-GTTCCCGGGCCTTGTACACA  1320
              | | ||||||||||||  ||||||||||  ||||  |||||  |||||||||  ||||||||||
Sbjct  121   AATCGCTAGTAATCGCGGATCAGCATGCCGCGGTGAATACGTTCCCGGGTCTTGTACACA  180

Query  1321  CCGCCCGTCACGTCACGAAAGTCGGTAACACCCGAAGCCGGTGGCCTAACCCTTGGGAGG  1380
              ||||||||||  |||||  |||  ||||||||||||||  |||||   |||||  |  ||||
Sbjct  181   CCGCCCGTCACACCACGAGAGTTGGTAACACCCGAAGTCGGTGAGGTAACCGTAAGGAGC  240

Query  1381  GAGCCGTCGAAGGTGGGACCAGCGATTGGGACGAAGTC  1418
               |||||  ||||||||||||||  |||||||  ||||||
Sbjct  241   CAGCCGCCGAAGGTGGGACCAATGATTGGGGTGAAGTC  278


AB184869 Streptomyces_bambergiensis
 Identities = 237/295 (80%), Gaps = 6/295 (2%)

Query  1166  TGGGCTGCACACGTGCTACAATGGCCGGTACAATGAGCTGCGATACCGCGAGGTGGAGCG  1225
```

```
            |||||| ||||||||||||||||||  || |||| | || ||||  ||| ||||   |||
Sbjct  1    TGGGCTACACACGTGCTACAATGGATGGAACAAAGGGCAGCGAAGCCGTGAGGCCAAGCA  60

Query  1226 AATCTCAAAAAGCCGGTCTCAGTTCGGATTGGGGTCTGCAACTCGACCCCATGAAGTCGG  1285
            |||| || ||| ||  ||||||||||||||  | |||||||||| |  ||||||| |||
Sbjct  61   AATCCCACAAAACCATTCTCAGTTCGGATTGCAGGCTGCAACTCGCCTGCATGAAGCCGG  120

Query  1286 AGTTGCTAGTAATCGCAGATCAGCATTGCTGCGGTGAATACGTTCCCGGGCCTTGTACAC  1345
            | | |||||||||||| |||||||| ||| |||||||||||||||||||| |||||||||
Sbjct  121  AATCGCTAGTAATCGCGGATCAGCA-TGCCGCGGTGAATACGTTCCCGGGTCTTGTACAC  179

Query  1346 ACCGCCCGTCACGTCACGAAAGTCGGTAACACCCGAAGCCGGTGGCCCAACCCCCTTGCG  1405
            ||||||||||||  ||||| ||| |||||||||||||| |||||    ||||     |
Sbjct  180  ACCGCCCGTCACACCACGAGAGTTGGTAACACCCGAAGTCGGTGAGGTAACC-----GTA  234

Query  1406 GGGAGGGAGCCGTCGAAGGTGGGACTGGCGATTGGGACGAAGTCGTAACAAGGTA  1460
             ||||  ||||| ||||||||||||    |||||||  ||||||||||||||||
Sbjct  235  AGGAGCCAGCCGCCGAAGGTGGGACCAATGATTGGGGTGAAGTCGTAACAAGGTA  289

Y17234 Microbacterium_laevaniformans
 Identities = 233/291 (80%), Gaps = 2/291 (1%)

Query  1128 TGGGCTTCACGCATGCTACAATGGCCGGTACAAAGGGCTGCAATACCGTGAGGTGGAGCG  1187
            |||||| ||| | ||||||||||  || ||||||||| || |  ||||||||   |||
Sbjct  1    TGGGCTACACACGTGCTACAATGGATGGAACAAAGGGCAGCGAAGCCGTGAGGCCAAGCA  60

Query  1188 AATCCCAAAAAGCCGGTCCCAGTTCGGATTGAGGTCTGCAACTCGACCTCATGAAGTCGG  1247
            ||||||| ||| ||  || ||||||||||||  | |||||||||| |  ||||||| |||
Sbjct  61   AATCCCACAAAACCATTCTCAGTTCGGATTGCAGGCTGCAACTCGCCTGCATGAAGCCGG  120

Query  1248 AGTCGCTAGTAATCGCAGATCAGCAACGCTGCGGTGAATACGTTCCCGGGTCTTGTACAC  1307
            | |||||||||||||| ||||||| | || ||||||||||||||||||||||||||||||
Sbjct  121  AATCGCTAGTAATCGCGGATCAGC-ATGCCGCGGTGAATACGTTCCCGGGTCTTGTACAC  179

Query  1308 ACCGCCCGTCAAGTCATGAAAGTCGGTAACACCTGAAGCCGGTGGCCCAACCCTTGTGGA  1367
            ||||||||||   || || ||| ||||||||| |||| |||||    || || |  |||
Sbjct  180  ACCGCCCGTCACACCACGAGAGTTGGTAACACCCGAAGTCGGTGAGGTAA-CCGTAAGGA  238

Query  1368 GGGAGCCGTCGAAGGTGGGATCGGTAATTAGGACTAAGTCGTAACAAGGTA  1418
            |  ||||| |||||||||| |  | ||| ||   ||||||||||||||||
Sbjct  239  GCCAGCCGCCGAAGGTGGGACCAATGATTGGGGTGAAGTCGTAACAAGGTA  289
```

Given these alignments, we roughly know how input sequences should align: two

sites from two different sequences should be aligned if they are aligned to the same site in our anchor string. In the example above, position 1128 of Microbacterium laevaniformans sequence should align with position 1166 of the sequence of Streptomyces bambergiensis because they both align to the first position of our anchor string. The motivation behind the chosen terminology, *anchor string*, follows from the fact that it helped us align sites from different sequences.

By introducing the concept of *anchor string*, objects from two different sequences can be directly compared. Given an anchor string $S_0$:

- The *anchor of a position* $S_i$ of a sequence $\mathcal{S}$ is the position in the anchor string that $S_i$ is aligned to when we align $\mathcal{S}$ against $S_0$. Two positions $a_i$, $b_j$ of two different sequences $a$ and $b$ should be aligned if they have the same anchor, i.e. they are aligned to the same position in the anchor string.

- The *anchor of a region/interval* $(x, y)$ in a sequence $S$ is a pair $(x', y')$ of positions in the anchor string such that $x'$ is the anchor of $S[x]$, $y'$ is the anchor of $S[y]$, and the region $S[x, y]$ is aligned to the region $S_0[x', y']$.

Ideally, each position in a sequence $\mathcal{S}$ has at most one anchor; that is, it is only aligned with one position in the anchor string. There are scenarios where this fails to happen: the anchor string contains a repetitive element, or the alignment near a gap is unreliable. We can avoid having repetitive elements by removing such elements from our anchor string. The unreliability of the alignments near a gap is a more prevalent problem.

For example, we have two equally likely single-gap alignments shown below, observed when aligning Actinocatenispora rupis sequence against Virgibacillus proomii sequence.

```
GCAGATCAGCAACGGTG  or  GCAGATCAGCAACGGTG (Actinocatenispora rupis)
GCGGATCAGC-ATGCCG      GCGGATCAGCA-TGCCG (Virgibacillus proomii)
```

Note that the letter 'A' of the bottom sequence can be aligned to two possible anchors in the top sequence. This problem arises because the position of a gap is not clear. We know that it must be in some region of the anchor string, but the particular position in this case is ambiguous. While the region of ambiguity can be as short as 2 bases, it can also be as long as a dozen bases in the following example obtained from aligning Virgibacillus proomii sequence against Staphylococcus sciuri sequence.

```
ATAGGGAGTTCCCTTCGGGGA--CAGAGTGAC (Staphylococcus sciuri)
ATAGAGTCTTCCCCTTCGGGGGACAAAGTGAC (Virgibacillus proomii)
or
ATAGGGAGTT--CCCTTCGGGGACAGAGTGAC (Staphylococcus sciuri)
ATAGAGTCTTCCCCTTCGGGGGACAAAGTGAC (Virgibacillus proomii)
```

Such ambiguity in pairwise alignment leads to more ambiguity in multiple alignment. In the first example, note that there is often a gap near position 145 in our anchor string. We extract the alignments around that gap in different sequences as follows.

```
AGC-ATG
AGCAT-G
AGCATG-
AGCA-TG
AGC-ATG
```

```
DQ442546 no gap
AY613990 gap on the opposite strand
```

Given that the gaps occur in close proximity, and that indels happen much less frequently than point substitution, it is plausible to hypothesize that the above gaps all result from a single indel event. While in the above example, the position of the indel event is not clear, from the given data, its length is clearly 1.

### 4.5.1 Finding Insertion/Deletion events

Sections 5 and 5.4 describe many phylogeny inference methods that require their input to be aligned sequences. Those methods aim to find an evolutionary hypothesis that explains how individual homologous regions evolved from a common ancestor. Therefore the identification of those individual homologous regions becomes very useful.

As gap boundaries are often unclear as discussed above (Section 4.5), how do they affect the traditional approaches of multiple sequence alignment and phylogeny inference?

A multiple sequence alignment algorithm will be forced to return a single answer. A phylogeny inference method then has two choices. It can either assume that the alignment is correct, and proceed with possible errors that might have been introduced; otherwise it can truncate near the boundaries of gaps, and proceed with alignments sufficiently removed from gaps.

Let us suppose we took the second choice, that is, we cannot reliably align the boundaries of gaps for use in phylogeny inference. We saw above that while gap boundaries are often not clear, their length is more easily obtained with confidence (Section 4.5). If two gaps share the same length, and are aligned to the same region, it is likely that they correspond to the same insertion/deletion event.

Using gap lengths in inferring phylogenies offer another advantage. When se-

quences are further apart, homoplasy happens more frequently: a base A can be mutated to G, and then to A again. This confuses character-based methods (maximum parsimony and maximum likelihood), as well as distance based method (Neighbor Joining). This is a problem because we only have 4 possible characters. However, the problem is less severe with gaps. It is very unlikely for a deletion to follow an insertion and completely neutralize it, for example $AC \to ATC \to AC$. It is even less likely for an insertion to follow a deletion an completely neutralize it, for example $AGTGC \to AC \to AGTGC$.

### 4.5.2 Gap detection algorithm

Consider this toy example, where two alignments of "AGCAG" with "AGCCAG" are shown:

```
AGC_AG  or AG_CAG  ?
AGCCAG     AGCCAG
```

In this setting it is not important whether the mutation was a deletion or an insertion. Either way, it is represented by a single gap ("_") character. Notice that the interval between strings "AG" and "AG" in the upper sequence has length 1, and in the lower sequence has length 2. We can therefore conclude that there must be a gap of length 1 in the upper sequence, even when the position of the gap is unclear.

With this example in mind, we design an algorithm that takes a sequence as an anchor, finds gaps in other sequences when aligned to the anchor, and identifies common gaps that appear in several sequences. Such gaps are evidence of insertion/deletion events, and sequences that share a common piece of evidence are expected to be close in the phylogeny. A maximum parsimony, maximum likelihood, or even Neighbor Joining algorithm can make use of such information.

Given a set of sequences $\mathcal{S} = \{S_1, S_2, ..., S_n\}$, there are several ways to pick an anchor sequence. It may be a random sequence in $\mathcal{S}$, or some homologous sequence not belonging to $\mathcal{S}$. The anchor sequence can also be an artifact that we create by concatenating substrings from $\mathcal{S}$. A good anchor sequence $S_0$ possesses the following qualities.

- $S_0$ contains no approximate repetitive substring.
- When any sequence $S_i$ in $\mathcal{S}$ is aligned against $S_0$, most positions of $S_i$ can be aligned to exactly one position in $S_0$ with confidence.
- If a region $A$ precedes another region $B$ in $S_i$, its alignment $A'$ should also precede $B$'s alignment $B'$ in $S_0$ (order preservation).

Given the selected anchor sequence $S_0$, for each $S_i \in \mathcal{S}$ we can find its homologous regions and gaps in-between as follows.

- Find all significant gap-free local alignments (matches) between $S_0$ and $S_i$. We can sort them by their positions in $S_0$. We number the matches in their sorted order to be $M_j, j = 1, .., m$. Each match $M_j$ is a vector with 4 components $M^0_{j,L}$, $M^0_{j,R}$, $M^i_{j,L}$, $M^i_{j,R}$ with $[M^i_{j,L}, M^i_{j,R}]$ being an interval/region in $S_i$ and $[M^0_{j,L}, M^0_{j,R}]$ being the anchor of this interval.
  Because the matches are gap-free, we have $M^0_{j,R} - M^0_{j,R} = M^i_{j,R} - M^i_{j,R}$ for any $M_j$.
- For each pair of subsequent matches $(M_j, M_{j+1})$, the gap between them is *anchored* at interval $[M^0_{j,R}, M^0_{j+1,L}]$, with their length calculated as $(M^i_{j+1,L} - M^0_{j+1,L}) - (M^i_{j,L} - M^0_{j,L})$

Here we introduce the notion *anchor of a gap*, which is determined by its adjacent matches. Once gaps from different sequences are detected, they can be grouped into

collections of gaps with the same length and proximal positions. Such a collection of gaps is expected to result from a single insertion/deletion.

Here we will demonstrate how the algorithm works on a toy example consisting of the following sequences

```
index  0123456
S0     AGTTAGT
S1     AGAG
S2     AGCGT
```

where the matches found between S0 and S1 is AG/AG, and between S0 and S2 is GT/GT.

