

The Limits of Progressive Multiple Sequence Alignment

A Dissertation

Presented in Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

With a Major in Bioinformatics and Computational Biology

In the College of Graduate Studies

University of Idaho

by

Lucas James Sheneman

August 2008

Major Professor: James A. Foster, Ph.D.

UMI Number: 3347530

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI®

UMI Microform 3347530

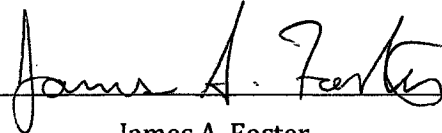
Copyright 2009 by ProQuest LLC.


All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

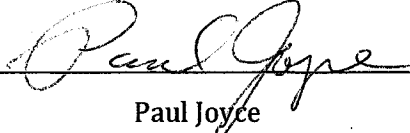
ProQuest LLC
789 E. Eisenhower Parkway
PO Box 1346
Ann Arbor, MI 48106-1346


AUTHORIZATION TO SUBMIT DISSERTATION

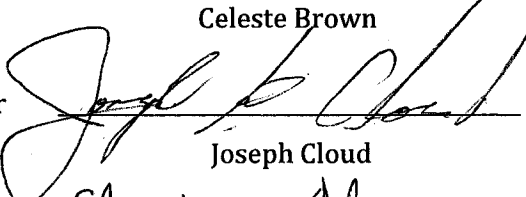
This dissertation of Lucas J. Sheneman, submitted for the degree of Doctor of Philosophy with a major in Bioinformatics and Computation Biology and titled "The Limits of Progressive Multiple Sequence Alignment," has been reviewed in final form. Permission, as indicated by the signatures and dates given below, is now granted to submit final copies to the College Of Graduate Studies for approval.

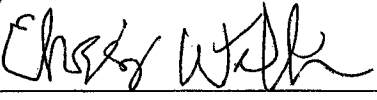
Major Professor  Date 14 Nov 08
James A. Foster


Committee Members  Date 11/13/08
Holly Wichman

 Date 11/13/2008
Paul Joyce

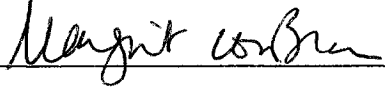
 Date 11/13/08
Celeste Brown

Department Administrator  Date 13 Nov 08
Joseph Cloud

Director of BCB Program  Date 11/13/08
Chris Williams

Discipline's College Dean  Date 11/14/08
Scott Wood

Final Approval and Acceptance by the College of Graduate Studies

 Date 11/21/08
Margrit Von Braun

Abstract

Biologists use progressive multiple sequence alignment to identify positional homology in regions of molecular sequences. The popularity of this method is due to the pragmatic trade-off between computational efficiency and accuracy. However, progressive alignment has several inherent limitations. This thesis assesses the underlying causes of these limitations and presents novel methodology for improving existing alignment algorithms.

In progressive sequence alignment, a guide tree is inferred and then sequences are pairwise aligned via dynamic programming in the order dictated by the tree. This results in an alignment representing the reconstructed positional homology of the input sequences. This thesis explores the relationship between guide tree topology and alignment accuracy.

We present two distinct genetic algorithms, both of which optimize a population of guide tree topologies using stochastic crossover and mutation operators. One genetic algorithm, EVALYN, achieves high accuracy scores when measured against published benchmark tests. However, we find that EVALYN's destructive crossover reduces the genetic algorithm to a stochastic hill climb.

Most progressive alignment programs infer guide trees using the Neighbor-Joining algorithm. The $O(N^3)$ time complexity of Neighbor-Joining makes it a bottleneck when aligning large datasets. The Relaxed Neighbor-Joining algorithm relaxes the requirements for joining tree nodes, thereby reducing the typical time complexity to $O(N^2 \log N)$ with no significant qualitative effects. This thesis describes Clearcut, the reference implementation for the Relaxed Neighbor-Joining algorithm. Results show that Clearcut dramatically outperforms existing Neighbor-Joining implementations.

Due to arithmetic ties encountered during the backtracking phase of dynamic programming, a backtracking path is a directed acyclic graph. Each path in the graph results in a different final alignment. The set of distinct alignments has Gaussian distributed accuracy scores. We introduce an unbiased backtracking algorithm and demonstrate how biased backtracking leads to suboptimal alignments.

We explore the correlation between guide tree quality and alignment quality by defining quality in terms of distance from the true alignment or tree and then systematically degrading guide trees via tree edit operators. There is a statistically significant correlation between guide tree quality and alignment quality, although the measured effect of guide tree selection on alignment quality is small.

Acknowledgements

I deeply appreciate my adviser, Dr. James A. Foster, for his nearly limitless patience, understanding, and optimism during my years at the University of Idaho. I wish to thank Jenny, my wife, for her understanding and sacrifices, especially when I was in the middle of intense coursework and we had babies at home. Appreciation goes out to Jason Evans, fellow student and friend for his help and guidance, especially in our joint work around Relaxed Neighbor-Joining. I thank Lynne McCreight for important logistical family support during the final stretch of this dissertation. Finally, I wish to express gratitude to my committee and all other BCB faculty, staff, and students for their help and ideas over the years.

Sheneman was partially funded by NIH P20 RR16454 from the INBRE Program of the National Center for Research Resources. Evans and Foster were partially funded by NIH NCRR 1P20 RR16448. Some of the experiments were run on the University of Idaho IBEST Beowulf cluster, which is funded in part by NSF EPS 00809035, NIH NCRR 1P20 RR16448, and NIH NCRR 1P20 RR16454.

Dedication

I dedicate this dissertation to my family...

to Dad, for buying me that Vic-20 when I was 8 years old
and teaching me how to program;
to Mom, for always supporting my interests and activities;
to Kelly, for egging me on to finish this work;
to Jenny, for always showing her love by trying hard to support
my crazy ideas;
to my daughter Annie, for our precious talks;
and to little Linnea, who makes everyday so joyous.

Contents

Authorization to Submit Dissertation.....	ii
Abstract.....	iii
Acknowledgements.....	iv
Dedication	v
List of Tables	ix
List of Figures.....	x
1. Evolving Guide Trees in Progressive Multiple Sequence Alignment	1
1.1. Preface	1
1.2. Abstract	1
1.3. Introduction.....	2
1.4. Progressive Alignment Background	3
1.5. Algorithm Implementation	5
1.5.1. Guide Tree Encoding with the Binary Coalescent	6
1.5.2. Evaluating Guide Tree Fitness	7
1.5.3. Selection, Crossover, and Mutation	9
1.5.4. Algorithm Parameters	11
1.6. Experimental Results.....	13
1.7. Conclusions.....	15
1.8. Future Work.....	15
2. Evolving Better Multiple Sequence Alignments	17
2.1. Preface	17
2.2. Abstract	17
2.3. Introduction.....	18
2.4. Previous Work.....	20
2.5. Algorithm Implementation	21
2.5.1. Crossover and Mutation Operators	22
2.5.2. Measuring Guide Tree and Alignment Fitness.....	23

2.6. Algorithm Analysis	24
2.7. Experimental Setup and Results.....	25
2.8. Conclusions.....	28
2.9. Future Work	29
3. Clearcut: A Fast Implementation of Relaxed Neighbor-Joining.....	31
3.1. Preface	31
3.2. Abstract	31
3.3. Background.....	32
3.4. Implementation	32
3.5. Performance	33
3.6. Future Enhancements	34
4. Estimating the Destructiveness of Crossover on Binary Tree Representations	35
4.1. Preface	35
4.2. Abstract	35
4.3. Introduction.....	36
4.4. Methods.....	38
4.5. Results.....	39
4.6. Discussion.....	41
5. The Effects of Dynamic Programming Bias on Progressive Multiple Sequence Alignment	43
5.1. Preface	43
5.2. Abstract	43
5.3. Introduction.....	44
5.3.1. Dynamic Programming Ties	46
5.3.2. Multiple Sequence Alignment	46
5.4. The Frequency of Ties.....	47
5.5. Removing Bias from Dynamic Programming.....	49
5.6. One Guide Tree, Many Alignments	52
5.7. A Progressive Alignment and Dynamic Programming Counterexample	54
5.8. Discussion.....	56

6. Quantifying The Correlation of Alignment Accuracy and Guide Tree Quality	58
6.1. Preface	58
6.2. Abstract	58
6.3. Introduction.....	59
6.4. Methods.....	60
6.4.1.Degrading Guide Trees.....	62
6.4.2.Experimental input data.....	63
6.4.3.TreeBASE Alignments and Trees.....	64
6.4.4.Simulated Sequences and Trees.....	65
6.4.5.Measuring Alignment quality	65
6.5. Results.....	66
6.5.1.Significance of Correlation.....	73
6.6. Discussion.....	74
6.7. Future Work	76
A. Relaxed Neighbor-Joining: A Fast Distance-Based Phylogenetic Tree Construction Method	77
A.1 Preface	77
A.2 Abstract	77
A.3 Introduction	78
A.4 Experiment	82
A.4.1 Correctness	84
A.4.2 Speed.....	84
A.4.3 Quality	84
A.5 Discussion	88
Bibliography	90

List of Tables

1.1 Time complexity for three kinds of dynamic programming alignments	8
1.2 Concise Summary of Experimental GA Parameters	12
1.3 The GA Alignment Scoring System with Affine Gap Penalties	12
2.1 Experimental settings for EVALYN	26
6.1: The selected TreeBASE datasets	64

List of Figures

1.1 The traditional progressive alignment algorithm.....	3
1.2 A coalescing binary tree of 8 sequences.....	6
1.3 The Process of Fitness Evaluation	8
1.4 The beta probability distribution used for selection	9
1.5 The crossover of two compatible coalescing binary trees	10
1.6 The fitness trend of the fittest individual across 10,000 iterations	13
1.7 The fitness curves demonstrated by the GA.....	14
1.8 Visual comparison of the resulting alignments.....	14
2.1 Example of a multiple sequence alignment of three DNA sequences.....	19
2.2 Computing pairwise edit distances to construct a guide tree.....	20
2.3 Crossover as a three-step process	22
2.4 Evaluating the fitness of a guide tree.....	23
2.5 A sum-of-pairs score is computed for a simple pairwise alignment.....	24
2.6 Comparing CLUSTAL W and EVALYN with default parameters	27
2.7 Comparing CLUSTAL W and EVALYN with identical parameters.....	28
3.1 Neighbor-Joining implementation speed tests.....	34
4.1 Lewis crossover with two parent trees	37
4.2 Computing normalized Robinson-Foulds distances	37
4.3 Example pair of distributions of Lewis crossover destructiveness.....	40
4.4 The distribution of Robinson-Foulds distances	41
5.1 The Needleman-Wunsch dynamic programming matrix	45
5.2 Arithmetic tie frequency along the directed graph during backtracking	48
5.3 Counting ties in DP matrix as function of sequence similarity.....	49
5.4 Extending dynamic programming: unbiased, stochastic backtracking.....	51
5.5 Example alignment score distributions, given a single guide tree	53

5.6 PMSA counterexample: Optimal DP leading to suboptimal progressive alignments	55
6.1 Naturally evolved M2045 fungal 18S rRNA TreeBASE study results	66
6.2 Results of simulations based on M2045 fungal 18S rRNA TreeBASE study	67
6.3 Results for the natural M3016 rbcL Chloroplast study	68
6.4 Results for the simulated M3016 rbcL Chloroplast study	69
6.5 Results using the shorter 5.8S rRNA M2783 fungal pathogen study (natural)	70
6.6 Results using the shorter 5.8S rRNA M2783 fungal pathogen study (simulated)	71
6.7 The largest measured effect for each dataset for both SP-error and TC-error	72
6.8 The Spearman rank coefficient for alignment quality and tree degradation	74
A.1 Phylogenetic tree and corresponding patristic distance matrix	80
A.2 A pectinate tree with $x = 4$ leaf taxa	81
A.3 A perfect tree with 2^{x-2} leaf taxa	82
A.4 Speed comparison of RNJ/NJ tree construction for two tree shapes	83
A.5 RNJ quality results (mean MSE of Robinson-Foulds distances) for four tree shapes	85
A.6 Quality results of RNJ versus NJ, for all four tree shapes	87

Chapter 1

Evolving Guide Trees in Progressive Multiple Sequence Alignment

- [1] Shyu, C., L. Sheneman, J.A. Foster (2004) Multiple Sequence Alignment with Evolutionary Computation, *Genetic Programming and Evolvable Machines special issue on Biological Applications of Evolutionary Computation*, (5)2:121-144.