```
Anchoring
S0     AGTTAGT
S1     AG  AG
S2     AG   GT

Aligning s0-S1:
Match 0: anchor [0,1], interval [0,1]
Match 1: anchor [4,5], interval [2,3]
 --> gap(1,4), length = (2 - 4) - (0 - 0) = -2
Aligning s0-S2:
Match 0: anchor [0,1], interval [0,1]
Match 1: anchor [5,6], interval [3,4]
 --> gap(1,5), length = (3 - 5) - (0 - 0) = -2
The two gaps overlap and have equal length, so
we group them into the same collection.
```

In the previous example, the letter 'C' in S2 may also be aligned to either 'A' or 'T' without any preference. While gap anchors do not have to be correct, gap lengths can usually be calculated with high accuracy.

We perform this gap detection algorithm on the LTP dataset (Chapter 3). We extract 20 sequences and print out the raw result, which is the grouping of leaves sharing some common gap. There are two examples, corresponding to two different ways of sampling species: to take a random subtree with approximatly 20 leaves, or to sample 20 leaves over all 8000 sequences.

In the first example, sequences are closer to each other. Their phylogeny from the LTP project is shown in Figure 4.7. We replace the species names by IDs to make it easy to follow.

Our algorithm has no access to the standard phylogeny, nor is it designed to do any phylogeny inference. It detects gaps, and attempts to group them. A group of gaps is hypothesized to correspond to one single insertion/deletion event. We expected to see sequences co-occurring in the same group to be close to each other in the standard phylogeny, and we actually observed that.

Gap collections found are listed below, with each line consisting of the IDs of sequences belonging to the same collection. The way to read this result is to see each line as a collection of sequences. Each collection is annotated with an interpretation with respect to the tree in Figure 4.7. If many collections are found as subtrees in the standard tree, the grouping of gaps would be a good signal for phylogeny inference.

```
0 1 : subtree at node 2
6 20: false positive
12 26: false positive
25 28: subtree at node 31
```

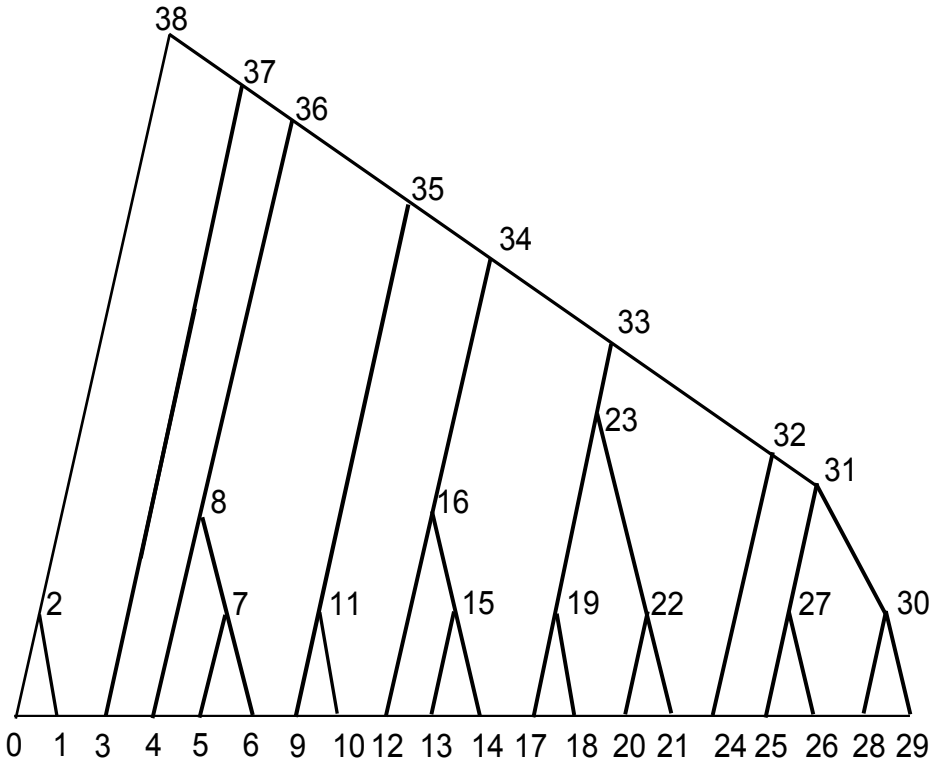


Figure 4.7: LTP subtree consisting of roughly 20 sequences

```
0 1 3 4 5 6: complement of leaves in subtree at node 36
12 25 26 28: subtree at node 31
9 12 13 18 26: false positive
0 1 3 4 5 6 10 17 21: complement of leaves in subtree at node 35
9 10 13 14 17 18 20 21: subtree at node 35
...
```

In the second example, sequences are farther from each other. Their phylogeny from the LTP project is shown in Figure 4.8.

Gap groups found are listed below, with each line consisting IDs of sequences belonging to the same group.

```
5 6: subtree at node 7
```

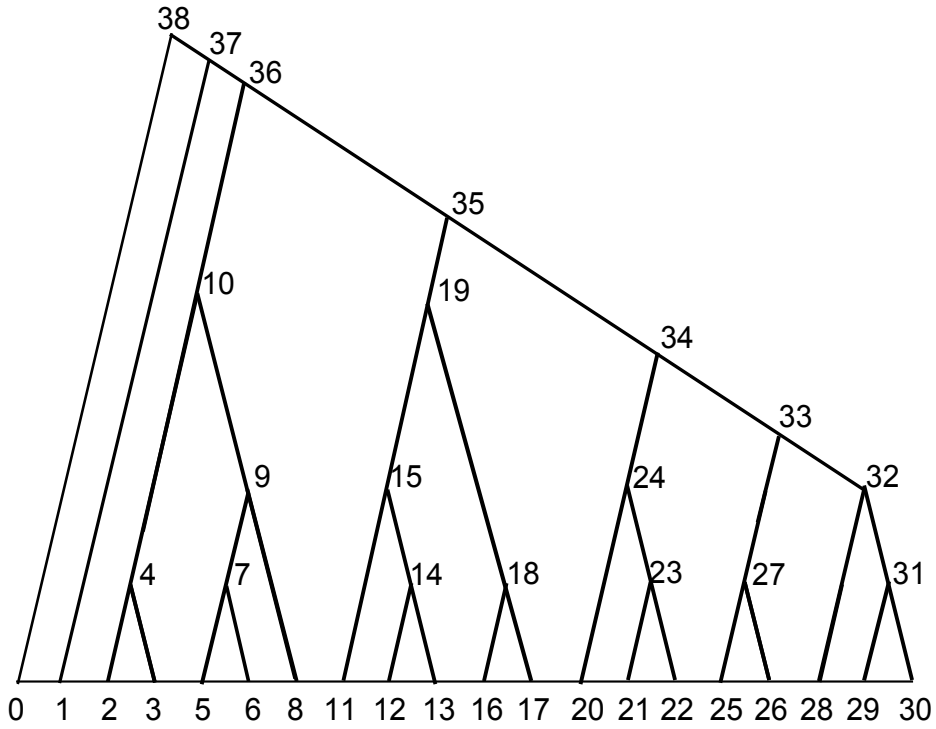


Figure 4.8: LTP restricted to a sample of 20 leaves