1.1 Preface

This paper was incorporated into a survey article published in 2004 in the Journal of Genetic Programming and Evolvable Machines.¹ The published survey article describes applying evolutionary computation (EC) to the problem of multiple sequence alignment (MSA). The article provided an overview of both the background and the current state-of-the-art. As such, the article describes two previously unpublished techniques, including the use of the *binary coalescent* representation that is explained in this chapter.

Large portions of this chapter appear in a slightly re-formatted, but otherwise unchanged form in the published survey article.

1.2 Abstract

We present a novel application of genetic algorithms to the problem of aligning multiple biological sequences through the optimization of guide trees. Individual guide trees are represented as coalescing binary trees that provide for efficient and meaningful crossover and mutation operations. We hypothesize that our technique avoids the limitations of other

¹ Special issue on Biological Applications of Evolutionary Computation.

heuristic tree-building techniques, and is therefore capable of producing better trees, as measured by higher quality alignments. Further, our approach is more scalable than commonly used progressive alignment techniques when aligning large datasets.

1.3 Introduction

Determining the optimal alignment of more than a handful of sequences has a prohibitive time complexity. Because of this, various heuristic approaches have been developed, many of which are capable of producing good alignments in a relatively short period of time. The most commonly used heuristic technique is known as progressive multiple sequence alignment (PMSA). The most notable implementation of this technique is the widely used CLUSTAL W [2] program.

Although most progressive alignment systems such as CLUSTAL W provide a fast, heuristic approach to multiple sequence alignment, CLUSTAL W itself performs a large number of optimal or near-optimal pairwise alignments, which can take a prohibitively long time to compute. Because of this, CLUSTAL W and other similar tools are unable to scale well beyond the alignment of a few thousand sequences, or beyond dozens of very long sequences.

We present a technique for evolving guide trees using genetic algorithms (GA). As will be described in additional detail, most progressive alignment approaches such as those used by CLUSTAL W rely on the heuristic construction of a guide tree to specify the order of pairwise alignments that result in a full multiple sequence alignment. By contrast, our approach iteratively optimizes a population of guide trees using a genetic algorithm and evaluates the fitness of each guide tree by performing a progressive multiple sequence alignment in the order specified by each guide tree. This avoids the relatively expensive steps of finding the optimal edit distances between all pairs of the sequences and then computing a guide tree based on these edit distances.

In addition, ties can occur in progressive alignment in which several combinations of pairwise distance measurements may result in the same distance score. Thus, multiple guide trees could theoretically result from the comparison of the same set of input sequences. Most systems arbitrarily break this tie in a deterministic fashion, but this avoids the construction of some guide trees that might ultimately produce more optimal alignments. Due to our application of a

GA to directly construct and evolve guide trees, we avoid this small but notable limitation of progressive alignment techniques altogether.

1.4 Progressive Alignment Background

Traditional progressive multiple sequence alignment involves at least a three-step process in which input sequences are first compared to one another using dynamic programming (DP) [3, 4] to estimate the edit distances between all possible pairs of sequences. The use of DP for computing pairwise distances guarantees an optimal result for the pairwise comparisons, but has time complexity of $O(n^2)$ for comparing just two sequences. For n input sequences, there are $\binom{n}{2}$ pairwise distances.

Notably, to counter the obvious scalability issues of performing so many optimal pairwise alignments, systems such as CLUSTAL W offer the option of using faster, less-accurate forms of pairwise distance measurements, but this ultimately results in the construction of less accurate guide trees, which can have a deleterious impact on the overall quality of the entire multiple sequence alignment.

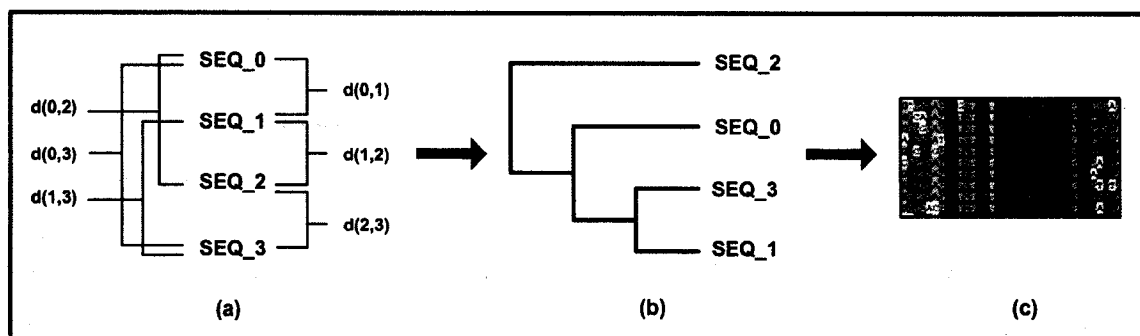


Figure 1.1: The traditional progressive alignment algorithm. (a) All possible pairs of sequences are optimally aligned using dynamic programming to determine their edit distance. Then, (b) edit distance information is used by a Neighbor-Joining algorithm to estimate and construct a guide tree. (c) Finally, the sequences are progressively aligned using the guide tree in order to produce an alignment.

After all pairwise distances have been computed, the distances are used to construct a guide tree using techniques such as Neighbor-Joining (NJ) [5]. The process of constructing a guide

tree based on pairwise distances is simple and reasonably scalable, but it is a subject to certain limitations. NJ is a simplistic iterative clustering algorithm which is based on the approach of using pairwise edit distance information to decompose an initial star-shaped tree into a full tree which represents, based on pairwise sequence distances, the phylogenetic relationships between all of the taxa on the tree. In such a tree, the most similar sequences are clustered together first, followed by the most similar sub-alignments, and so on. Eventually, an entire tree is built which represents the similarity relationships between all of the sequences. In progressive multiple sequence alignment, the tree built by processes such as Neighbor-Joining are used as the guide that ultimately describes an order of operations of aligning sequences and sub-alignments.

This traditional progressive alignment approach is, in general, a sound one. It is relatively fast and often produces both good guide trees and, more importantly, reasonably good alignments. The largest single problem with the progressive alignment technique is that it is susceptible to convergence at local optima since in each progressive alignment, new information is unable to be completely incorporated into the overall multiple alignment. For example, when aligning a single sequence with an alignment, the sequence is not aligned directly and optimally with every internal sequence within the alignment. Instead, the sequence is aligned with the alignment as a whole. Gaps in the resulting alignment are inserted into the sequence, or into the alignment as a whole, but are not optimally distributed throughout the resultant alignment. It is this form of gradient descent which prevents progressive alignment techniques from fully incorporating new information at each stage of a progressive alignment, and which is a major cause for error and variance from the true optimal alignment. The accuracy of the guide tree is therefore critical. By building a guide tree based on edit distances between sequences, and thus first aligning sequences with the highest similarity, the information gradient associated with each step of the alignment is minimized, as is the number of gaps in the overall alignment.

A great deal of time in a progressive alignment is spent in constructing a matrix of edit distances that are then used to build a guide tree via Neighbor-Joining. As mentioned before, there are $\binom{n}{2}$ pairwise sequence alignments necessary to build the distance matrix, where n is the number of sequences being aligned. Each pairwise sequence alignment has a time complexity of $O(n^2)$, where n is the average length of the two sequences being aligned. The time complexity of the Neighbor-Joining algorithm is also $O(n^2)$, where n is the number of sequences being joined into a guide tree. The time complexity for simply constructing a guide tree via

progressive alignment is $O(m^2n^2)$ where m is the length of the sequences being aligned, and n is the number of sequences being aligned.

It is our central hypothesis that by avoiding deterministic guide tree construction, the time necessary to directly compute a guide tree can be entirely avoided. We hypothesize that for very large datasets, our approach will scale demonstrably better than traditional progressive alignment techniques, while simultaneously resulting in higher quality multiple alignments.

1.5 Algorithm Implementation

The algorithm implements an iterative steady-state genetic algorithm. The population in the GA consists of guide trees that are represented in an efficient, coalescing binary tree data structure that enables fast and meaningful crossover and mutation. Variability operators such as crossover and mutation are constructed such that the modified tree always remains a properly formed PMSA guide tree. Rank-based selection is implemented via the use of a random number generator that samples from a carefully parameterized beta probability distribution. This non-uniform random selection, when overlaid across a sorted table of fitness scores for all individuals in the population, allows for strongly biased rank-based selection wherein highly fit parents are far more likely to be selected for crossover, and whose offspring replace low-fit individuals on the opposite end of the distribution. Elitism is implemented, as the fittest individual in a population is never destroyed by less-fit offspring. Fitness for any individual is objectively computed by performing the progressive alignment in the pairwise order specified by the individual guide tree. The fitness of an individual tree is computed as the log of the alignment score of the final alignment produced by performing the progressive alignment in the order specified by that tree. In this way, a guide tree is optimized *only* with respect to the most important measurement: the intrinsic quality of the final multiple sequence alignment. Because of this, extraneous optimality criteria and sources of possible errors (such as misleading Neighbor-Joining trees) are ignored as the GA focuses only on maximizing progressive alignment scores by evolving successively better guide trees.

1.5.1 Guide Tree Encoding with the Binary Coalescent

We present a novel chromosome encoding for the individual guide trees in our GA. The encoding is extremely efficient in the contexts of both space and time and allows for the application of fast and meaningful crossover and mutation operators. One of the most important aspects of our chromosome encoding is that it avoids the problem of dealing with duplicate leaves during branch swapping.

Each individual in the population represents a possible guide tree, and is stored as an integer vector describing how nodes on one level of a coalescing tree connect to the next level of the same coalescing tree. At the lowest level of the tree, level 0, there are n terminal nodes, where n is the number of sequences being aligned. Each terminal node corresponds to a particular sequence from the n sequences being aligned. The ordering of the terminal nodes is static. At the next level, there are $n-1$ nodes to which each of the terminal nodes may connect. This forces at least one coalescence at level 1. In general, each level of the coalescing tree has $n-x$ nodes, where n is the number of sequences being aligned, and x is the level of the tree. A binary coalescing tree with n terminal nodes has n levels, and is therefore triangular in shape and requires at most the following number of nodes:

$$\text{Number of Nodes} = \frac{n(n+1)}{2}$$

where n is the number of leaf nodes on the tree.

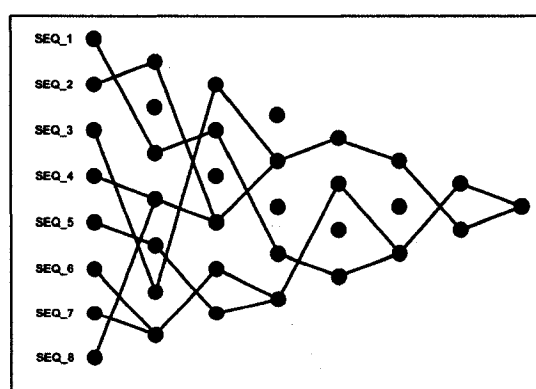


Figure 1.2: A coalescing binary tree of 8 sequences. Note that full coalescence occurs at an upper bound of n steps, but can often occur sooner.

Since a node is little more than the description of the edge from a given node to another node at a subsequent level, these nodes (and therefore the tree itself) can be represented as an integer vector. If the nodes of each column in such a tree are numbered from 0 through $n-x$, where n is the number of leaf nodes and x is the column index, then the tree shown above in Figure 1.2 can be efficiently represented as the string:

2,0,5,3,4,7,7,3,3,(-1),1,3,5,0,4,1,3,(-1),1,4,4,(-1),0,(-1),3,1,0,2,(-1),2,1,(-1),0,0,0

In this encoding, each value represents a description of the edge from a node at level x in the coalescing tree to another node at level $x+1$. The value -1 is used to represent edgeless nodes.

Since we are essentially evolving a phylogenetic tree, we add the constraint that any one node can have no more than two connections from the left. To efficiently enforce this constraint, at each node we also track the number of connections from the previous level. For a binary coalescing tree, these values are 0, 1, or 2. By examining the number of left and right connections at each node, it is straightforward to quickly confirm the validity of a given tree.

1.5.2 Evaluating Guide Tree Fitness

The initial population of trees in our GA consists of some number of randomly generated trees. These trees are built in a bottom-up fashion in a completely random walk up to the root of the tree. For each node at given level of a tree, a node in a subsequent level is chosen entirely at random, constrained only by the limitation that nodes at level x are allowed a maximum of two connections from level $x-1$. Completely viable, random trees can be built very quickly using this approach.

Tree fitness is computed for each individual in the initial random population as well as for each offspring that results from crossover/mutation operations. Fitness is computed by first building an intermediate *evaluation tree* which is a temporary data structure used to hold the sequences and partial alignments as the progressive alignment is computed by a post-order traversal of the evaluation tree. At each node in the evaluation tree, the fitness function recursively descends, and then performs an alignment. An alignment can occur between a pair of sequences, between a single sequence and a partial alignment, or between two partial alignments. In this way, the complete progressive alignment is built up until the root node of the evaluation tree contains the complete alignment and the score for that alignment. The

natural log of this alignment score is then computed and represents the objective fitness for the guide tree.

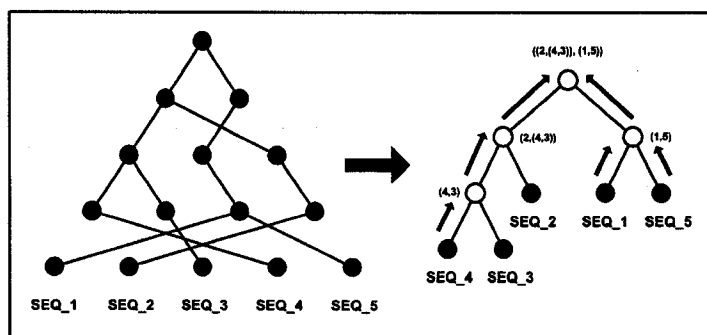


Figure 1.3: The Process of Fitness Evaluation. The coalescing binary tree is first converted to an evaluation tree, and then a progressive alignment is performed via a post-order traversal of the evaluation tree in which sequences and partial alignments are progressively aligned into a complete alignment of all of the input sequences.

Fitness evaluation is the most computationally time-consuming component of this genetic algorithm, especially towards the top of the evaluation tree, where large partial alignments are themselves being aligned. Specifically, the time complexity of computing alignments is given in Table 1.1.

Table 1.1: Time complexity for three kinds of dynamic programming alignments [6].

Type of Alignment	Time Complexity
Sequence + Sequence	$O(mn)$ where m and n are the lengths of the sequences being aligned
Sequence + Alignment	$O(kn + \min\{s, k\}mn)$ where m is the length of the sequence, n is the length of the alignment, k is the number of sequences represented comprising the alignment, and s is the size of the alphabet
Alignment + Alignment	$O(km + nl + \min\{s^2, kl\}mn)$ where m and n are the lengths of the two alignments, k and l are the numbers of sequences in the alignments, and s is the size of the alphabet

The fitness of the individual guide trees is maintained in a fitness table that is sorted in descending order of relative fitness. This sorted fitness table is used for the selection process, as a precursor to crossover and mutation.

1.5.3 Selection, Crossover, and Mutation

A rank-based selection process is implemented wherein a parameterized beta distribution is overlaid across a sorted fitness table. Two unique parents and one tree to be replaced are chosen at random from this non-uniform probability distribution which has a strong bias for selecting parents with a high fitness as well as a strong bias towards selecting lower-fit individuals to be replaced by the offspring of crossover. The beta distribution is parameterized with $\alpha = 3.0$, and a $\beta = 0.5$, and has the distribution shown in Figure 1.4 below.

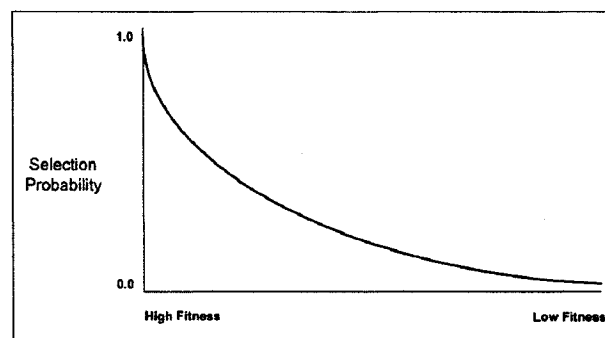


Figure 1.4 – The beta probability distribution used for selection of parents for the crossover operation. The inverse of this distribution is used for the selection of a child.

Once the rank-based selection process chooses two unique parents and one child (also unique), these individuals are processed by the crossover operator. The GA performs a type of single-point crossover in which a crossover point is chosen at random for one of the parent trees. In effect, a crossover point is simply one of the n levels on the coalescing tree data structure. A second crossover point is chosen on the second parent using a linear search for a compatible matching level.

Two binary coalescing trees are not always compatible for crossover. Preliminary analysis indicates that an incompatible selection occurs in < 5% of all cases, and appears to decrease

slowly as a function of tree size. Compatible parent trees are trees in which there exist some internal level at which the internal nodes on the trees can be entirely connected via edges in order to produce a viable child. Since the selection of crossover points is stochastically driven, we arbitrarily attempt 20 times to identify possible crossover points between two randomly selected parent trees before deciding that the trees are incompatible for crossover. In the event that two trees are found to be incompatible, we re-select new parents and attempt crossover until compatible trees are found.

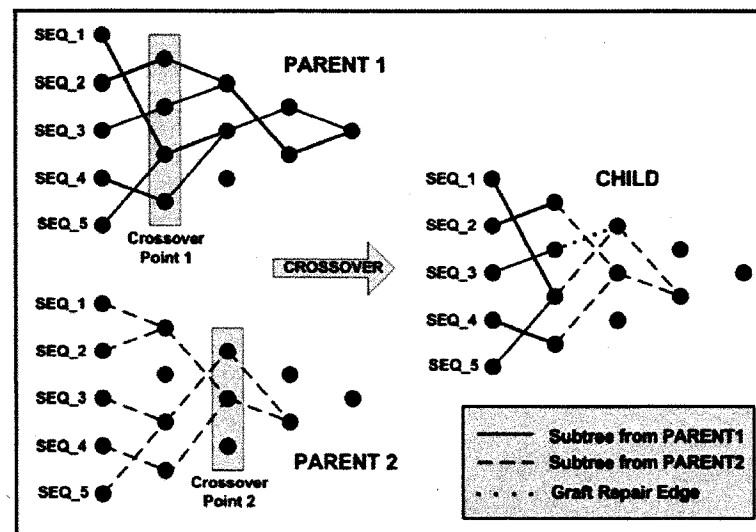


Figure 1.5: The crossover of two compatible coalescing binary trees. Note that a graft repair was needed in this example in order to prevent orphaning the leaf node SEQ_3.

The crossover process is shown in Figure 1.5. The lower portion of the first parent at Crossover Point 1 is added to the upper portion of the second parent at Crossover Point 2. This preserves all of the lower-level node relationships below Crossover Point 1 from the first parent, while mixing with the preserved upper-level sub-tree ordering specified in the second parent at and above Crossover Point 2. At the point at which the two different trees now intersect in the child tree, edges between nodes are constructed in such a way that the child tree always remains viable. In some cases, as shown in Figure 1.5, some edges will need to be constructed at the interface which did not exist at all in the previous tree. In this case, we repair the graft by randomly selecting a viable node from the next level in the coalescing tree such that no terminal nodes are ultimately orphaned, and the tree remains fully connected and viable.

Mutation of a child tree after crossover is a simple process of selecting some connected node on the tree and changing its upper edge to connect to a randomly selected, but viable node. In some cases, removing an edge from a node which is connected higher up in the tree requires a recursive repair mechanism which traverses up the tree from the newly orphaned node, removing all edges from the visited nodes until a node is found that has two connected nodes. Once all such edges are removed, the tree is again viable.

1.5.4 Algorithm Parameters

The GA has a steady-state population of 30 individual guide trees. The algorithm is iterative and not generational, and iterates for a configurable number of times before halting. For the tests and experiments conducted with this GA, the algorithm was arbitrarily terminated at 10,000 iterations. The number of iterations required to reach convergence on the globally optimal guide tree is a function of both the number of sequences and the length of the sequences. In real world application of this GA approach, the termination condition for the GA could be dynamically calculated at run-time as a function of average sequence length and the number of sequences being aligned. Alternatively, the algorithm could continue until no further improvement was observed.

Since this is an iterative, steady-state GA, a single selection and crossover happens at each generation. However, some minority of individual trees may not be compatible for crossover, and so there is an effective crossover probability that is something slightly smaller than 100%. In each iteration, each child tree from crossover has a 10% chance of incurring a single point mutation. Examining the effects of manipulating the mutation rate is an area of future work.

In addition to the GA parameters, alignment parameters were also provided. As mentioned previously, dynamic programming algorithms were used to align sequences and alignments based on the evolved guide trees. The alignments and alignment scores are produced in the context of an evolutionary model that takes the form of a scoring system that specifies the penalties for opening new gaps in an alignment or extending an open series of gaps.

The central idea behind making this distinction is based on the fundamental idea that it should be considered more expensive to open a new region of gaps than to simply extend an existing gap region. In addition, different scores are assigned to residue matches and mismatches in an alignment.

Table 1.2: Concise Summary of Experimental GA Parameters

GA Parameter	Value
Population Type	Steady-State
Population Size	30
Population Initialization	Bottom-up randomly generated, viable guide tree
Number of Iterations	10,000
Selection Type	Biased, Rank-Based using Beta probability distribution
Crossover Type	Branch Swapping on Coalescing Binary Tree
Crossover Rate	Slightly less than 1.0
Mutation Type	Random intra-tree, same-level branch migration
Random Number Generator	R250 from GNU Scientific Library

The scoring system used in this GA implements affine gap penalties, and is parameterized as shown in Table 1.3.

Table 1.3: The GA Alignment Scoring System with Affine Gap Penalties

Score Type	Value
Gap Opening Penalty	-5.0
Gap Extension Penalty	-1.0
Nucleotide Match Score	10
Nucleotide Mismatch Score	2.0

We used the same gap and substitution penalties between Clustal W and our GA to more accurately contrast the relative performance of both algorithms.

1.6 Experimental Results

All experimental runs were conducted on a workstation with a 1GHz Intel Pentium-III CPU, 640MB of RAM, running Redhat Linux version 7.2. Over 30 tests were run in which the number of sequences and/or different sequence lengths were varied. The input sequences for all of the experimental runs were constructed using a periodic, stochastic variation of the Jukes-Cantor [7] model of nucleotide evolution across a fixed star-shaped topology. All generated nucleotide sequences across such a tree are roughly evolutionarily equidistant. Evolving sequences across more realistic and variable topologies would lead to more realistic sequence simulation and should be included in future work.

For each experiment, alignments were performed both with our GA and with CLUSTAL W (v1.82). Performance, in terms of both efficiency and apparent alignment quality, are summarized for several of our experimental runs. In order to more accurately compare the results of CLUSTAL W to our GA, we attempted to identically parameterize each system. In order to do this, we identically configured affine gap penalties and substitution costs, and disabled the delayed alignment of divergent sequences in CLUSTAL W. All of the experimental runs produced similar overall results for all of the input sequences, regardless of the number of sequences being aligned, and the results presented in this section were chosen as representative results for the entire experimental evaluation of our system.