```
6 8: subtree at node 9

1 22: false positive

3 21: false positive

5 29: false positive

5 30: false positive

8 26: false positive

11 17: false positive

20 28: false positive

25 26: subtree at node 27

28 30: subtree at node 32

29 30: subtree at node 31

1 16 17: subtree at node 18

1 25 26: subtree at node 27

3 8 13 21: false positive
```

```
2 21 29 30: false positive
3 5 8 13 21: false positive
3 5 20 25 28: false positive
3 5 8 13 21 22: false positive
3 5 20 25 26 28: false positive
2 12 16 25 26 28: false positive
1 11 12 16 17 20 28: subtree at node 35
```

The perfect phylogeny method (Chapter 5) finds a consensus tree from a given set of splits. It never became practical, because we could not find good splits that agree with the underlying phylogeny. The straightforward splits obtained from clustering all sequences sharing the same base at a given column never worked, even for well-conserved columns, because there are often substitutions happening in different branches of the phylogeny that mutate into the same base.

Such problem is less severe with characters based on gap length: it is less likely to have two insertions happening in different branches of the phylogeny that have the same length. Given the clusters of gaps from our new algorithm, it is tempting to find ways to use these clusters in a similar approach. When we compare the clusters with the standard phylogeny from the LTP project, we see that they do not agree 100%. However, with the first dataset of nearby sequences, we can often find a split that agrees with a gap collection, off by a few nodes. This suggests that the signals obtained from gaps are stronger than those obtained from single base comparison. What is left is to find a way to utilize these signals to improve the current phylogeny inference methods.

# Chapter 5

# Phylogeny inference methods

## 5.1 Maximum Parsimony

Given a set of sequences $\mathcal{S}$, this method finds a phylogeny $t(\mathcal{S})$ as a binary tree whose leaf nodes correspond to the members of $\mathcal{S}$. As a general criterion for the selection of $t(\mathcal{S})$, each edge is assigned a weight based on some metric, and $t(\mathcal{S})$ is selected as a tree minimizing the total weights of edges. See Fig.5.1 for an example with the Hamming distance as edge weights.

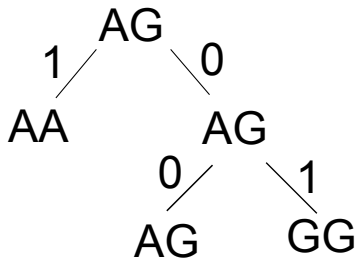



Figure 5.1: $\mathcal{S} = AA, AG, GG$, $t(\mathcal{S})$ has a total weight of 2

The weights assigned to phylogeny edges found by maximum parsimony are frequently Hamming distances. They reflect the number of mutation events required to explain the evolution along an edge in $t(\mathcal{S})$. A maximum parsimony tree $t(\mathcal{S})$

minimizes the number of hypotheses (mutation events) required to explain the given observations (sequences).

A *perfect phylogeny* is a phylogeny that explains the observed sequences $\mathcal{S}$ with at most one mutation event per position in the whole tree. It is a special case of maximum parsimony, where each site mutates at most once in the whole history. There is a fast and provably correct algorithm to find the tree and its internal nodes ([Saitou and Nei, 1987]).

Perfect phylogeny rarely works with real datasets, because

- The same base can appear in two or more disjoint set of leaves.
- Sites are treated equally, regardless of their possibly different mutation rates.

## 5.2 Maximum likelihood

The maximum parsimony method aims to find the smallest number of mutations that explains the evolution of observed sequences. By relying on the mere count of mutations, the maximum parsimony method implicitly assumes that all mutation events are independent and equally significant.

However, this assumption is not realistic. If we have inferred the possible mutations in a set of homologous sequences, we can make predictions about another homologous sequence.

- We expect to find mutations in the less conserved regions than in the more conserved ones.
- If two regions $A$ and $B$ have similar variability, and $A$ shows high similarity

to a known sequence, then $B$ should not diverge too much. The reason is that realistically each region is exposed to the same interval of evolution.

- Different mutations have different chances of happening. Insertions/deletions happen much less frequently than substitutions. Different substitutions also have different chance of happening: we would not necessarily expect that it is equally likely for A to be substituted by C, G or T.

Once we want to model these properties, we need a more sophisticated method than merely counting the number of mutations. Maximum likelihood is a framework that embodies this idea naturally.

Maximum likelihood assumes an evolutionary model that assigns a probability to each mutation, and finds a tree that maximizes the probability conditioned on the sequences. Most maximum likelihood variants assume independent mutations among sites, so that the probability of a tree of sequences can be written as the product of the probabilities of trees of characters, and are also called character based method sometimes.

Given a set of sequences $\mathcal{S}$ and a phylogeny $t(\mathcal{S})$ over these sequences, we want to measure the likelihood that the phylogeny reflects the true underlying evolution process that generated $\mathcal{S}$. For simplicity, we usually work on aligned sequences, and therefore assume a fixed length $l$ for all sequences. For an index $i$, we can replace each sequence $S_u$ in $t(\mathcal{S})$ by its $i$-th character. The resulting phylogeny $t(\mathcal{S}_i)$ has the same structure as the original phylogeny, but each node is only a single character. Suppose we can calculate the likelihood $L(t(\mathcal{S}_i))$ of such a single-character phylogeny, then the likelihood $L(t(\mathcal{S}))$ of the original phylogeny can then be calculated as the product of the likelihoods $L(t(\mathcal{S}_i))$ over all $i = 1, ..., l$.

$$L(t(\mathcal{S})) = \prod_{i=1}^{l} L(t(\mathcal{S}_i))$$

The likelihood of a tree of characters is the sum of likelihoods with different bases at the root.

$$L(t(\mathcal{S}_i)) = \sum_{b \in \{A,C,G,T\}} \pi_b L(t(\mathcal{S}_i) | R(t)_i = b)$$

Here $\pi_b$ is the probability of having nucleotide $b$. $R(t)$ is the common ancestor sequence of $t(\mathcal{S})$, which we may call $t$ for short. $R(t)_i$ is the $i$-th character of $R(t)$. In this formula we assumed the same nucleotide distribution along the sequence and among species.

The quantity $L(t|R(t)_i = b)$ can be recursively computed from its subtrees $t_i$ and $t_j$ as follows.

$$L(t|R(t)_i = b) = \prod_{x \in \{i,j\}} \sum_{c \in \{A,C,G,T\}} P_{bc}(\delta_{ax}) L(t_x | R(t_x)_i = c)$$

Here $\delta_{ax}$ is the estimated branch length at the root node to its subtree $x$, and $P_{bc}(\delta_{ax})$ is the rate of mutation from character $b$ to character $c$, given the estimated branch length.

If we assume branch lengths to be constant and that the mutation rate is very small, maximum likelihood becomes maximum parsimony.

## 5.3 Clustering methods

While maximum likelihood assumes a model and tries to find some result that best explains the observations, it is not the only paradigm. Phylogeny inference can also

be seen as a generalization of the clustering problem. Suppose we want to infer a phylogeny $t(\mathcal{S})$ over a set of sequences $\mathcal{S}$, $|\mathcal{S}| = n$. Each edge of $T$ can be seen as a partition of $n$ sequences into two sets of leaves. We expect the sequences in the same set to show a higher degree of similarity among themselves than with the sequences in the other set. A natural implementation of this scheme is to recursively partition the input sets to obtain a hierarchical clustering tree as the output.

This framework is well suited toward combining phylogenies. Each phylogeny will define a set of partitioning (or splits). If we obtain different phylogenies from different methods, one way to combine them is to find a subset of leaves that all the splits from different phylogenies agree on. Another way is to find the most common splits that agree on the original set of leaves.

A perfect phylogeny can also be seen as an instance of clustering methods. Suppose the sequences are already aligned to obtain a matrix of $n$ rows and $l$ columns. If a column contains only two bases, we can define a split based on this column: sequences sharing the same base would be in the same partition. If the splits we obtain from all the columns do not conflict with each other, we have a perfect phylogeny. It is interesting to see how perfect phylogeny lies in the intersection between maximum parsimony and clustering methods.

Neighbor-Joining ([Saitou and Nei, 1987], [Gascuel and Steel, 2006], [Tamura et al., 2004]) is designed from the other extreme (bottom-up): it combines all the columns to obtain one single distance measure. Usually, the distance used is the edit distance or some of its variant. While for perfect phylogenies, any difference in a single column results in a split, Neighbor Joining (NJ) does not take individual columns into consideration.

NJ first finds a split $(X, Y)$ where $|X| = 2$. The criterion is similar to that of clustering: minimize the distance within X, while maximizing the distance between

$X$ and $Y$. Once such a split is found, the common parent of the two leaves in X replaces them, and the algorithm is iterated. To be clear, in the original formulation of NJ, the common parent is not expressed as a sequence, but its distances to leaves in Y are estimated.

It is extremely hard to come up with a stochastic model that captures all the properties of evolution. Most of the time, we either use too few or too many parameters. Suppose the common ancestor $R$ evolved into two sequences $S_1$ and $S_2$. We would expect that the difference between $R$ and $S_1$ is comparable to that between $R$ and $S_2$, since they are both exposed to the same amount of evolution time. However, there are many other factors which are difficult to model: the mutation rate may vary among sites, lineages, and period in history. The sites may not even mutate independently.

With a limited number of columns, we try to estimate the different parameters that describe the mentioned effects. By estimating fewer parameters, NJ tries to avoid overfitting. This may be the reason why it works reasonably well across different datasets. It has also been criticized as not utilizing all the information presented in the sequence data. This is the unavoidable trade-off when we want to reduce the number of parameters in the model. NJ works better when we have longer sequences to obtain better estimates of sequence distances. As the length of input sequences is decreased, the accuracy of NJ reduces substantially.

## 5.4 Neighbor Joining and its variants

Many multiple sequence alignment algorithms refer to some guide tree. Maximum likelihood and maximum parsimony phylogeny inference methods also utilize some initial tree to limit the searching space. Due to its speed and reasonable accuracy in

different applications, Neighbor Joining [Saitou and Nei, 1987] is usually the method of choice to create the initial guide tree.

Suppose we want to infer the phylogeny $t(\mathcal{S})$ for some set of sequences $\mathcal{S}$, $|\mathcal{S}| = n$, then Neighbor Joining (NJ) takes in as it input a matrix $d_{n\times n}$, referred to for convenience as distance matrix. The entries of this matrix comply with the following three conditions:

- $d_{i,i} = 0, \forall i$
- $d_{i,j} \geq 0, \forall i, j$
- $d_{i,j} = dj, i, \forall i, j$

Due to convention and convenience, we use the terminology "distance matrix" even though the distances do not necessarily satisfy the triangular inequality.

The Neighbor-Joining algorithm proceeds as follows:

1. Compute a matrix $Q_{n\times n}$ where

$$Q_{i,j} = d_{i,j} - \frac{1}{n-2}(\sum_{k\neq i} d_{i,k} + \sum_{k\neq j} d_{j,k}) \tag{5.1}$$

2. *Q-criterion* - Select $i$,$j$ with smallest $Q_{i,j}$. Connect them to a common parent $u$. Replace $i$ and $j$ by $u$ in the set of leaves. For any other leaf $x$, the distance to $u$ is updated to

$$d_{x,u} = d_{u,x} = \frac{1}{2}(d_{x,i} + d_{x,j} - d_{i,j}) \tag{5.2}$$

3. Repeat from Step 1 until only three taxa are left.

A *cherry* is a pair of nodes with a common parent [Radu Mihaescu, 2007]. NJ

algorithm iterates between finding a cherry with equation (5.1), merging them and updating the new distances by equation (5.2).

NJ assumes the distance metric is tree-additive. It also works if we slightly perturb additive distance metrics, as shown in the following implementation:

1. [Studier and Keppler, 1988] If $d$ is tree-additive, $(S_i, S_j)$ is a cherry in the real phylogeny $t(\mathcal{S})$.

2. [Bryant, 2005] The NJ selection criterion (Q-value) is the only linear function on distances that gives the correct result for tree additive metrics.

3. [Atteson, 1999] Let $D_{n\times n}$ where $D_{i,j}$ is the tree distance between $S_i$ and $S_j$ in $t(\mathcal{S})$. If the $l_\infty$ distance between $d$ and $D$ is smaller than half the smallest element in $D$, NJ returns the correct tree.

Interestingly, NJ branch length estimates are often non-additive. As $d_{i,j} = d_{u,i} + d_{u,j}$, we can expand equation (5.2) as follows.

$$d_{u,x} = \frac{1}{2}(d_{x,i} + d_{x,j} - d_{u,i} - d_{u,j}) = \frac{(d_{x,i} - d_{u,i}) + (d_{x,j} - d_{u,j})}{2}$$

For real data, usually the metric is not exactly additive. In that case, $d_{x,i} - d_{u,i} \neq d_{x,j} - d_{u,j}$. Since $d_{u,x}$ is the average among these two terms, clearly it would not be equal to any of them. Moreover, one of the inequalities will also break the triangle inequality:

$$d_{x,i} - d_{u,i} > d_{u,x} \Rightarrow d_{x,i} > d_{u,i} + d_{u,x}$$

In short, while aiming at reproducing an additive metric, NJ output fails to be a metric.

To fix this, we can simply add a large constant to all $d_{x,u}$ without affecting the subsequent NJ rounds. However, it would be interesting to look into the main cause of this phenomenon.

NJ takes in a distance matrix $d$ as its input. The distances are usually pairwise edit distance or some corrected version. If we use the same distance metric to obtain a weighted version of the real phylogeny $t(\mathcal{S})$, we can define another matrix $D_{|S|\times|S|}$ with entries being the distance in $t(\mathcal{S})$ defined by equation (1.1). As $D$ is tree-additive, NJ will run correctly if we have $D$ as the input instead of $d$. The problem is that we do not have $D$. In fact, $d$ is often an underestimate of $D$.

$$d_{i,j} \leq D_{i,j}, \forall (i,j)$$

In NJ, as new distances are calculated from old distances, any error in the initial estimate is propagated further. If we know the sequence of the common parent, the new distances can be calculated from pairwise edit distances, which more closely approximate the tree distance $D$.

However, the sequence of the common parent is also unknown, so we try to find it using different heuristics. The heuristics can be plugged into the original NJ algorithm by the following framework.

1. Obtain $d$ by edit distances
2. Find a cherry $(x, y)$ to be merged using the NJ criterion (5.1)
3. Use a heuristic to obtain the sequence $S_u$ of the common parent $u$, and then estimate the new distances $d(u, x)$ for all other nodes $x$ by comparing $S_u$ with $S_x$.
4. Replace $x$ and $y$ by $u$ in the set of leaves

5. If there are more than 1 species left, jump back to (2)

Finding a good description of the internal nodes in the phylogeny is an interesting problem in its own right. It provides a better estimate of phylogeny edge weights to be used to estimate the evolutionary divergence between species. Here we present a few heuristic methods to estimate the common parent. We assume that the input sequences have been aligned by some multiple sequence alignment algorithm.

### 5.4.1 Centroid method

Suppose we want to merge the cherry $(x, y)$. If at one aligned position, both sequences have the same base, the common parent is assumed to have that base. However, if the two bases are different, we need another sequence to resolve which base to be assigned to the parent.

In this centroid method, we pick another sequence $u$ that is sufficiently close to $x$ and $y$, using the minimum value of $d_{x,u} + d_{y,u}$. The common parent sequence would be the result of majority voting at all positions.

Clearly, positions with three different bases still cannot be resolved. We expect the number of such cases to be small, due to the proximity of $x$, $y$ and $u$. In the few cases where majority voting failed, in other words $x_i$, $y_i$ and $u_i$ are pairwise distinct, we greedily pick a random base among $\{x_i, y_i, u_i\}$, expecting that minor errors might be introduced to the estimation of distances.

An alternative to this greedy approach is described in the next subsection.

### 5.4.2 Parsimony method

Positions where $x$ and $y$ are different would introduce ambiguity to the common parent. One way to handle that is to leave them undecided, and use Fitch algorithm [Fitch, 1971] to decide the base at ambiguous positions in order to minimize the number of mutations required.

Fitch algorithm works as follows. For initialization, it replaces each sequence $S_i$ by its singleton profile $P_i$ (concept introduced in Section 4.2), a sequence of the same length as $S_i$ with its entries defined as follows:

$$P_i[j] = \{S_i[j]\}$$

Now each position of a sequence is represented by a set of possible characters. If the size of the set is greater than 1, the position is an ambiguous position.

Upon a request to find the common parent of two sequences $x$ and $y$, Fitch algorithm assumes that they have the same length $n = |x| = |y|$. The resulting common parent would be a sequence $u$ of length $n$, with entries computed as follows:

$$\forall i = 1, ..., n, u[i] = \begin{cases} x[i] \cup y[i] & \text{if } x[i] \cap x[i] = \emptyset \\ x[i] \cap y[i] & \text{otherwise} \end{cases}$$

Fitch algorithm requires the phylogeny to be known. Therefore, we use the Q-criterion (eq. 5.1) to build the tree bottom-up, and resolve the ambiguity as soon as possible. Each time a new common parent sequence is estimated, we have to compute its distances with other profiles (completing the distance matrix $d$ for use in the Q-criterion).

Given two profiles $x$ and $y$ with ambiguous positions, the pairwise distances $d(x, y)$

is the minimum possible distance among all pairs of sequences $(x', y')$ given the existing ambiguity in $x$ and $y$. For example, consider the following alignment:

```
A C G T T A vs. T T G A T A
  G   A             T
  T
```

The second and fourth position of the first sequence is an ambiguous base. Likewise, the sixth position of the second sequence is an ambiguous base. The alignment between these two sequences would be assigned the same score as the following alignment

```
ATGATA vs. TTGATA
```

A nearby sequence $v$ would not have $d(u, v)$ affected much by this estimate, since it would be the same as setting the common parent to be one found by majority voting in Section 5.4.1. A far away sequence $v$ would have $d(u, v)$ estimation affected heavily. However, such distances should not significantly affect the local structure of the tree near $x$ and $y$, and will be corrected in subsequent iterations of NJ when we proceed to internal nodes closer to $v$.

### 5.4.3 Parsimony method on naive NJ tree

We can also use Fitch algorithm in a different way. First, create a draft phylogeny that is correct near the leaves. Such a tree would be input to Fitch algorithm to find the common parent. The common parent would be used for subsequent iterations of NJ as usual. In one implementation, we pick the draft tree to be the naive NJ output tree (Fig. 5.2), due to the substantial confidence in clustering neighboring sequences.

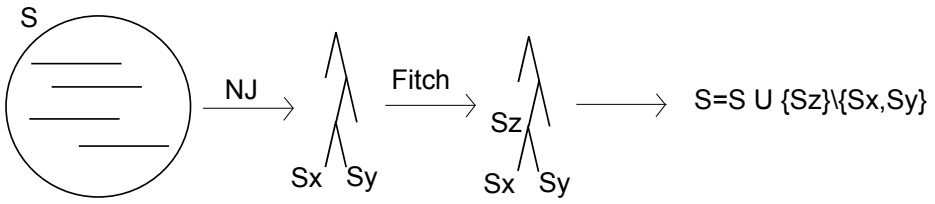



Figure 5.2: Parsimony method on naive NJ tree. First arrow: the first cherry $(S_x, S_y)$ and a draft tree is computed using NJ from pairwise distances. Second arrow: the draft tree is used to estimate the common parent $S_z$ using Fitch algorithm. Third arrow: $S_z$ replaces $S_x$ and $S_y$ in $\mathcal{S}$, and the algorithm repeats from the first step.

### 5.4.4 Perfect NJ method

Since we tried different heuristics to obtain the parent sequence, it makes sense to ask how far can we go with the best possible heuristics. In our simulation testing data, the sequences of the internal nodes are known. Therefore, instead of trying to guess the parent sequence, we can just replace them by the real sequence in the test data if the chosen pair is also a pair in the original data (Fig. 5.3). While this is not really a method to solve Phylogeny Estimation, it lets us gauge the accuracy of methods that try to guess the parent sequence.

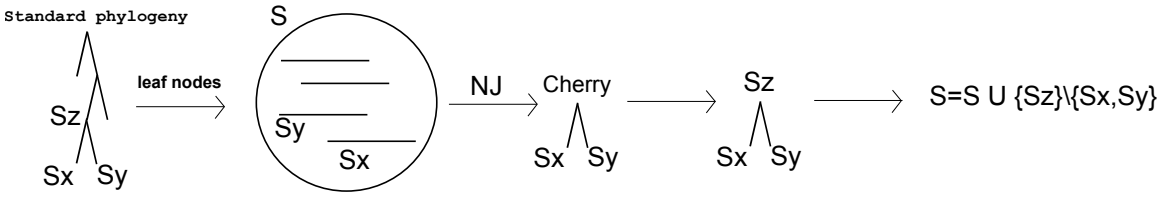



Figure 5.3: Perfect NJ method. First arrow: collect sequences at the leave nodes of a standard phylogeny into $\mathcal{S}$. Second arrow: use the Q-criterion (eq. 5.1) to pick the first cherry $(S_x, S_y)$. Third arrow: if $(S_x, S_y)$ is also a cherry in the standard phylogeny, let $S_z$ be the corresponding parent sequence in the standard phylogeny; otherwise, we obtain $S_z$ using the parsimony method on naive NJ tree. Fourth arrow: replace $S_x$ and $S_y$ by $S_z$, and repeats from the second step.

## 5.5 Evaluation

Phylogenies inferred by different methods can be compared among themselves or to some standard phylogeny by means of the *Robinson-Foulds tree distance* as follows.

A *split* is a partitioning of the set of leaves into two sets of leaves which remain connected after an edge is removed from a tree. If two trees $T_1$ and $T_2$ are equivalent, for each edge in $T_1$ there is a corresponding edge in $T_2$ that produces the same split. The *Robinson-Foulds tree distance* between trees $T_1$ and $T_2$ counts the number of splits in $T_1$ that cannot be found in $T_2$, and those in $T_2$ that cannot be found in $T_1$. Two identical trees would have a distance 0.

We modify this measure to account for the number of sequences by calculating the fraction of correctly inferred splits over the total number of splits in the original phylogeny.

With this modified measure, referred to as *modified RF-measure*, a similarity score ranges from 0 to 1, with a score of 1 indicating that two phylogenies are exactly the same.

We compare different methods by generating different sets of sequences with an accepted or known phylogeny. The sequences either come from either simulation or are actual 16s RNAs. Figures 5.4 and 5.5 indicate that the performances of most NJ variants we introduced are comparable, while the centroid method clearly lags behind. PerfectNJ is not any better than Parsimony, which is a surprising observation. When we compare the performance between simulated and real data, it is clear that the accuracy with real data is much lower. This is due to the simplistic simulation model we used (Chapter 3). When faced with the more complicated real sequence data, the information introduced by ancestor sequences is more valuable, making perfectNJ perform slightly better than the rest.

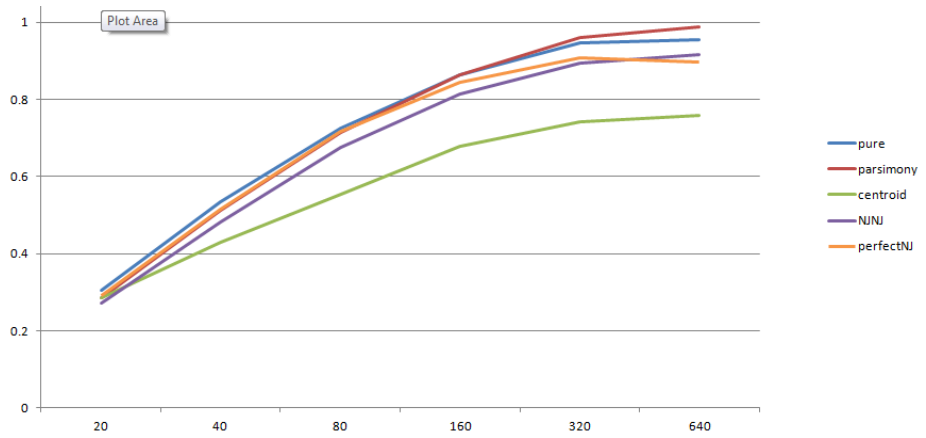



Figure 5.4: *Modified RF-measure* plotted vs. sequence length with different NJ variants; simulated data of 50 sequences, default parameters. The lines corresponds to methods described in previous sections: *pure*: naive NJ, parsimony: Section 5.4.2, *centroid*: Section 5.4.1, *NJNJ*: Section 5.4.3, *perfectNJ*: Section 5.4.4.

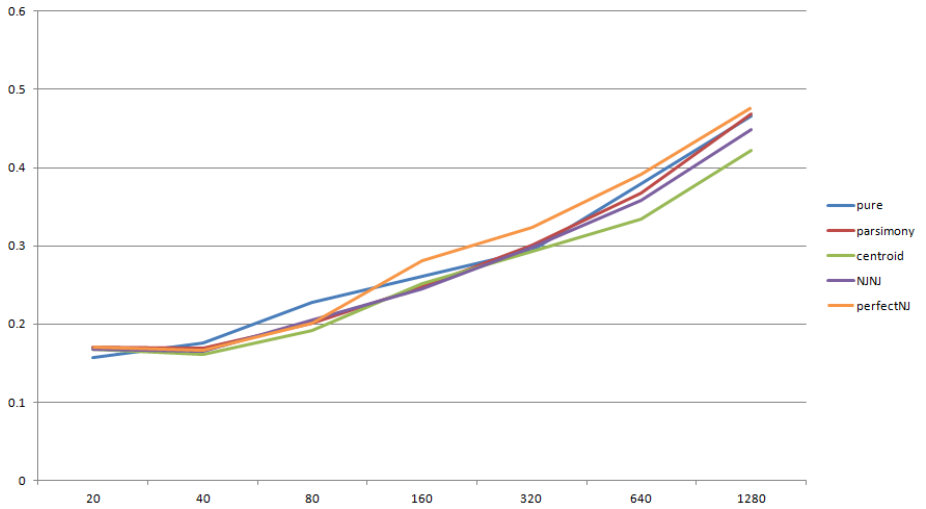



Figure 5.5: *Modified RF-measure* plotted vs. sequence length with different NJ variants; real data with 50 sequences. The lines corresponds to methods described in previous sections: *pure*: naive NJ, parsimony: Section 5.4.2, *centroid*: Section 5.4.1, *NJNJ*: Section 5.4.3, *perfectNJ*: Section 5.4.4.

The Robinson-Foulds metric only uses binary counts on the splits: if split $(X, Y)$ is also found in the new phylogeny with one element off: $(X \setminus \{x\}, Y \cup \{x\})$, the accumulated score is still 0. We decided to try another measure, named *proportional RF-measure* that accounts for such similarities. Denoting the original tree $T_1$, and

the inferred tree $T_2$, the new accuracy measure works as follows.

1. $C = [\,], W = [\,]$
2. Pick the most balanced split $(X, Y)$ in $T_1$, e.g. minimizing $||X| - |Y||$
3. Find the closest split $(X_2, Y_2)$ in $T_2$, e.g. maximizing $|X_2 \cap X| + |Y_2 \cap Y|$
4. Report the score for this split as $c = \frac{|X_2 \cap X| + |Y_2 \cap Y|}{|X| + |Y|}$
5. $C \leftarrow c$,$W \leftarrow |X| + |Y|$
6. $T_1$ and $T_2$ are split into 2 subtrees each according to these splits, and step (2) onwards is performed recursively
7. The overall score of the whole tree is the weighted average of scores in $C$ according to weights in $W$:

$$\frac{\sum_{i=1..|C|} C_i * W_i}{\sum_i W_i}$$

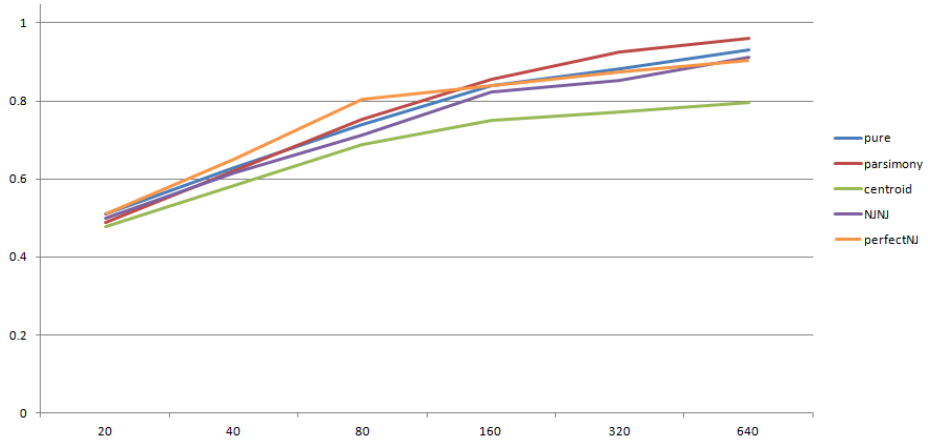



Figure 5.6: *Proportional RF-measure* plotted vs. sequence length with different NJ variants; simulated data with 50 sequences. The lines corresponds to methods described in previous sections: *pure*: naive NJ, parsimony: Section 5.4.2, *centroid*: Section 5.4.1, *NJNJ*: Section 5.4.3, *perfectNJ*: Section 5.4.4.

Figure 5.6 suggests that the *proportional RF-measure* agrees well with the *modified RF-measure*. The same conclusion is highlighted in this case: most NJ variants are comparable, with parsimony performing slightly better, and centroid method still lagging behind.

Given the testing results, we gain more confidence in the parsimony approach (Section 5.4.2). It gives comparable results to the naive Neighbor-Joining that only depends on pseudo-distances, both for simulated and real sequences. Besides, it suggests sequences at the internal nodes of the phylogeny, which is of various benefits. Without those sequences, it is impossible to determine where certain substitutions occur in the phylogeny. Without being able to detect substitutions as events, we cannot use a scoring model that closely resemble the underlying biology of sequences, and have to resort to artificial scoring models such as sum-of-pairs scores instead (Section 2.2.1).

Without the sequences at the internal nodes, the algorithm will remain a black box to users. Even if users want to inspect the result of the naive Neighbor-Joining algorithm, it is hard to see what went wrong. It is hard to relate the distance estimates used in the naive Neighbor-Joining algorithm to the biological events that generated the input sequences.

Lastly, while the parsimony method does not offer significant improvement in previous test cases, there are other modifications to the algorithm that the parsimony method can take advantage of. The parsimony principle is most reliable when the likelihoods of events are low, such that a hypothesis that minimizes the number of events is much more likely to be true. In the current algorithm, we treat all positions in the same way regardless whether they are conserved or volatile. Moreover, we ignore indel events, which have much lower probability than point substitutions. An algorithm that takes into account both of these observations should allow the

parsimony method to improve the accuracy significantly.

# Chapter 6

# Combining multiple sequence alignment with phylogeny inference

Frequently phylogeny inference requires that its input sequences be aligned. On the other hand, multiple alignment algorithms frequently compute guide trees before actually doing alignment. Computing guide trees in turn requires some pairwise alignments to be computed. One may see that multiple alignment and phylogeny inference are two closely related problem, and that the solution of one may relate to the solution of the other. For example, the package MUSCLE [Edgar, 2004] solves this problem by iterating between these two problems (Fig. 6.1).

In Section 5.4 we have discussed variants of the Neighbor Joining algorithm that augment the phylogeny's internal nodes with sequences, rather than being purely a distance-based method. Such approach is strikingly similar to the Progressive Alignment approach in Section 4.2, where a profile is computed at each internal node to summarize the alignment at its subtree. In this Chapter, we will combine the two approaches to construct an algorithm that does multiple sequence alignment and phylogeny inference simultaneously. One natural way to do this is the following

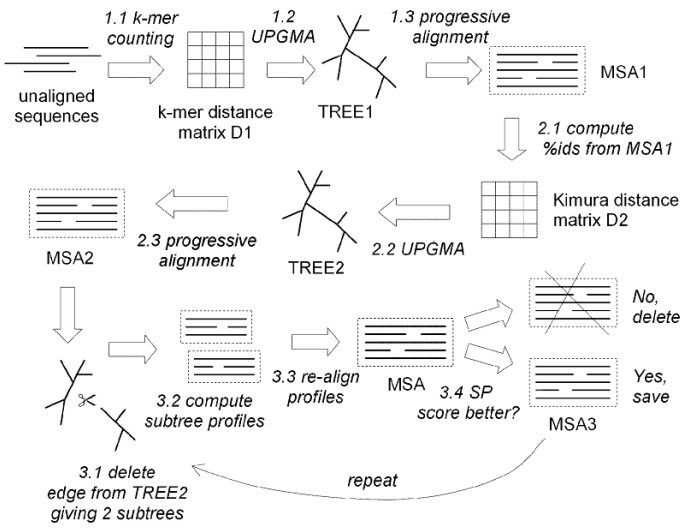



Figure 6.1: MUSCLE [Edgar, 2004] finds distance matrix D1, then phylogeny TREE1, then distance matrix D2 and phylogeny TREE2. TREE2 is used as a guide tree for multiple alignment. The result is iteratively improved.

framework.

Input: a set of sequences $\mathcal{S}$.

1. Replace each sequence $S_i$ in $\mathcal{S}$ by its singleton profile $P_i$ (concept introduced in Section 4.2)

2. While there are more than one profile in $\mathcal{S}$:

   (a) Let $n = |\mathcal{S}|$, number elements of $\mathcal{S}$ arbitrarily as $P_1, ..., P_n$.

   (b) Compute a matrix $d_{n\times n}$ where $d_{i,j}$ is the pairwise distance of profiles $P_i$ and $P_j$.

   (c) Compute a matrix $Q_{n\times n}$ where

$$Q_{i,j} = d_{i,j} - \frac{1}{n-2}(\sum_{k\neq i} d_{i,k} + \sum_{k\neq j} d_{j,k}) \tag{6.1}$$

   (d) Select $P_x, P_y$ with smallest $Q_{x,y}$ and $x \neq y$.

(e) Align $P_x$ and $P_y$ to obtain $P_z$

(f) Remove $P_x$ and $P_y$ and add $P_z$ to $\mathcal{S}$

A close look at this framework suggests that it is the fusion of the framework described in Section 4.2 and the Neighbor-Joining algorithm in Section 5.4.

An implementation of this framework requires a profile representation that can return meaningful scores (approximately tree-additive) for pairwise alignments which are compatible with the Q-criterion of Neighbor Joining (eq. 5.1). In the following sections we present two different profile representations, one more satisfactory than the other.

## 6.1 Generalized Fitch algorithm

### 6.1.1 Singleton Profile

In this method, a profile $P$ is a sequence, such that each element $P[i]$ is the set of possible characters that can be found at position $i$ of $P$. For example, the corresponding profile for "AGCTA" would be $(\{A\}, \{G\}, \{C\}, \{T\}, \{A\})$, and for "GCCTA" would be $(\{G\}, \{C\}, \{C\}, \{T\}, \{A\})$.

### 6.1.2 Profile alignment

Recall that the *parsimony method* (Section 5.4.2) repeats replacing a *cherry* by its estimated common parent sequence. If two profiles $P_1$, $P_2$ of the cherry have equal lengths, their alignment suggests a common parent profile specified by Fitch algorithm:

$$\forall i = 1, ..., |P_1|, P[i] = \begin{cases} P_1[i] \cup P_2[i] & \text{if } P_1[i] \cap P_2[i] = \emptyset \\ P_1[i] \cap P_2[i] & \text{otherwise} \end{cases}$$

Similarly, if $P_1$ and $P_2$ have different lengths, we can align them into $P_1'$ and $P_2'$ with equal length, where $P_1'$ is obtained from $P_1$ and $P_2'$ is obtained from $P_2$ by inserting gaps in between. We can now use the same construction of the common parent.

For example, the following alignment between "AGCTA" and "GCCTA":

```
AGC_TA
_GCCTA
```

would result in the following profile $(\{-, A\}, \{G\}, \{C\}, \{-, C\}, \{T\}, \{A\})$.

Two profiles can be aligned to compute their distance as before. The Needleman-Wunsch algorithm can still be used as long as we can define the distance between two positions of two profiles. Given two ambiguous characters represented by two sets $C_1$ and $C_2$, the distance is 0 if they intersects, and 1 otherwise. The above alignment is assigned a distance of 2, since we need two substitutions to change one sequence into another. For more examples, we have $distance(\{A, G\}, \{A, C\}) = 0$ and $distance(\{T, -\}, \{A, C\}) = 1$.

The new algorithm is sensitive to alignment errors. In particular, if the gap penalty is too high, gap blocks will be collapsed; if the gap penalty is too low, artificial gaps are introduced to better match the sequences. When more gaps are introduced during the simulation, the accuracy of alignment and subsequently phylogeny inference degrades quickly. However, it works well as designed for highly similar input sequences with few gaps (Fig. 6.2).

In the following visualization, each row is an aligned input sequence. Gaps are represented by gray cells. For each column, bases are colored by their counts, from highest to lowest: red, orange, yellow, blue, black. Hence, a column with a blue cell must contain at least 3 different bases. Sequences are generated by simulation with few gaps ($pIns = 0.03, insertSize = 3$).

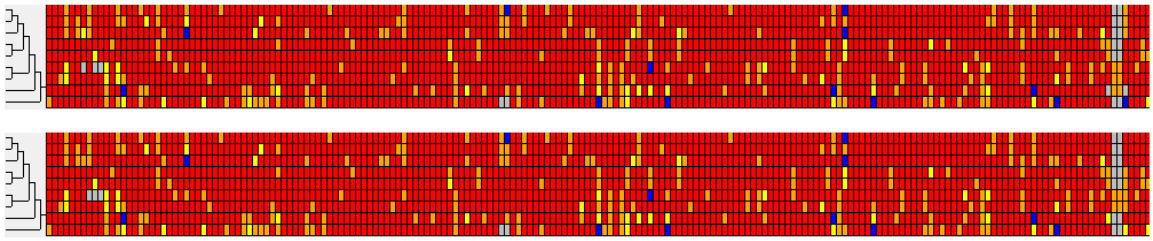


Figure 6.2: Top alignment: result from Generalized Fitch algorithm; bottom alignment: standard alignment from simulation. Note how gaps (gray blocks) are misplaced in the top alignment. Sequences are generated by simulation (Chapter 3) with the following default parameters: $pIns = 0.03, insertSize = 3, n = 200, maxp = 0.1, pSurvive = 0.5, maxp = 0.1$

While the Generalized Fitch algorithm cannot be used for distant sequences with many insertions/deletions, its failure offers one useful insight. The key problem where the algorithm fails is to align characters near gaps. Since we do not implement an affine gap penalty, and since it is not straightforward to extend the affine gap penalty to multiple sequence alignment (Section 2.2.2), stretch of gap characters are often broken into smaller stretches to make room for more base-base matches. This motivates us to employ a more sophisticated approach in the following section.

## 6.2 Maximum parsimony with insertion/deletion events

Most available multiple sequence alignment approaches return outputs in the matrix form (Chapter 4). Such approaches have the following shortcomings:

- Unclear boundaries of gaps may result in wrong alignments (Section 4.5).
- Since the building blocks of gaps are single gap characters, it is hard to track how the same insertion/deletion event appears in different sequences (Section 2.2.2).
- The number of columns grows with the number of input sequences, making the alignment unreadable when there are thousands of sequences being aligned.

To illustrate the third point, here we present a part of the 16S rRNA sequence of Acanthopleuribacter pedis in a multiple alignment with other 2000 rRNA sequences. The full alignment is around 6000bp long, even though each sequence is only 1500bp long on average.