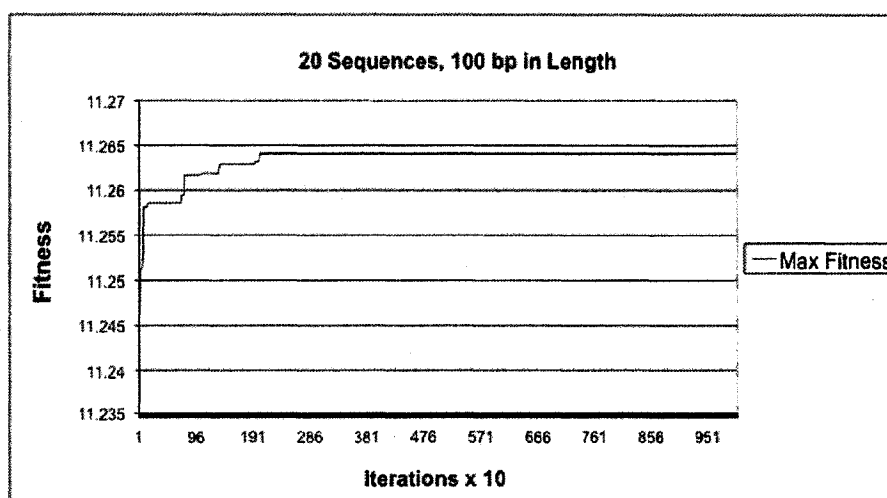


Figure 1.6: The fitness trend of the fittest individual across 10,000 iterations when evolving a guide tree to align 20 sequences of length 100 bps.

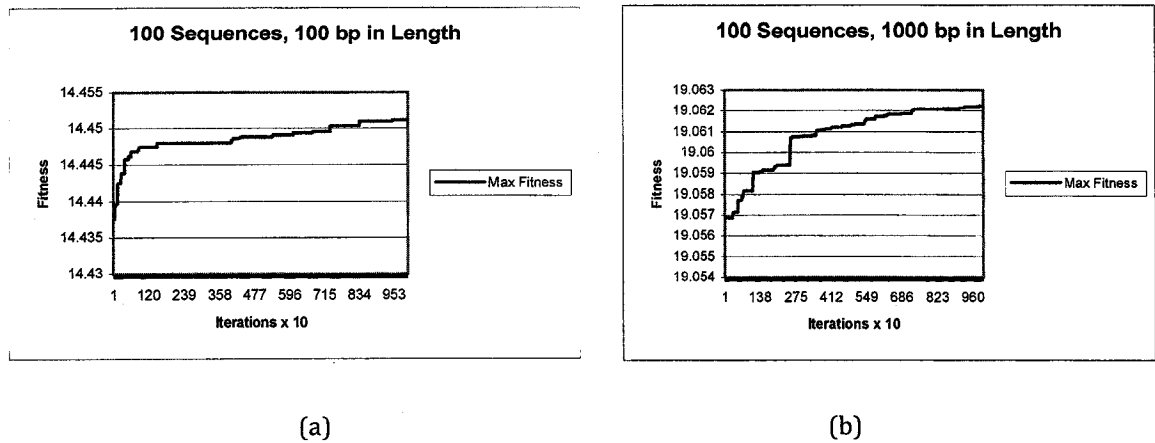
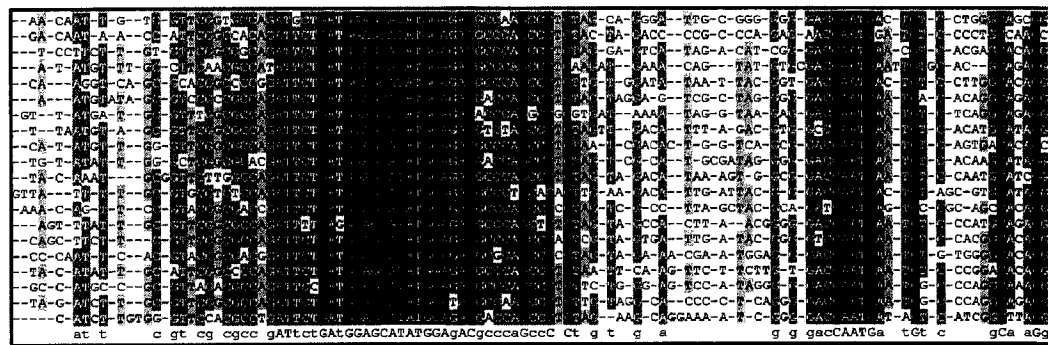
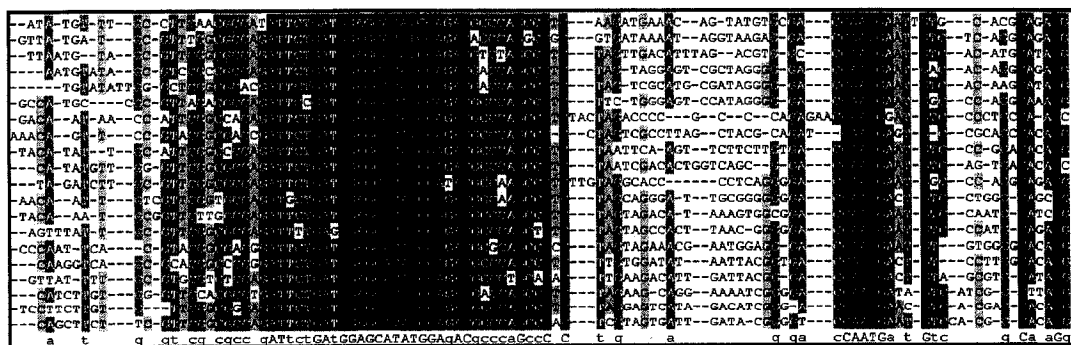


Figure 1.7: The fitness curves demonstrated by the GA when applied to two different inputs, one of which is (a) 100 bp long, and one of which is (b) 1000 bp long. Note that in (B), the fitness improvement is more gradual. This effect is due to the larger number of nucleotides in each sequence being aligned. Neither graph reached a true plateau, indicating that 10,000 iterations is insufficient for these sequences.



(a)



(b)

Figure 1.7: Visual comparison of the alignment produced by the GA (a) and the alignment produced by CLUSTAL W (b). Although extremely similar, the GA alignment is slightly more optimal. For example, the alignment produced by CLUSTAL W has one additional column, indicating that more gaps were used to construct the alignment. The slight improvement in alignment quality is also apparent on a closer visual inspection.

1.7 Conclusions

We effectively applied a novel genetic algorithm to the problem of progressive multiple sequence alignment. In addition, we presented a novel binary coalescent tree data structure that lends itself nicely to the problem of performing crossover on two trees without duplicating terminal nodes.

In our tests, our evolutionary approach has been shown to produce more refined alignments than CLUSTAL W. In addition, we believe that preliminary execution time data seem to indicate a trend that our approach may be more scalable than CLUSTAL W when aligning very large numbers of very long sequences, although more work must be performed to make a conclusive determination on the relative scalability of our technique.

1.8 Future Work

Our approach can be shown to produce better alignments than CLUSTAL W while also being more scalable than CLUSTAL W for large datasets (*i.e.* 500 sequences of length 2500 or larger). Additional testing needs to be done in order to evaluate this hypothesis and determine the precise inflection point at which stochastic optimization techniques such as genetic algorithms surpass CLUSTAL W in both alignment quality and scalability.

We intend to study the parameterization of our GA in greater detail to determine how best to converge to a near-optimal guide tree in the minimum number of iterations. The effects of modifying tunable parameters such as population size and mutation rate on solution convergence rates and solution quality will be more fully explored. Additionally, we intend to explore new styles of crossover and mutation operators based on coalescing trees in order to maximize the benefits of those operators.

Currently, this GA is designed to handle only nucleotide sequences. We intend to extend this algorithm to handle protein sequence data as well, with support for fully pluggable amino acid substitution matrices (such as BLOSUM and PAM). Once this GA supports the alignment of amino acid sequences, we intend to perform benchmarked comparisons against other techniques using the BALiBASE [8] alignment database.

The problem of sequence alignment is embarrassingly parallel. Genetic algorithms are also inherently parallel, and so we intend to parallelize our fitness function such that it can be efficiently handled in a Beowulf cluster configuration.

Finally, we will examine the effects of evolving guide trees whose fitness is evaluated by some measure of fitness distinct from final alignment quality. Other fitness criteria for applying evolutionary pressure to improving guide trees may be equally effective in achieving high-quality alignments, and perhaps considerably faster.

Chapter 2

Evolving Better Multiple Sequence Alignments

- [9] Sheneman, L., J.A. Foster (2004) Evolving Better Multiple Sequence Alignments, In the *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004)*, Seattle, WA.

2.1 Preface

We later applied the genetic algorithm described in Chapter 1 to the BALiBASE benchmark. The results were generally disappointing. Looking closer at our method, we found that crossover on the binary coalescent was too destructive to efficiently explore tree space. In fact, crossover with the binary coalescent reduced to a random search. A review of the literature on optimizing phylogenetic trees with evolutionary computation led to the idea of a simpler representation with a better understood and less destructive crossover operator. This chapter was presented at the 2004 Genetic and Evolutionary Computation Conference (GECCO) in Seattle and appears in its entirety in the peer-reviewed conference proceedings.

2.2 Abstract

Aligning multiple DNA or protein sequences is a fundamental step in the analyses of phylogeny, homology and molecular structure. Heuristic algorithms are applied because optimal multiple sequence alignment is prohibitively expensive. Heuristic alignment algorithms represent a practical trade-off between speed and accuracy, but they can be improved. We present EVALYN (*EVolved ALYNments*), a novel approach to multiple sequence alignment in which sequences

are progressively aligned based on a *guide tree* optimized by a genetic algorithm. We hypothesize that a genetic algorithm can find better guide trees than traditional, deterministic clustering algorithms. We compare our novel evolutionary approach to CLUSTAL W and find that EVALYN performs consistently and significantly better as measured by a common alignment scoring technique. Additionally, we hypothesize that evolutionary guide tree optimization is inherently efficient and has less time complexity than the commonly used Neighbor-Joining algorithm. We present a compelling analysis in support of this scalability hypothesis.

2.3 Introduction

Aligning multiple DNA or amino acid sequences is an extremely important task in modern biology. Researchers apply multiple sequence alignment (MSA) to a diverse set of problems. MSA is used to find positional homology across distinct biological sequences as a first step in inferring evolutionary relationships between organisms. MSA is used in gene identification and discovery and in identifying similarity in molecular structure and function. Among other practical applications, MSA plays a critical role in the diagnoses of genetic disease and the development of modern pharmaceuticals.

A sequence alignment is composed of two or more biological sequences that are arranged such that homologous characters within the sequences are grouped (aligned). Alignments are often represented as a two-dimensional matrix where rows are sequences and columns are sequence positions. A good alignment is one that maximizes positional homology across all columns and all sequences. Alignments are constructed by inserting *gaps* in order to shift characters and group homologous characters into columns. Since it is impossible to know whether evolution inserted or deleted characters relative to one another, these insertions/deletions are simply called *indels*. Gaps represent indels in sequence alignments. Placing a single gap in a sequence causes the remainder of the sequence to shift by one position. Gaps placed in the optimal positions will result in an alignment where positional homology is maximized as shown in Figure 2.1.

```

A-CTTCAACTAAGT-ATTG-AATAAA-CT-GCTTAGATATATCTCCAATTATTAGCTATCGCTTAT-GGATTATATTAC
ACCTTTA--TAAGTCATTG-ACT-AAGCTCGCCTAGAT-----AATTACCCGCTATCG---ATATCC-CCTATTAC
-CC-TCAACTAAGT-ATTG-AATAAAG---GCTTAGATATATCTCCAATTACTAGCTAT----TATATCCTCATAT---

```

Figure 2.1: Example of a multiple sequence alignment of three DNA sequences. Gaps represent indels and are denoted by the dash (-) character

Before the advent of alignment algorithms, researchers laboriously aligned multiple sequences by hand. This task was both error-prone and time-consuming. In the 1970's, researchers developed simple pairwise alignment algorithms based on dynamic programming (DP) and proved that they produce optimal alignments with respect to any given scoring system. [3, 4]. Although these algorithms extend easily to the simultaneous and optimal alignment of multiple sequences, they are NP-Hard [10] and have a time-complexity of $O(L^N)$ where L is the average length of the sequences being aligned and N is the number of sequences being aligned. Using DP, the simultaneous optimal alignment of more than a handful of sequences is prohibitively expensive. As a result, heuristic approaches trade quality for speed.