```
>Partial AB303221 rna Acanthopleuribacter pedis Acanthopleuribacteraceae
GG--GG--GA -A-A--C-C- -C-U-G-A-C -G-C-A-GC- A-A--C-GCC -G-C-G-UG- G-G-U-GA-- --U-G-A-A-
G-C-AU---- ---------- ---------- -----CU-UG --GU-G-U-G -UAAA-G-C- CC---UG-UC -G-U--U-AG
G-G-A-CU-- AA--GGA-C- --G-G-U--U -GA----U-- U------AA- ---------- --------G- A--G----UU
---A-AUC-G -UC-U-U-GA -A-G-G-UA- C-CU---G-A -A-G------ A-G----G-A AGC-C-CC-G G-C-UAA-C-
-U-C-C-G-U -G-CCA-G-C -A--G-C--C G-C--GG--U A-AU--AC-- -GG-AG-GGG --GCA-A-G- -C-G--UU-A
U-U-CGG-AA -UU-AC-U-- GG-GC--GU- -AAA-GG-GC -GC--G-UA- G-G-C-G-G- -C-CU-G-G- U-CA-G-U-G
-G--G-A-AG UG--AAA-GC -C-C-UC-GG ---------- ---------- ---------- ---------- ----------
---------- ---------- --CU-C-AA- C-C-G-A-G- G-A--A-U-- A-G--C-U-U --C-C-CA-U A-C-U-G-C-
CA--A-GC-U -A-GA-G-U- -A-U-GG--G A-G-AG-G-G -A--AG-U-- GG-A-AUA-- -U-C-C-G-G U--GU-A-G-
CG-GU--G-- AA-AUG-C-G U-AG--AG-A -UC-G-G-A- U-GG-AAC-A CC-AG--U-- G--GC-GAA- G-G-C--G-A
-C-U-U-C-C UG--G--AC- C-A-U-C-A- C-U--GA-CG --C-U-G-A- UG--C-G-CG -A-AA-G-C- ----------
---G-UGGG- G-AG-C-A-A A-CA--GG-A U-UA-GAUA- C-----C-C- U-G-GUA-G- UC-CA--C-G -CCC-U-AAA
--C-GA-UG- A--A------ CA-C---U-- --------U- U--G--U-G- G-U--A-C-G -G-G------ --UAUC-GAC
C--------C -C-U-G--U- -A-C-U-G-- C-A--GG--A --G-C-U-A- --AC-GC-A- U-UAA-G-U- --G--U-UCC
-GC-C-UG-G G-G-AG-UA- -CG-G----- U-C--G-C-A -A---G-G-- C-U-G-AA-- ---------- ----------
```

To overcome these shortcomings, we design an algorithm that keeps track of how homologous regions evolve among input sequences. This algorithm is developed from

the anchor based approach in Section 4.5.

We now first describe how singleton profiles are generated from single sequences. We then move on to see how profiles are aligned to give distances for use in the Q-criterion.

### 6.2.1 Singleton profile

Given an anchor sequence $S_0$, a sequence $S$ can be searched for homologous regions it shares with $S_0$. To detect indels, we divide homologous regions into gap-free homologous regions (*matches*).

Each match corresponds to an *interval* in $S$, and an *anchor interval* in $S_0$ (Section 4.5). The singleton profile stores the anchor interval and the interval substring of $S$ for each such match.

For example, given the following match:

```
S   10  ACACGAC  16
S0   0  ACAAGAC   6
```

The singleton profile would store the substring $S[10, 16]$ = "ACACGAC" using the format in Section 5.4.2, together with its anchor interval (0,6).

If there are $k$ matches, there would be $k - 1$ gaps between them. The singleton profile would store the lengths of those $k - 1$ gaps. More specifically, a profile stores a set of possible lengths for each gap. In a singleton profile, all such sets have size 1, because there is only exactly one possible length for each gap. When two conflicting gap lengths are aligned in a profile alignment (details in Section 6.2.2), the resulting set of gap lengths is the union of the conflicting sets. In other words, these sets of

gap lengths are used by Fitch algorithm exactly the way sets of characters are used in Section 5.4.2.

In short, a profile consists of three components: a list of strings, a list of indices where those strings are anchored in $S_0$, and a list of possible gap lengths between consecutive matches.

### 6.2.2 Profile alignment

Each profile is best imagined as a set of disjoint intervals (Fig. 6.3) to help intuition.

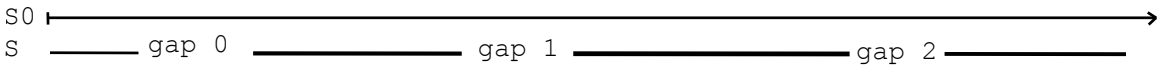



Figure 6.3: The profile of sequence $S$ with anchor sequence $S_0$

The alignment of profiles $P_1$ and $P_2$ needs to take into account their positions in the phylogeny, for reasons which will become apparent later. Let us suppose $P_1$ and $P_2$ are at the root of subtrees $T_1$ and $T_2$, respectively. The alignment consists of the following steps.

1. Find the intersection of the set of intervals of $P_1$ with the set of intervals of $P_2$.

2. For each interval in $P_1$ or $P_2$ that has no intersection with the intersection found in Step 1, consider if we need to keep it in the alignment. Such an interval corresponds to a homologous region in the anchor sequence $S_0$. It is kept in the common parent if and only if that homologous region can be found outside of $T_1$ and $T_2$ (Fig. 6.4).

3. Sort the set of intervals $M$ found in Step 1 and Step 2 ascending by their left index. Note that the intervals are disjoint due to the way we generated them.

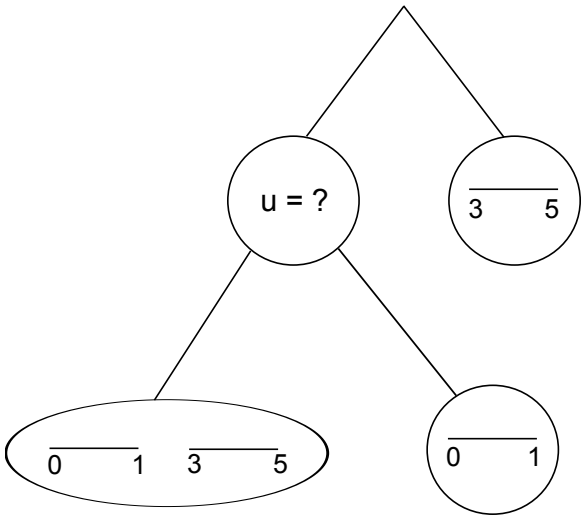



Figure 6.4: Suppose we want to know which intervals exist in the node $u$. From its two leaves, we know it contain interval [0,1], but are not sure if it contain interval [3,5]. However, there is another clue: some other leaves outside this subtree contain interval [3,5]. Because it is unlikely that a deleted sequence would be inserted back, we can conclude that the internal node $u$ should contain interval [3,5].

4. For every pair of consecutive intervals $(M_i, M_{i+1})$, find its set of gap lengths in $P_1$ and $P_2$. For a profile $P_i$, its corresponding set can be empty, if either $M_i$ or $M_{i+1}$ is absent in $P_i$ (Fig. 6.5). The resulting set of gap lengths is found by applying Fitch algorithm over the set of gap lengths in $P_1$ and the set of gap lengths in $P_2$. If those two sets are disjoint, we record that one insertion/deletion event was found.

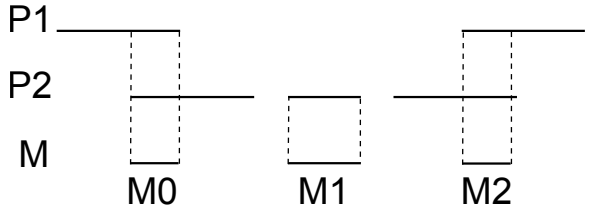



Figure 6.5: When $P_1$ and $P_2$ are aligned, $M$ is the set of intervals found in step 3, which consists of 3 intervals $M_0$, $M_1$, $M_2$. The gap between $M_0$ and $M_1$ does not exist in $P_1$, because $M_1$ does not exist in $P_1$. The set of possible gap lengths in $P_1$ corresponding to the gap between $M_0$ and $M_1$ is thus $\emptyset$.

5. For each interval $M_i$, find its corresponding substring $S_1$ in $P_1$, and $S_2$ in $P_2$. The

resulting substring is combined from $S_1$ and $S_2$ using Fitch algorithm (Section 5.4.2). At the same time, we record the number of substitutions one had to make, which is the number of times we encounter two disjoint sets in Fitch algorithm.

6. Report the profile consisting of the match set $M$, its corresponding strings, and the list of gap lengths. Also report the number of substitutions and indels recorded.

An example run of this alignment algorithm follows.

We have a set $\mathcal{S}$ of 20 input sequences [1], labeled $S_0$ to $S_{19}$ for convenience. The anchor sequence $A$ is randomly selected $A = S_{13}$ (in this example we cannot use the usual notation $S_0$ for the anchor sequence because 0 is a legitimate index for a sequence in $\mathcal{S}$).

First, for each $i$, the sequence $S_i$ is converted into its corresponding singleton profile $P_i$. For example:

$P_{13}$ consists of one match/interval [0,1346], because it is the same as the anchor sequence.

$P_0$ consists of matches to these intervals in $A$: [7,37], [76,103] , [131,235], ... , [753,1007], and [1263,1339]. We calculated the lengths of the gaps, bracketed them and put them in-between their two surrounding intervals in the following compact representation:

7 37 [1] 76 103 [17] 131 235 [-1] 240 358 [2] 380 533 [0] 541 748 [1] 753 1007 [0] 1069 1261 [1] 1263 1339

[1] Accession numbers {AJ012667, AB233332, X59765, AM980986, AY995560, AY859682, DQ442546, AY613990, AB184869, Y17234, DQ888330, DQ062743, AF433173, DQ666683, EU407777, EU376963, AB094401, AJ833000, DQ280368, AY926460}

This means that between interval [7,37] and [76,103] there is a gap of length 1, and so on.

Similarly, $P_1$ is computed to give rise to the following intervals and gap lengths:

16 37 [1] 76 101 [17] 131 358 [1] 366 533 [0] 541 748 [1] 753 928 [2] 944 1046 [2] 1069 1261 [1] 1263 1339

Analogously for $P_2$:

13 50 [11] 163 358 [1] 364 748 [1] 753 1046 [2] 1068 1261 [1] 1264 1339

We calculate the singleton profile for the other 17 sequences. While not shown here, the complete representation of each profile also consists of the characters at each position of the intervals/matches. These bases are used to calculate the number of substitutions among profiles, which is subsequently used to guide the Neighbor-Joining framework.

Once the singleton profiles are calculated, the first few iterations of Neighbor Joining happen as follows:

First, profile $P_0$ and $P_1$ are aligned into profile $P_{20}$:
16 37 [1] 76 101 [17] 131 235 [0, -1] 240 358 [1, 2] 380 533 [0] 541 748 [1] 753 928 [0, 2] 944 1007 [0, 2] 1069 1261 [1] 1263 1339

The first interval of $P_0$ [7,37] intersects with the first interval of $P_1$ [16,37] to give the first interval of $P_{20}$, [16,37]. Because the next gap is of length 1 in both profile, the corresponding gap in $P_{20}$ is also of length 1. The second interval of $P_0$ [76,103] intersects with the second interval of $P_1$ [76,101] to give the second interval of $P_{20}$, [76,101]. The gap after the third interval of $P_0$, [131,235], is not found in $P_1$, therefore the corresponding third gap in $P_{20}$ has two possibilities [0,-1]: we cannot decide yet whether $P_{20}$ contains a gap at that position. Note how the third interval in

$P_1$, [131,358], is broken up into two intervals to align with the third and forth interval in $P_0$, [131,235] and [240,358].

The next iteration of Neighbor-Joining aligns $P_{20}$ with $P_2$ into the new profile $P_{21}$. 16 37 [1] 76 101 [17] 163 235 [0] 240 358 [1] 380 533 [0] 541 748 [1] 753 928 [0] 944 1007 [2] 1069 1261 [1] 1264 1339

The similar procedure is used to obtain the intervals in $P_{21}$. Note that the second interval of $P_{20}$, [76,101], was not found in $P_2$. We have to consider whether this is an insertion from $P_{21}$ to $P_{20}$, or a deletion from $P_{21}$ to $P_2$. In the former case, $P_{21}$ does not contain [76,101], while it would contain the interval in the latter case. Since other sequences outside of the subtree also contain the interval [76,101], such as $P_4$, $P_6$ (data unshown), we know that $P_{21}$ should contain [76,101]: it is very unlikely that the interval was deleted from some ancestor of $P_{21}$, and then inserted back in its child $P_{20}$.

Some of the gap lengths undetermined in $P_{20}$ are now fixed. For example, the third gap in $P_{20}$ with two possible lengths 0 or -1 is now fixed as 0, because the corresponding gap in $P_2$ is of length 0. While we fixed the gap length according to parsimony principle, the biological interpretation is that a deletion of length 1 has happened from $P_{20}$ to $P_0$.

Here we present all 18 iterations of the Neighbor Joining framework, each in the following format

```
First profile index, second profile index, resulted profile index
Indices of leaves in the corresponding subtree
Intervals and gaps in the alignment
```

The first few steps described in the previous section is bold-typed.

**1 0 20**

**(1, 0)**
**16 37 [1] 76 101 [17] 131 235 [0, -1] 240 358 [1, 2] 380 533 [0] 541 748 [1] 753 928 [0, 2] 944 1007 [0, 2] 1069 1261 [1] 1263 1339**

**20 2 21**
**(1, 0, 2)**
**16 37 [1] 76 101 [17] 163 235 [0] 240 358 [1] 380 533 [0] 541 748 [1] 753 928 [0] 944 1007 [2] 1069 1261 [1] 1264 1339**

12 10 22
(12, 10)
13 53 [11, 12] 123 322 [-5, -4] 383 734 [0, -3] 743 749 [0, -5] 751 898 [-3, -2] 943 1036 [4] 1075 1184 [0, -1] 1193 1261 [0, 2] 1263 1266 [0, 2] 1271 1339

22 11 23
(12, 10, 11)
13 53 [11, 12] 123 322 [-4] 383 734 [-3] 745 749 [0] 751 898 [1, -3, -2] 949 1036 [3, 4] 1075 1184 [0] 1193 1261 [2] 1263 1266 [0] 1271 1339 18 17 24 (18, 17) 16 94 [8, 10] 130 324 [0, -1] 351 379 [0, 1] 381 475 [0, -1] 478 740 [0, -2] 753 903 [0] 928 1036 [3] 1069 1261 [1] 1268 1297 [0, -1] 1304 1335

24 19 25
(18, 17, 19)
16 94 [8, 10, 2] 130 324 [0] 351 361 [1] 381 475 [0] 478 527 [0] 544 740 [0] 753 903 [0, -1] 938 1036 [2, 3] 1069 1261 [1] 1268 1297 [0] 1304 1335

16 15 26
(16, 15)
9 111 [8] 145 317 [1] 383 740 [1, -1] 756 901 [0, 2] 947 1034 [4, -3] 1069 1261 [1, 2] 1267 1343

26 25 27
(16, 15, 18, 17, 19)
16 94 [8] 145 317 [1] 383 475 [0] 478 527 [0] 544 740 [0, 1, -1] 756 901 [0] 947 1034 [2, 3, 4, -3] 1069 1261 [1] 1268 1297 [0] 1304 1335

23 13 28
(12, 10, 11, 13)
13 53 [0, 11, 12] 123 322 [0, -4] 383 734 [0, -3] 745 749 [0] 751 898 [0, 1, -3, -2] 949 1036 [0, 3, 4] 1075 1184 [0] 1193 1261 [0, 2] 1263 1266 [0] 1271 1339

28 14 29
(12, 10, 11, 13, 14)
13 45 [0, 27, 11, 12] 123 322 [0, 1, -4] 383 527 [0] 542 734 [0] 745 749 [0] 751 898 [0, 1, -3, -2, 7] 949 1036 [0, 2, 3, 4] 1075 1184 [0] 1193 1261 [2] 1263 1266 [0] 1271 1339

29 27 30
(12, 10, 11, 13, 14, 16, 15, 18, 17, 19)
16 45 [0, 27, 11, 12, 8] 145 317 [1] 383 475 [0] 478 527 [0] 544 734 [0, 1, -1] 756 898 [0] 949 1034 [2, 3, 4] 1075 1184 [0] 1193 1261 [1] 1271 1297 [0] 1304 1335

30 21 31
(12, 10, 11, 13, 14, 16, 15, 18, 17, 19, 1, 0, 2)
16 37 [1] 76 101 [17] 163 235 [0] 240 317 [1] 383 475 [0] 478 527 [0] 544 734 [1] 756 898 [0] 949 1007 [2] 1075 1184 [0] 1193 1261 [1] 1271 1297 [0] 1304 1335

5 4 32
(5, 4)
11 37 [1] 43 94 [9] 130 357 [-19] 383 527 [0] 542 740 [1, 2] 753 903 [-2] 948 1036 [4] 1069 1178 [0, 1] 1184 1264

[2] 1267 1339

32 3 33
(5, 4, 3)
16 37 [0, 1] 43 56 [9, 13] 130 357 [-19] 383 527 [0] 542 740 [2] 753 902 [-3, -2] 948 1036 [4] 1069 1178 [0] 1184 1264 [2] 1271 1336

31 9 34
(12, 10, 11, 13, 14, 16, 15, 18, 17, 19, 1, 0, 2, 9)
16 37 [1] 76 101 [17] 163 175 [0, -1] 181 235 [0] 240 317 [1, -19] 383 475 [0] 478 504 [0, -1] 520 527 [0] 544 734 [1, 2] 756 898 [0, -1] 949 1007 [2, 4] 1075 1184 [0] 1193 1261 [1, 2] 1271 1297 [0] 1304 1335

8 7 35
(8, 7)
8 37 [1] 44 95 [9] 123 367 [-19] 383 487 [0, -1] 489 748 [2] 752 901 [-2] 947 1036 [11, 4] 1069 1262 [0, 2] 1263 1271 [0, -2] 1278 1339

35 6 36
(8, 7, 6)
13 37 [1] 44 93 [9, 10] 129 367 [-15, -19] 383 487 [0] 489 748 [2] 752 901 [-2] 947 1036 [11, 4, 14] 1069 1262 [0, 1, 2] 1268 1271 [0] 1278 1339

36 34 37
(8, 7, 6, 12, 10, 11, 13, 14, 16, 15, 18, 17, 19, 1, 0, 2, 9)
16 37 [1] 76 93 [9, 10, 17] 163 175 [0] 181 235 [0] 240 317 [-19] 383 475 [0] 478 487 [0] 489 504 [0] 520 527 [0] 544 734 [2] 756 898 [0, -1, -2] 949 1007 [4] 1075 1184 [0] 1193 1261 [1, 2] 1271 1271 [0] 1278 1297 [0] 1304 1335

37 33 38

(8, 7, 6, 12, 10, 11, 13, 14, 16, 15, 18, 17, 19, 1, 0, 2, 9, 5, 4, 3)

16 37 [] 163 175 [0] 181 235 [0] 240 317 [-19] 383 475 [0] 478 487 [0] 489 504 [0] 520 527 [0] 544 734 [2] 756

898 [-2] 949 1007 [4] 1075 1178 [0] 1184 1184 [0] 1193 1261 [2] 1271 1271 [0] 1278 1297 [0] 1304 1335

The whole algorithm returns a phylogeny over input sequences, as done in naive Neighbor-Joining algorithm. Besides, we have an important bi-product: we can inspect the result of the algorithm by seeing how gaps evolved. For example, we look at two gaps found in the root node: one between intervals [949,1007] and [1075,1178], and one between intervals [240,317] and [383,475] (Fig. 6.6).

First, let us consider the gap between intervals [949,1007] and [1075,1178] in the root profile $P_{38}$. The algorithm suggests that the gap is of length 4 originally. It is kept intact as the root profile $P_{38}$ evolved into its descendants $P_3$, $P_4$, $P_5$. The algorithm also suggests that the gap is still kept intact in its descendants $P_{37}$, $P_{36}$, $P_{30}$. However,

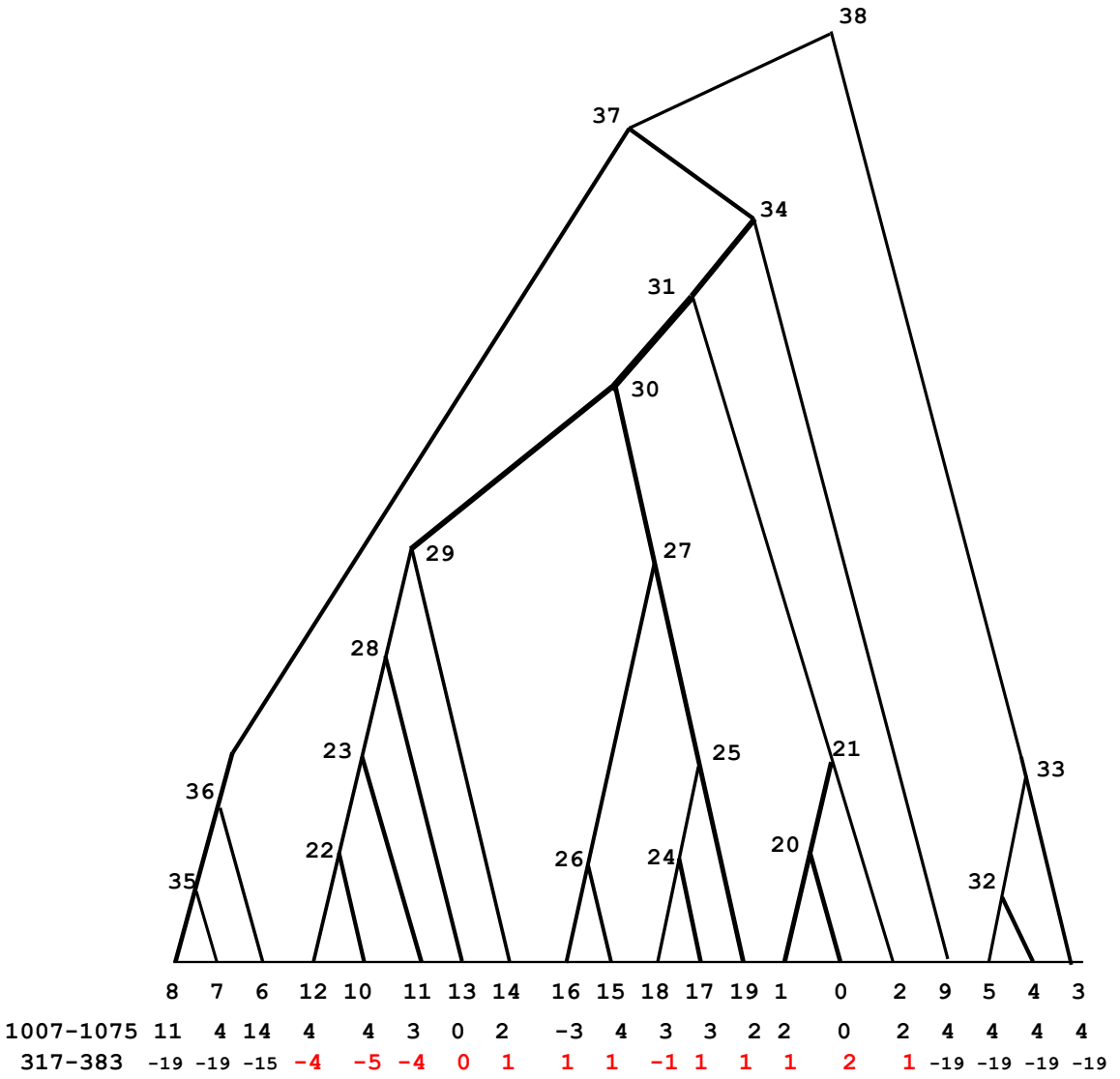



Figure 6.6: Phylogeny found by our algorithm with gaps length found at the leaves. Two gaps were displayed here.

there was a deletion of length 2 from $P_{31}$ to $P_{21}$, so that the corresponding gap at $P_{21}$ is only of length 2 (instead of 4 as its ancestor). We can also tell that there are two independent insertions from $P_{36}$ to $P_6$, and from $P_{35}$ to $P_8$, because they had different gap lengths from each other and from their ancestors (11 and 14 versus 4).

A similar story can be learnt from the gap between intervals [240,317] and [383,475] in the root. This time there are less indel events found. The algorithm suggests that

the gap is of length -19 at the root profile $P_{38}$. It was kept intact in its descendants $P_3$, $P_4$, $P_5$, $P_{34}$ and $P_{36}$. There was an insertion of length 4 from $P_{36}$ to $P_6$, so that the corresponding gap length changed into -15.

One major event happened from $P_{34}$ to $P_{31}$: an insertion of length 20 changes the original gap of length -19 into a gap of length 1. The new gap length is kept intact in many of the descendants, $P_{16}$, $P_{15}$,...,$P_2$, $P_1$. This single event has caused our set of input sequences to separate into two different groups: one with a gap length of about -19 ($P_3$, $P_4$, $P_5$, $P_6$, $P_7$, $P_8$, $P_9$), and one with gap length of about 1 (rest of the leaves, highlighted in Figure 6.6). The detection of this major event suggests that it corresponds to a split between the aforementioned groups of leaves. Suppose an algorithm output a phylogeny that violates this split, it would be disputed by the principle of parsimony, because we have succeeded in finding a hypothesis that use only one event to explain an observation in 20 leaves, which is a more plausible hypothesis.

# Chapter 7

# Conclusions

Multiple sequence alignment and phylogeny inference are important problems that have been studied since the 1980s [Fitch, 1971] [Saitou and Nei, 1987] [Feng and Doolittle, 1987]. They provide indispensable tools for biologists to compare several sequences at the same time. The problems are increasingly important, as DNA sequencing becomes faster and cheaper and more genomes become available. However, these two problems have not been completely solved, and researchers are still investigating them [Liu et al., 2012].

In this thesis, we first surveyed different approaches to each problem:

For multiple alignment, progressive alignment is an approach that divides the problem into steps of pairwise alignments. They rely on the observation that the alignment between similar sequences are more reliable, therefore should be done first. Consistency-based alignment is an approach that enhances pairwise alignments by taking other sequences into account.

For phylogeny inference, there are three main approaches: maximum parsimony methods, maximum likelihood methods, and clustering methods. Maximum parsi-