Progressive multiple sequence alignment is the most common heuristic [11] and is depicted in Figure 2.2. In traditional progressive MSA, a distance matrix is formed by using DP to compute the optimal edit distance between all possible combinations of sequence pairs. A clustering algorithm such as Neighbor-Joining [5] takes a distance matrix as input and deterministically constructs a *guide tree* based on these distances, grouping closely related sequences prior to more divergent sequences. Once a guide tree has been constructed, sequences are progressively pairwise aligned in the order dictated by the guide tree. Closely related sequences are aligned prior to more distant sequences. Progressive MSA avoids the computationally intractable problem of the simultaneous alignment of multiple sequences by instead performing incremental pairwise alignments.

However, traditional progressive MSA has fundamental problems. Most importantly, after sequences are pairwise aligned, any inserted gaps in that pairwise alignment become immutable, and subsequent alignments with other sequences cannot retroactively add additional information to improve previously aligned sequences. This is a form of error propagation, also known as "*once a gap, always a gap*". The guide tree has a direct qualitative impact on this error propagation, as the amount of error is heavily dependent on the order in

which sequences are progressively aligned. Since guide tree construction algorithms such as Neighbor-Joining are greedy and starting-point dependent, they are easily trapped in local optima, often resulting in suboptimal multiple alignments. We hypothesize that an evolutionary algorithm is better able to avoid entrapment in local optima and will perform better than Neighbor-Joining in constructing good guide trees. Better guide trees result in better multiple sequence alignments.

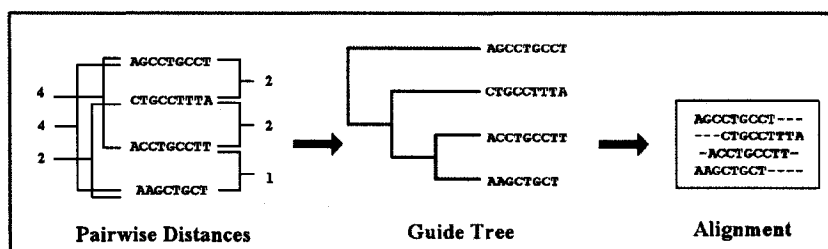


Figure 2.2: Computing pairwise edit distances to construct a guide tree. Traditional progressive algorithms cluster similar sequences prior to divergent sequences. A guide tree specifies an ordering of pairwise alignment operations that construct a complete multiple sequence alignment

Additionally, for large datasets, Neighbor-Joining has a prohibitive time complexity of $O(N^3)$, where N is the number of input sequences. As researchers apply MSA to larger and larger datasets, Neighbor-Joining scales poorly as N grows large. It is our hypothesis that an evolutionary computational approach to guide tree construction is more scalable than Neighbor-Joining.

2.4 Previous Work

Notredame and Higgins performed the seminal work in applying a genetic algorithm (GA) to MSA with a tool known as *Sequence Alignment by Genetic Algorithm* (SAGA) [12]. SAGA evolves a population of alignments using a complex set of 22 crossover and mutation operators in an attempt to gradually improve the fitness of the alignments in the population. Providing meaningful scores for sequence alignments can be somewhat problematic, and by default SAGA relies on a weighted sum-of-pairs approach [13] in which each pair of sequences in an

alignment is compared and scored and then the scores from all of the pairwise alignments are summed to produce a representative score for the entire alignment.

Although SAGA produces high quality results that are comparable (or sometimes better) than other popular heuristic techniques, SAGA scales poorly when aligning more than 20 sequences. SAGA applies a large and overly complex litany of crossover and mutation operators, which are dynamically scheduled via a sophisticated adaptive, self-tuning mechanism. By contrast, the GA approach outlined herein uses only one form of crossover and one mutation operator, thus simplifying the implementation and analysis of the algorithm.

Thomsen *et al.* [14] developed an alignment post-processing program that uses a genetic algorithm to *improve* alignments constructed by algorithms such as CLUSTAL V [15]. A population of alignments is initialized by randomly distributing gaps throughout the individual alignments yet seeding the population with a single alignment produced by CLUSTAL V. Assuming that this CLUSTAL-derived seed was of higher quality than the randomly generated seeds, any fitness improvement in the fittest individual is, by definition, an improvement over the output of CLUSTAL V. The authors aligned as many as 71 sequences with an average length of 100 residues and arguably demonstrated a 10% quality improvement over CLUSTAL V.

Related work has been done towards the application of genetic algorithms to the problem of evolving phylogenetic trees. Most notably, [16] and [17] used genetic algorithms to evolve trees which were optimized with respect to maximum likelihood. Additional previous work has been done by [18] in inferring phylogenetic trees with respect to maximum parsimony [19].

Notably, the manipulation of tree-based data structures with genetic algorithms has been widely explored in the genetic programming literature [20].

2.5 Algorithm Implementation

We present EVALYN, a novel progressive multiple sequence alignment (MSA) program that utilizes a genetic algorithm (GA) to optimize guide trees. EVALYN starts with a steady-state population of randomly constructed binary trees and iteratively optimizes this population using a combination of selection, crossover, and mutation. Guide trees are rooted binary trees. Each node in the guide tree contains an alignment that in turn contains at least one sequence. Leaf nodes have no children and contain the original input sequences.

2.5.1 Crossover and Mutation

In every iteration of the genetic algorithm, EVALYN selects two unique parents for crossover based on an exponential distribution of relative rank. This ensures that highly fit trees are selected for crossover far more often than unfit trees, yet all trees are viable crossover candidates. Similarly, EVALYN selects a single unfit guide tree to be replaced by the offspring of a crossover operation.

EVALYN's crossover operator is depicted in Figure 2.3. We implement tree crossover in a way similar to that described in GAML [21], a genetic algorithm for phylogenetic inference. Both selected parents are copied and a randomly chosen *crossover point* (internal node) is selected in the first parent. The first parent is re-rooted at the crossover point, and the remainder of the tree above the crossover point is discarded. All leaf nodes that exist in this new, smaller tree are removed from the second parent, and the second parent is *collapsed* into a typical bifurcating tree. This collapsed second parent is then *attached* to the first parent at a randomly chosen *insertion point*.

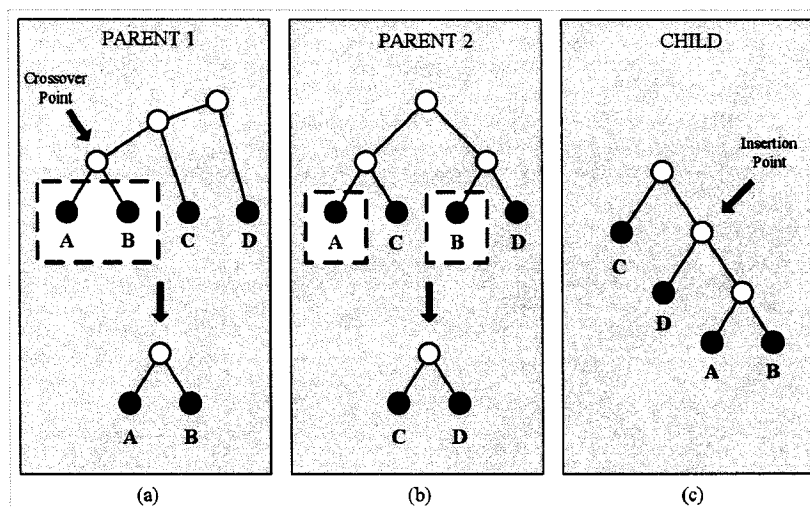


Figure 2.3: Crossover as a three-step process. First, a copy of *PARENT 1* is rooted at a randomly selected crossover point and all nodes above this point are discarded as shown in (a). Next, all leaves are removed from a copy of *PARENT 2* which exist in the newly rooted tree from (a). As shown in (b), leaves *A* and *B* are removed from *PARENT 2*, and the tree is collapsed to form a new bifurcating tree containing only leaves *C* and *D*. In (c), the final child tree is constructed by combining the sub-trees from (a) and (b) at a randomly chosen insertion point

With some small probability, EVALYN mutates the child tree by performing a same-tree branch swap. Interestingly, this is implemented by performing a crossover operation on two copies of the *same* child guide tree, effectively swapping branches within the same tree.

2.5.2 Measuring Guide Tree and Alignment Fitness

As shown in Figure 2.4, the fitness of a particular guide tree is measured by performing progressive MSA in the order dictated by the guide tree and then scoring the resulting alignment.

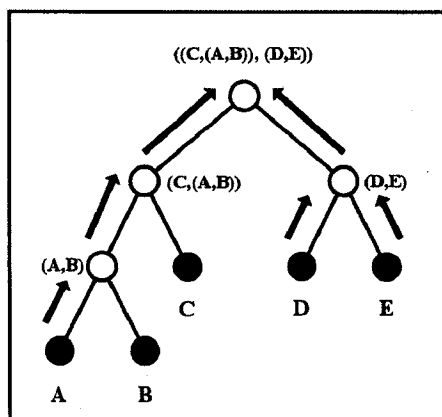


Figure 2.4: Evaluating the fitness of a guide tree. Fitness is computed by performing the progressive sequence alignment in the order dictated by the guide tree. EVALYN performs a depth-first traversal of the guide tree and aligns *A* and *B* first. Sequence *C* is then aligned to the alignment of *A* and *B* to form a 3-sequence alignment of sequences *A*, *B*, and *C*. The complete multiple sequence alignment of all sequences is performed by aligning the two alignments on either side of the root node of the guide tree. The sum-of-pairs score of the final alignment is the fitness of the alignment

We use the common sum-of-pairs score (SPS) as our scoring method as outlined in Figure 2.5. In the case of protein, the evolutionary distance between every pair of residues in each column of the alignment is computed using a probabilistic residue substitution model such as PAM [22] or BLOSUM [23]. DNA is similarly handled using simple nucleotide substitution models that properly weight transitions and transversions. The SPS for the entire alignment is simply the

sum of the SPS for each column in the alignment. Gaps are typically assigned large penalties, while substitutions are assigned smaller negative penalties or positive rewards. For our purposes, a higher SPS indicates a better alignment. Therefore, there is selective pressure favoring alignments with higher sum-of-pairs scores.

	A	C	G	T
A	2	0	0	0
C	0	2	0	0
G	0	0	2	0
T	0	0	0	2

AFFINE GAP COSTS	SPS = 2 + (-5) + 2 + 0 + 2 = 1
GAP OPEN = -5.0	
GAP EXTEND = -0.1	

AACGT	
A-CCT	
	0

Figure 2.5: A sum-of-pairs score (SPS) is computed for a simple pairwise alignment. A substitution matrix assigns points for nucleotide matches/mismatches and affine gap penalties are applied

As in other MSA implementations, EVALYN implements *affine gap penalties*, in which the leading gap in a subsequence of contiguous gaps invokes significantly higher penalties than non-leading gaps. Affine gap penalties result in dense, contiguous gapped regions instead of sparsely distributed, isolated gaps. This affine gap model is more biologically realistic and ultimately creates more biologically realistic alignments.

2.6 Algorithm Analysis

We hypothesize that EVALYN has less computational complexity than Neighbor-Joining, and is therefore more scalable than CLUSTAL W as a function of the number of input sequences. We briefly analyze the time complexity of EVALYN and compare it to the time complexity of the Neighbor-Joining algorithm used in CLUSTAL W to demonstrate support for this hypothesis. We show that with only regard to the number of input sequences, EVALYN is an $O(N)$ algorithm. By contrast, CLUSTAL W's Neighbor-Joining algorithm is $O(N^3)$.

EVALYN evaluates guide tree fitness by performing the progressive MSA in the order dictated by the guide tree and computing the sum-of-pairs score for the resulting alignment. At each step in the progressive alignment, we compute an optimal global pairwise alignment [3]. Each pairwise alignment has an $O(L^2)$ complexity, where L is the average length of the sequences or partial alignments being aligned. There are $N-1$ such pairwise alignments for every evaluation of a guide tree, resulting in an $O(N \times L^2)$ complexity, where N is the number of input sequences being aligned, and L is the average length of all sequences. This fitness evaluation happens once per iteration. With I iterations, EVALYN becomes an $O(I \times N \times L^2)$ algorithm. Finally, when evaluating the fitness of the initial, randomly generated population of size P , the fitness of each guide tree must be computed prior to iteration, resulting in an initial cost of $O(P \times N \times L^2)$. In typical usage, $I \gg P$ and we can simplify our analysis to $O(I \times N \times L^2)$. Although EVALYN is an iterative algorithm and has large amounts of constant-time overhead in the form of the multiple I , it does have a linear time complexity with respect to the N input sequences.

Although CLUSTAL W is fast in the typical usage scenario, it performs very poorly as N grows very large (thousands of input sequences). CLUSTAL W uses Neighbor-Joining to construct guide trees, and Neighbor-Joining has been shown to possess a $O(N^3)$ time complexity [24].

The *practical* question remains as to whether or not EVALYN is capable of finding comparable or better guide trees in less time than CLUSTAL W when aligning extremely large numbers of sequences. For example, if N is very large, it may be the case that I must be similarly large in order for EVALYN to converge on guide trees which score better than those constructed via Neighbor-Joining. We've shown that EVALYN is $O(N)$, but how fast does I (or P) need to grow as a function of N in order to get good alignments? Future experiments will focus on characterizing this behavior.

2.7 Experimental Setup and Results

We hypothesize that a genetic algorithm (GA) is capable of finding better guide trees than those that are constructed using traditional deterministic clustering algorithms such as Neighbor-Joining. To test this hypothesis, we compare EVALYN to the popular CLUSTAL W progressive MSA tool [2]. CLUSTAL W uses Neighbor-Joining to construct guide trees based on a computed pairwise distance matrix. Both EVALYN and CLUSTAL W compute the sum-of-pairs score for

the final multiple sequence alignment, and this is used as an objective metric of alignment quality.

First, we create DNA sequences via simulation under the Jukes-Cantor model [7] of sequence evolution in which transitions and transversions are equally probable. We simulated the DNA sequences by producing a random template sequence of the desired length and then using this template sequence to generate related sequences with no more than 50% sequence divergence. This simplistic simulation implies a true tree with a star topology and an ungapped true alignment. In this way, we generated 10 independent sets of 50 sequences, all of which were 100 nucleotides in length. We performed 10 experimental runs of EVALYN on each of the 10 input datasets and averaged the results.

Second, CLUSTAL W was run with *default* parameters on each of the 10 input datasets and we recorded the final sum-of-pairs score of the output alignment. In 10 independent trials, EVALYN used the same 10 inputs and saved the best guide tree after 2500 iterations. In all cases, EVALYN used a population size of 500 guide trees, a mutation rate of 0.01, and ran for 2500 iterations as shown in Table 2.1.

Table 2.1: Experimental settings for EVALYN.

Population Type	Steady-state
Population Size	500 guide trees
Crossover Rate	100% (steady-state population)
Mutation Rate	0.01
Iterations	2500
Selection Type	Rank-Based
Substitution Matrix	Matches = 1.9, Mismatches = 0
Gap Open Penalty	-10.0
Gap Extension Penalty	-0.01

Where possible, EVALYN was parameterized identically to CLUSTAL W with respect to gap penalties and nucleotide substitution costs. The results from this experiment are shown in Figure 2.6.

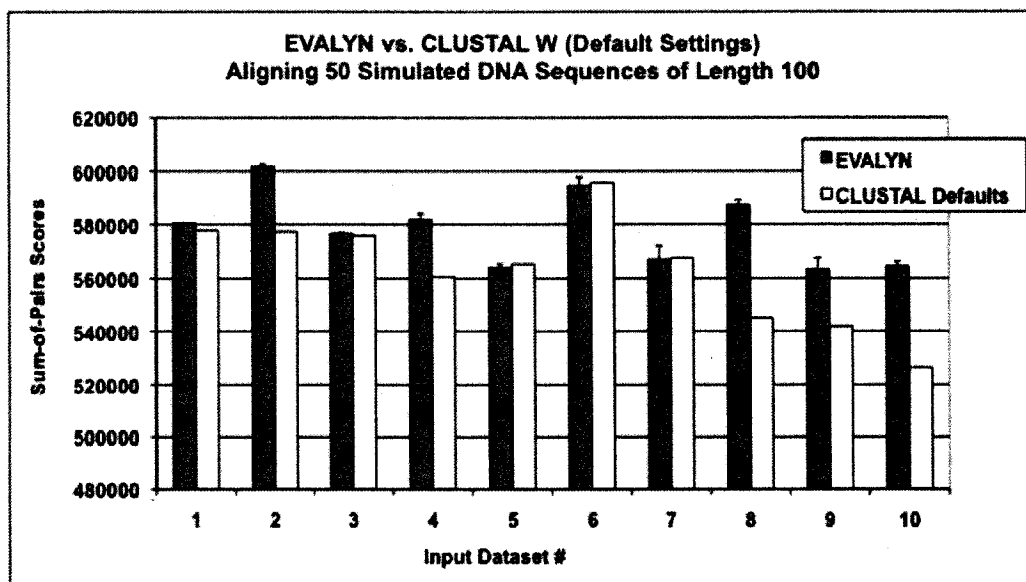


Figure 2.6: Comparing CLUSTAL W and EVALYN with default parameters. We ran CLUSTAL W with default settings. Across all 10 input sets, EVALYN produced comparable or better guide trees which resulted in better alignments with higher sum-of-pairs scores (SPS) than CLUSTAL W. Since EVALYN is a stochastic algorithm, the mean SPS and standard deviation across repeated trials is shown

Third, we invoked CLUSTAL W *again* for each dataset, but instead of generating its own guide trees, CLUSTAL W instead used the best guide trees produced by EVALYN. This technique removed any experimental error due to possible inconsistencies in alignment scoring between CLUSTAL W and EVALYN. We computed the mean and standard deviation of the sum-of-pairs scores across all 10 trials for each of the 10 inputs. We also calculated the standard deviation across EVALYN runs to assess the error and statistical significance of our results.

Results indicate that EVALYN outperforms CLUSTAL W significantly and consistently when the two programs have identical parameterization. In the case where CLUSTAL W was run with default settings, EVALYN continued to outperform CLUSTAL W as shown in Figure 2.7.

Tests of the statistical significance of all results were performed using a non-parametric Wilcoxon signed-rank test and showed that these results are statistically significant under that test.

By taking optimized guide trees produced by EVALYN and providing them as input to CLUSTAL W, we have found strong evidence to support our first hypothesis that guide trees

evolved via a genetic algorithm produce better multiple sequence alignments than guide trees constructed using Neighbor-Joining.

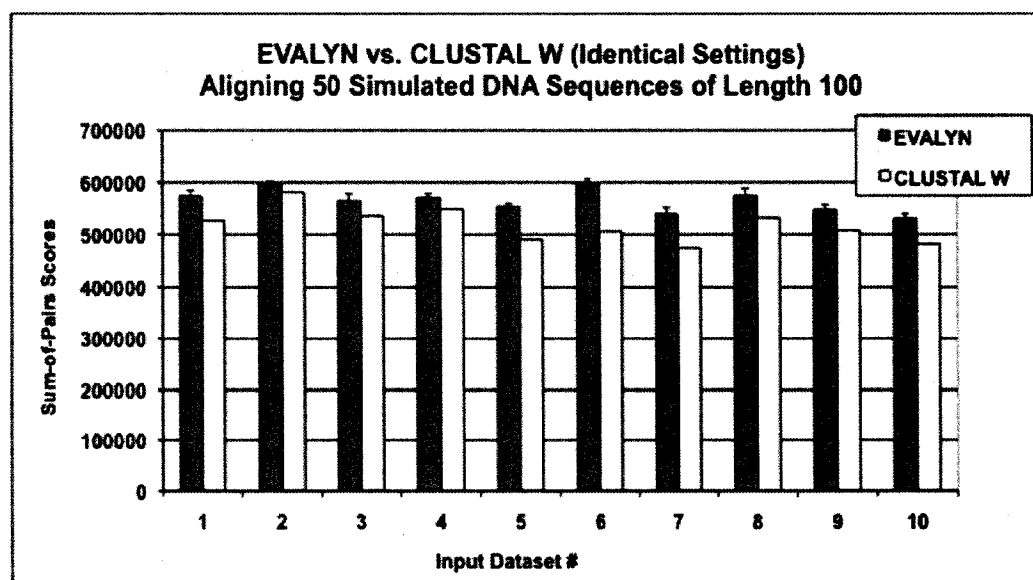


Figure 2.7: Comparing CLUSTAL W and EVALYN with identical parameters (same substitution matrix, same gap penalties, *etc.*) Across all 10 input sets, EVALYN produced better guide trees that resulted in better alignments with higher sum-of-pairs scores

2.8 Conclusions

Aligning multiple sequences of biological data is a critical aspect of modern biology. Since constructing optimal alignments is prohibitively time-intensive, popular heuristic MSA algorithms trade accuracy for speed in order to maximize their practical usefulness. We introduced EVALYN, a genetic algorithm for performing multiple sequence alignment. We demonstrated that EVALYN produces higher-scoring alignments than CLUSTAL W under the popular sum-of-pairs metric. In addition, we provided a strong analytical argument that an evolutionary computational approach to guide tree optimization as used in EVALYN is more scalable than traditional guide tree construction algorithms such as Neighbor-Joining.

By using a genetic algorithm to produce better guide trees, we achieved an important practical goal of producing a measurably better multiple sequence alignment program than CLUSTAL W, currently the most popular and actively used MSA program today.

2.9 Future Work

In future work, we will examine different metrics of alignment quality in order to show that EVALYN is able to produce biologically significant results. Although the sum-of-pairs score (SPS) for an alignment is commonly used, it has some inherent problems. First, alignments with the highest SPS are not always the most biologically significant or meaningful. Guide trees and multiple sequence alignments constructed under the SPS optimality criterion may in fact produce alignments that are meaningless when interpreted by a biologist. Finding ways of quantifying the level of biological significance of an alignment is an ongoing and active area of research. Toward this end, we intend to test EVALYN against the Benchmark Alignment dataBASE (BALiBASE) [8], which is a carefully designed set of protein alignments that were aligned and verified with elucidated or inferred structural and functional information. The BALiBASE alignments are composed of real protein sequences, and are intended to be a kind of “gold standard” by which alignment algorithms can be tested for their ability to recover biologically significant alignments. In addition to testing EVALYN against BALiBASE, we will develop new fitness functions to maximize meaningful biological signal in alignments. For example, future fitness functions may take into account predicted or elucidated secondary structure.

All phylogenetic analysis begins with multiple sequence alignment, which establishes the positional homology of the sequence data that is used for the basis of constructing and optimizing phylogenetic trees. Phylogenetic analysis is extremely dependent on the assumptions, biases, and accuracy of the initial multiple sequence alignment. State-of-the-art work in phylogenetic inferencing attempts to address this by simultaneously optimizing *both* alignment *and* phylogeny. As guide trees serve as rough estimations of sequence phylogeny, EVALYN also simultaneously optimizes phylogeny and alignment. In effect, EVALYN performs phylogenetic tree optimization by using alignment sum-of-pairs scores as the optimality criterion. Along these lines, we will explore additional measures of guide tree fitness such as parsimony [19] and the more statistically rigorous approach of maximum-likelihood [25].

Finally, empirically characterizing the scalability of EVALYN across different numbers, lengths, and types of input sequences is a key focus of future work. In this paper, we analyzed

EVALYN to show that it has a linear time complexity with respect to the number of input sequences. However, the rate of solution convergence as a function of the number of input sequences is not yet well understood. Future experimentation will explore the relationship between population size, GA convergence properties, sequence divergence effects, and alignment quality.