mony methods and maximum likelihood methods bear significant resemblance, as they both aim at building a tree that maximize some objective scoring. In a loose sense, maximum parsimony approach is a simplified version of maximum likelihood approach where the algorithm uses a simpler evolution model. Distinct from these two, clustering methods instead greedily make use of splits to form partial solutions and proceeds from there.

Our main contribution in this thesis is to combine these seemingly unrelated ideas to provide a biologically relevant view of multiple sequence alignment and phylogeny inference. Our algorithms are able to detect point substitutions/insertions/deletions, and to suggest where these events happen in the phylogeny.

**Substitutions:** To be able to suggest where substitutions happen in the phylogeny, we had to keep track of sequences at internal nodes of the phylogeny. This is guided by the maximum parsimony principle, combined with the Q-criterion (eq. 5.1) to guide the process of picking *cherries* to combine. The algorithm makes use of three ideas: maximum parsimony, Neighbor-Joining algorithm, and progressive alignment.

**Insertions/deletions:** Keeping track of insertions/deletions is actually keeping track of gaps in alignments, or equivalently, keeping track of matches surrounding gaps. While our use of anchor sequence is novel, the idea of utilizing non-gapped local alignments have been used before [Morgenstern et al., 1998]. We also rely heavily on the maximum parsimony principle.

The end result of this thesis is a collection of algorithms, of which the most important one (Section 6.2) does multiple sequence alignment, phylogeny inference, and mutational event detection simultaneously. This algorithm offers various benefits.

- Sequences at internal nodes are estimated. This is useful for evolutionary stud-

ies.

- Mutational events are detected and located at specific edges of the phylogeny. This allows a more biologically relevant scoring model to be used, hence a better way to compare different solutions for multiple sequence alignment and phylogeny inference.

- Most importantly, the construction of the phylogeny from sequences is now open for users to investigate. For example, by looking at how gap lengths are distributed, a researcher can validate a given phylogeny (example in Section 6.2.2, figure 6.6). Without this feature, it is hard for biologists to curate results given by algorithms that often operate on huge matrices of real numbers.

# Further work

While in this thesis we chose a particular implementation, our ideas can extend many current approaches. For example, one can implement a maximum likelihood objective function that takes the substitutions/insertions/deletions detected into account. Alternatively, one may instead use the insertions/deletions detected as candidate splits for clustering methods.

In the scope of our current implementations, there are two important directions for further studies:

- *Estimation of the substitution/insertion/deletion rate at each position of the sequence*: It is important to distinguish conserved positions from volatile positions, so that the algorithms can treat them differently.

- *Construction of good anchor sequences that work with diverse set of input sequences*: Currently, we use a random sequence from the input set as the anchor

sequence. Distant sequences will have fewer matches; this in turn leads to degrading accuracy in constructing phylogeny at distant sequences. One possible fix to this problem is to concatenate substrings from different distant sequences to obtain the anchor sequence. Such an anchor sequence should give higher coverage when searched against most input sequences.

# Bibliography

[Atteson, 1999] Atteson, K. (1999). The performance of neighbor-joining methods of phylogenetic reconstruction. *Algorithmica*, (25).

[Bryant, 2005] Bryant, D. (2005). On the uniqueness of the selection criterion in neighbor-joining. *Journal of Classication*, 22(1).

[Do et al., 2005] Do, C. B., Mahabhashyam, M. S. P., Brudno, M., and Batzoglou, S. (2005). ProbCons: Probabilistic consistency-based multiple sequence alignment. *Genome Research*, 15(2):330–340.

[Edgar, 2004] Edgar, R. C. (2004). MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5):1792–1797.

[Felsenstein, 2003] Felsenstein, J. (2003). *Inferring Phylogenies*. Sinauer Associates, 2 edition.

[Feng and Doolittle, 1987] Feng, D.-F. and Doolittle, R. (1987). Progressive sequence alignment as a prerequisitetto correct phylogenetic trees. *Journal of Molecular Evolution*, 25:351–360.

[Fitch, 1971] Fitch, W. M. (1971). Toward Defining the Course of Evolution: Minimum Change for a Specific Tree Topology. *Systematic Zoology*, 20(4):406–416.

[Gascuel and Steel, 2006] Gascuel, O. and Steel, M. (2006). Neighbor-Joining Revealed. *Molecular Biology and Evolution*, 23(11):1997–2000.

[Jukes, 1969] Jukes, T. H. (1969). Evolution of protein molecules. *Manmmalian Protein Metabolism*, pages 21–132.

[Lee et al., 2002] Lee, C., Grasso, C., and Sharlow, M. F. (2002). Multiple sequence alignment using partial order graphs. *Bioinformatics*, 18(3):452–464.

[Lipman et al., 1989] Lipman, D. J., Altschul, S. F., and Kececioglu, J. D. (1989). A tool for multiple sequence alignment. *Proceedings of the National Academy of Sciences*, 86(12):4412–4415.

[Liu et al., 2012] Liu, K., Warnow, T. J., Holder, M. T., Nelesen, S. M., Yu, J., Stamatakis, A. P., and Linder, C. R. (2012). SATé-II: Very Fast and Accurate Simultaneous Estimation of Multiple Sequence Alignments and Phylogenetic Trees. *Systematic Biology*, 61(1):90–106.

[Löytynoja and Goldman, 2005] Löytynoja, A. and Goldman, N. (2005). An algorithm for progressive multiple alignment of sequences with insertions. *Proceedings of the National Academy of Sciences of the United States of America*, 102(30):10557–10562.

[Maddison, 2007] Maddison (2007). The Tree of Life Web Project.

[Morgenstern et al., 1998] Morgenstern, B., Frech, K., Dress, A., and Werner, T. (1998). DIALIGN: finding local similarities by multiple sequence alignment. *Bioinformatics*, 14(3):290–294.

[Munoz et al., 2011] Munoz, R., Yarza, P., Ludwig, W., Euzéby, J., Amann, R., Schleifer, K.-H., Glöckner, F. O., and Rosselló-Móra, R. (2011). Release LTPs104 of the All-Species Living Tree. *Systematic and Applied Microbiology*, 34(3):169–170.

[Notredame et al., 2000] Notredame, C., Higgins, D. G., and Heringa, J. (2000). T-coffee: a novel method for fast and accurate multiple sequence alignment. *Journal of Molecular Biology*, 302(1):205–217.

[Radu Mihaescu, 2007] Radu Mihaescu, L. P. (2007). Why Neighbor-Joining Works. *ALGORITHMICA*, 54(1).

[Report, 2010] Report (2010). IUCN Red List of Threatened Species 2010.

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

[Smith and Waterman, 1981] Smith, T. F. and Waterman, M. S. (1981). Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197.

[Studier and Keppler, 1988] Studier, J. A. and Keppler, K. J. (1988). A note on the neighbor-joining method of Saitou and Nei. *Molecular Biology and Evolution 5*.

[Tamura et al., 2004] Tamura, K., Nei, M., and Kumar, S. (2004). Prospects for inferring very large phylogenies by using the neighbor-joining method. *Proceedings of the National Academy of Sciences of the United States of America*, 101(30):11030–11035.

[Thompson et al., 1994] Thompson, J. D., Higgins, D. G., and Gibson, T. J. (1994). CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Research*, 22(22):4673–4680.