Chapter 3

Clearcut: A Fast Implementation of Relaxed Neighbor-Joining

- [26] Sheneman, L., Evans, J. and Foster, J.A., Clearcut: a fast implementation of Relaxed Neighbor-Joining. *Bioinformatics*. 2006 Nov 15;22(22):2823-4

3.1 Preface

This brief paper was published in *Bioinformatics* in 2006 and describes Clearcut, the reference implementation of the Relaxed Neighbor-Joining (RNJ) algorithm. The separate paper describing Relaxed Neighbor-Joining was published in the *Journal of Molecular Evolution* and appears in Appendix A.

3.2 Abstract

Clearcut is an open source reference implementation for the Relaxed Neighbor-Joining (RNJ) algorithm. While traditional Neighbor-Joining (NJ) remains a popular method for distance-based phylogenetic tree reconstruction, it suffers from an $O(N^3)$ time complexity and cannot reasonably handle very large datasets. By contrast, RNJ realizes a typical-case time complexity of $O(N^2 \log N)$ without any significant qualitative difference in output. RNJ is particularly useful when inferring very large trees. In addition, RNJ retains the desirable property that it will always reconstruct the true tree given a matrix of additive pairwise distances. Clearcut implements RNJ as a C program which takes either a set of aligned sequences or a pre-computed distance matrix as input and produces a phylogenetic tree.

3.3 Background

Researchers are handling increasingly larger datasets that will require algorithms with significantly improved performance. Relaxed Neighbor-Joining [27] is an extremely fast distance-based phylogenetic tree construction algorithm that modifies the popular Neighbor-Joining algorithm [5]. Both algorithms share similar theoretical properties, including the guarantee that the true tree will be recovered if the distance matrix is purely additive [28]. In the much more common case where distances are non-additive, the qualitative difference in output between the two algorithms is negligible.

While NJ has historically been considered an extremely fast method for inferring phylogenies, it has been shown to run in time $O(N^3)$ for all inputs [29]. While RNJ also has a worst-case runtime proportional to N^3 , it has an average-case runtime of $O(N^2 \log N)$, allowing RNJ to process much larger trees in most cases. Likewise, this speed improvement allows RNJ to bootstrap more trees in a shorter period of time without measurably sacrificing tree quality.

As the name implies, Neighbor-Joining works by starting with a star-shaped tree and iteratively joining “neighboring” nodes until a bifurcating tree is constructed. At each step, traditional NJ searches the entire distance matrix and identifies and joins the node pair with the global minimum distance. In contrast, RNJ opportunistically joins any two neighboring nodes immediately after it is determined that the nodes are closer to each other than any other node in the distance matrix. It is not required that the candidate nodes be the closest of all nodes remaining in the matrix. In this sense, our algorithm *relaxes* the requirement of exhaustively searching the distance matrix at each step to find the closest two nodes to join.

3.4 Implementation

Clearcut is a small, standalone program written in C under the Linux operating system. The current distribution also compiles cleanly with GCC under FreeBSD and Mac OS X. Clearcut is entirely a text-based program that takes all arguments on the command-line. Clearcut source code is freely distributed under the BSD license.

Clearcut implements *both* relaxed and traditional Neighbor-Joining. It is capable of taking input either in the form of a pre-computed pairwise distance matrix or a set of aligned sequences in FASTA format. When presented with an alignment, Clearcut will compute pairwise distances by first determining the percent identity between all sequence pairs. Optionally, compensation for multiple hits is possible by applying either a Jukes-Cantor or Kimura correction to the pairwise distances. These optional distance corrections can be applied in an analogous manner to either DNA or amino acid sequences.

Unlike traditional NJ, RNJ is a non-deterministic algorithm and is sensitive to the order in which distances are input and the order in which nodes are joined. Command-line options can force Clearcut to randomly reorder taxa to mitigate any stochastic bias resulting from the original order in which taxa are presented in the input. A similar argument controls whether attempts to join nodes are done randomly or in a strictly deterministic order. Attempting to join randomly selected nodes can reduce systematic bias in some cases, while it is faster to attempt to join nodes in a consistent, systematic way.

Since RNJ is a non-deterministic algorithm, Clearcut optionally allows the user to quickly generate any number of distinct, equally valid RNJ trees from the same non-additive distance matrix.

3.5 Performance

We compared Clearcut to several popular traditional NJ implementations including PHYLIP Neighbor [30], QuickTree [24], and QuickJoin [31]. We artificially constructed trees of different sizes that were representative of the two extreme tree shapes: maximally deep (pectinate) and maximally shallow (perfect). We stochastically assigned gamma-distributed branch lengths to each branch and then used the simulated tree to construct a purely additive distance matrix.

Compared to existing Neighbor-Joining programs, Clearcut's RNJ implementation reconstructed the true phylogenetic tree in a fraction of the time for all tested tree shapes and sizes (Figure 3.1). QuickJoin, the second fastest NJ implementation, was unable to handle our largest input due to its extremely large memory requirement.

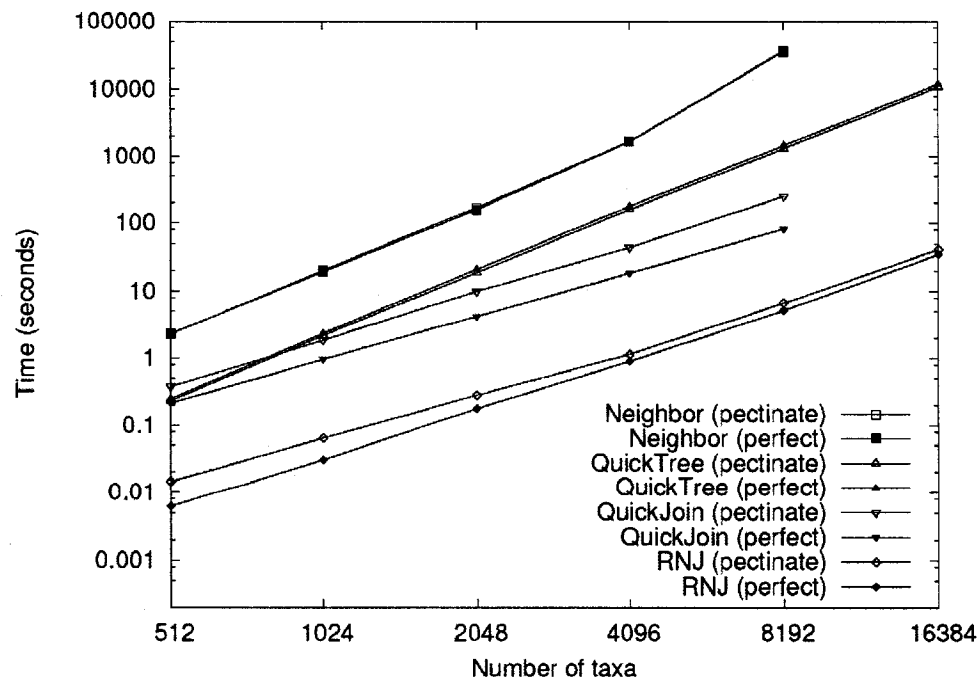


Figure 3.1: Neighbor-Joining implementation speed tests. These tests between Clearcut's RNJ implementation and other traditional NJ programs demonstrate that Clearcut is dramatically faster for all tested tree shapes and sizes. Note the logarithmic scale used on both axes.

Due to rigorous implementation optimizations, especially with respect to cache locality, even Clearcut's traditional NJ implementation is extremely fast.

3.6 Future Enhancements

Future versions of Clearcut will allow users to bootstrap RNJ trees by sampling with replacement from the provided distance matrix. A majority-rule consensus tree will be constructed from the bootstrap and nodal-support values will be computed. The labeled consensus trees will be rendered in Graphviz format.

Additionally, future versions of Clearcut will be optionally compiled first into a C library, that will allow it to be easily linked and embedded inside other programs.

Chapter 4

Estimating the Destructiveness of Crossover on Binary Tree Representations

- [32] Sheneman, L., Foster, J.A., (2006) Estimating the Destructiveness of Crossover on Binary Tree Representations, In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '06)*, Seattle WA., July 8th - July 13th, 2006.

4.1 Preface

This chapter appears in article form in the 2006 Genetic and Evolutionary Computation Conference (GECCO '06) proceedings and was presented at that conference's graduate student workshop. The technique described in this paper was developed during efforts to characterize the asymptotic runtime complexity of EVALYN. This diagnostic method makes it possible for the first time to empirically estimate the destructiveness of crossover on very specific tree representations and thereby infer an operator's expected efficiency in exploring treespace.

4.2 Abstract

For many applications, evolutionary algorithms represent individuals as typical binary trees with n leaves and $n-1$ internal nodes. When designing a crossover operator for a particular representation and application, it is desirable to quantify the operator's destructiveness in order to estimate its effectiveness at using building blocks. For the case of binary tree representations, we present a novel approach for empirically estimating the destructiveness of any crossover operator by computing and summarizing the distribution of Robinson-Foulds

distances from the parent to the entire neighborhood of possible children. We demonstrate the approach by quantifying the destructiveness of a popular tree-based crossover operator as applied to the problem of phylogenetic inferencing. The benefits and limitations of the destructiveness metric are discussed.

4.3 Introduction

Evolutionary algorithms can operate directly on many data structures, including trees and graphs. The most common example of tree-based evolutionary computation is genetic programming (GP) [20] in which a genetic algorithm operates on parse trees. The theoretical groundwork for establishing the ability of a GP to find and exploit building blocks is becoming increasingly established (for at least a narrow range of representations and crossover operators) [33,34]. However, a general schema theorem for GAs applied to tree structures is largely or entirely underrepresented in the literature.

Phylogenetic inferencing is a common application of GAs using tree structure representations [21,17,18]. A phylogeny expresses the evolutionary relationships between a set of organisms, and this relationship is often represented as either a rooted or unrooted bifurcating (*i.e.* binary) tree. The GAs search the space of valid phylogenetic trees, using the common optimality criteria of maximum parsimony or maximal likelihood. The above cited examples of phylogenetic inferencing GAs all use distinct crossover operators, but all of them operate on the same underlying binary tree structure. While the performance of GAs in the area of phylogenetic inferencing has had mixed results, most biologists who are inferring phylogenies rely on simpler techniques such as parallel hill climbing. A potential explanation for this lack of real-world adoption of GAs in this area may be due to the difficulty in establishing and quantifying the effectiveness of the GA in efficiently assembling building blocks on tree structures. Alternatively, the theoretical mechanisms by which such a tree-based GA is capable of efficiently searching phylogenetic tree space are far from understood. However, by establishing a technique for empirically quantifying the destructiveness of a particular tree-based operator, one may be able to elucidate the mechanisms by which building blocks are constructed and destroyed.

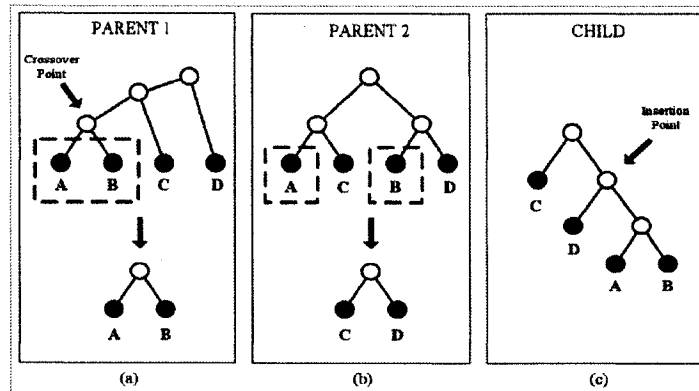


Figure 4.1: Lewis crossover with two parent trees. A node known as a crossover point is identified in the first parent tree (a) and the remainder of the tree is pruned, leaving only the crossover point and all of its children nodes. From the second parent (b), all of the nodes that were selected from the first parent are pruned from the tree. In (c), the two remaining subtrees are recombined at an insertion point, creating a child tree which should share some topological characteristics of the parents.

Several metrics exist for measuring the distance between any two trees. The most ubiquitous metric is known as the Robinson-Foulds approach [35]. This technique works by first acknowledging that any internal branch in a binary tree represents a bisection of the tree into two sets of leaf nodes. If all of the unique bisections are enumerated for both trees, one can identify and count all of the unique bisections in order to estimate the distance between trees. Robinson-Foulds is a true mathematical measure of distance, largely because it satisfies the triangle inequality [36]. That is, the direct distance between any two trees is always shorter than a distance computed using intermediate trees.

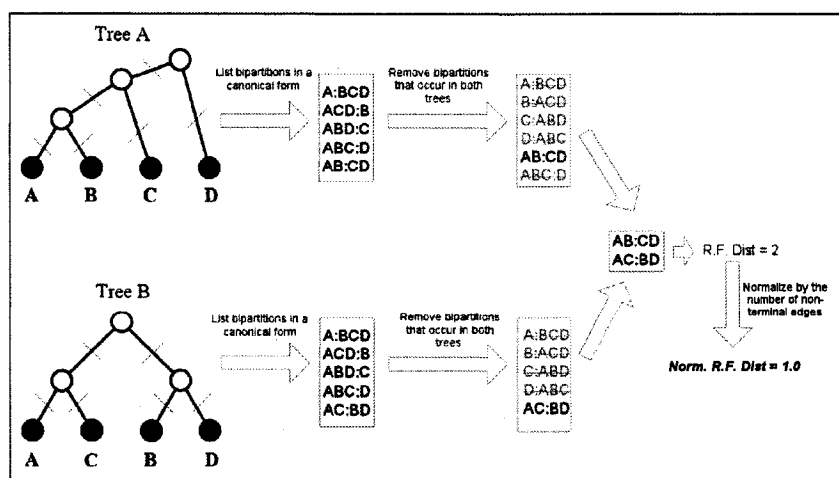


Figure 4.2: Computing normalized Robinson-Foulds distances. Here, we use two small trees with distinct topologies.

If one accepts that the EC building blocks in a tree representation are entirely or largely described by the topological characteristics of the tree, and one further assumes that the Robinson-Foulds metric reasonably estimates the distance between the topologies of the two trees, then Robinson-Foulds can be applied as a reasonable measure of the destructiveness of the crossover operator. One can recombine two parent trees and perform crossovers in all possible combinations (or a sufficiently large and unbiased sample of combinations). From this, one can build a representative neighborhood of child trees that resulted from all possible recombinations of the original parent trees by the crossover operator. By measuring the Robinson-Foulds distance between every child and each of its two parents, one can estimate the destructiveness of the operator by analyzing the distribution of R.F. distances that result. For example, child trees that have very little topological resemblance to either of their parents have obviously been transformed via crossover to such a degree that most meaningful building blocks have likely been destroyed. Overly destructive crossover operators are identified by summarizing or visualizing distributions of Robinson-Foulds distances.

4.4 Methods

We randomly generated several pairs of parent binary trees, each with ten leaf nodes. For each pair of parents, we exhaustively applied Lewis's crossover operator at every valid crossover and insertion point and generated the neighborhood of all possible offspring to the two parent trees under Lewis's crossover. We then calculated the normalized Robinson-Foulds distance from every child to each of the original parent trees. This resulted in two distributions: one distribution of distances of child to parent for each parent. Both distributions were summarized and plotted as a histogram.

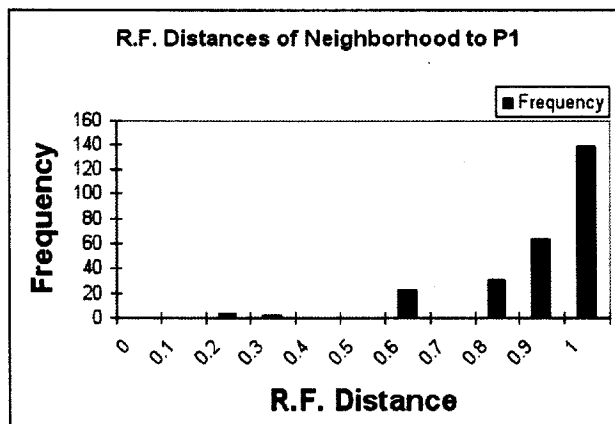
For perspective and as a form of experimental control, we also generated a random tree and then generated a population of random trees of approximately the same magnitude as the neighborhood of children resulting from the application of Lewis's crossover. The distribution of Robinson-Foulds distances from each of the randomly generated trees to the original tree was computed. This distribution represents the worst-case for which the neighborhood should have no systemic topological similarity to the original tree. The distribution of distances from

trees generated by highly destructive recombination should approach this worst-case distribution, and thus this worst-case distribution can be used as a form of experimental control and comparison.

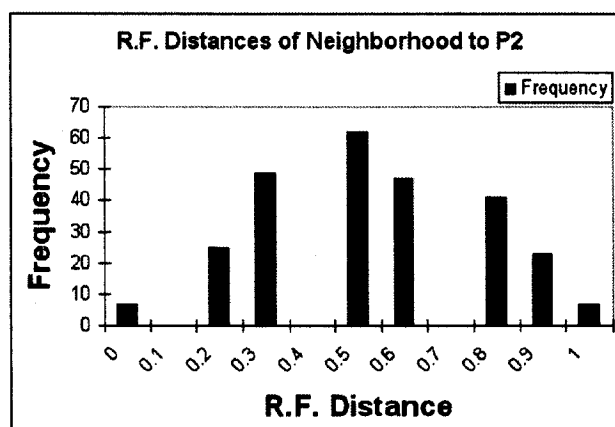
4.5 Results

We discovered some surprising, yet easily explained details about the underlying mechanisms and overall destructiveness of Lewis's crossover operator. The two distributions of the crossover neighborhood vs. each parent tree were significantly different in mean, variation, and overall shape as shown in Figure 4.3. The obvious conclusion to be reached from this result is that Lewis's crossover operator is consistently far more destructive to the first parent than the second. This result indicates a systematic bias in the way in which Lewis's crossover uses potential building blocks from each parent: Lewis's crossover operator preserves most of the topology (*i.e.* building blocks) of the second parent and destroys almost all of the building blocks from the first parent.

This asymmetrical crossover destructiveness occurs for the simple reason that in binary trees, the majority of nodes are closer to the terminals of the tree. In fact, approximately half of all nodes are terminals, and one-quarter of all remaining nodes directly share an edge with a terminal. This introduces a dramatic bias in the way in which nodes are selected for crossover points from the first parent. A review of the source code for Lewis's GAML indicate that this bias is not programmatically removed from his crossover algorithm. Because of this bias of randomly choosing nodes near the leaves of the first parent as crossover points, only single nodes or very small subtrees are preserved from the first parent, and then these are attached to a minimally pruned second parent.



(a)



(b)

Figure 4.3: Example pair of distributions of Lewis crossover destructiveness. This is a representative pair of distributions found when analyzing the destructiveness of Lewis's crossover operator. In each of our experimental runs, we found that in general the distances from the neighborhood to the first parent (a) were far greater ($mean=0.878$, $stdev=0.169$) than the distances of the neighborhood to the second parent (b) ($mean=0.487$, $stdev=0.234$). The shapes of the distributions were always consistent, with (a) being largely exponential in shape, while (b) was largely normal in shape.

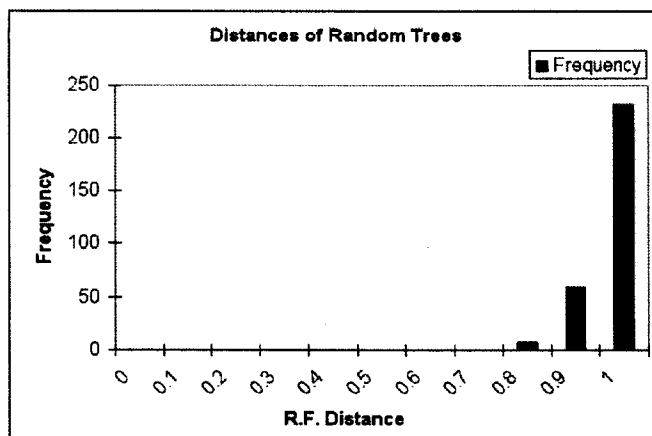


Figure 4.4: The distribution of Robinson-Foulds distances. Here, we show the distance from one randomly generated tree to a large group of other randomly generated trees. This distribution is useful as an experimental control, in that it shows the worst-case distribution of scores for an operator that is as maximally destructive as a random step. The mean R.F. distance was 0.964, with a standard deviation of 0.07.

4.6 Discussion

A distribution of Robinson-Foulds distances gives an estimation of the destructiveness of a particular crossover operator. Different crossover operators should have different measures of destructiveness. Obviously, with a GA, every operator is destructive to some extent. The goal in designing an effective crossover operator is to balance the destructiveness in order to ensure that the operator is not so destructive that the operator is throwing away everything that might have been beneficial in the parents. Such a massively destructive step is equivalent to proposing a highly inefficient random step. However, an operator that is so non-destructive that the operator is proposing nothing but small, local search steps is unlikely to be effective either.

We applied the Robinson-Foulds distance metric to the crossover operator that Lewis implemented in GAML. We found some surprising results that gave us insight not only into how effective we can expect Lewis's crossover operator to be in general, but it allowed us to trivially identify the underlying biases present in the crossover operator which would prevent it from behaving efficiently.

Robinson-Foulds is not without its problems. First, the distance estimate given by the Robinson-Foulds approach is a weighted estimate of the distances and is heavily influenced by the starting topology of the tree. For example, seemingly small, individual changes to pectinate topologies can result in maximum normalized distances and these large distances are not possible in one step when starting with balanced (perfect trees). Ideally, one would discard Robinson-Foulds and instead use a distance metric that computes or estimates the minimum edit distance between tree topologies (*i.e.* the cost of transforming one tree to another by application of the operator). Unfortunately, the general problem of determining the minimum edit distance is NP-complete. In addition, for our use in generating a distribution of distances between parents and all possible children, such an edit-distance approach is non-informative (all distances would be 1), thus using Robinson-Foulds as a topological distance estimate seems a reasonable approach. Finally, since we are dealing with randomly generated trees which are, taken as a whole, closer to perfectly balanced than pectinate, the chances of dealing with the topology-oriented biases of Robin-Foulds are mitigated.

Chapter 5

Eliminating Dynamic Programming Bias in Multiple Sequence Alignment Algorithms

5.1 Preface

This paper was submitted to BMC Bioinformatics in 2007 and rejected. We may revise the work and resubmit to another journal.

This work stems from dealing with subtle implementation details uncovered while programming EVALYN. EVALYN uses Needleman-Wunsch dynamic programming (DP) to perform optimal pairwise alignments, but the backtracking step often encountered arithmetic ties, which were stochastically resolved to avoid bias. This work explores the nature and frequency of these dynamic programming ties and a novel sampling approach for finding better alignments.

5.2 Abstract

Most useful algorithms that align multiple molecular sequences rely on dynamic programming. In practice, dynamic programming algorithms commonly produce arithmetic ties. Conventional methods for resolving these ties often produce biased results through arbitrary and deterministic strategies. These tie resolution strategies substantially influence the final sequence alignment, especially in the context of progressive multiple sequence alignment. With progressive alignment, the search through alignment space is constrained by tie resolutions since alignment steps depend heavily on previous tie resolution decisions. In fact, the standard model of progressive multiple sequence alignment is flawed, as we demonstrate with a simple example where the optimal multiple sequence alignment can only be achieved by combining

sub-optimal sub-alignments. We demonstrate that progressive alignment with unbiased tie resolution results in a set of possible alignments and a corresponding distribution of alignment scores. This distribution can be sampled to find better alignments.

5.3 Introduction

Dynamic programming (DP) is a common, mathematically rigorous technique for solving a wide variety of optimization problems [37]. In DP, problems are broken down into independent sub-problems, and these sub-problems are individually and systematically solved and eventually combined to assemble the overall solution. DP is inherently *Markovian* in the sense that every incremental step of DP is dependent only on a small set of immediately previous steps (*i.e.* adjacent sub-problems). Dynamic programming is commonly used to optimally align molecular sequences with respect to a fixed substitution matrix and gap penalties [3,4,38].

In pairwise sequence alignment, DP constructs a 2-dimensional matrix in which all the characters from one sequence (or alignment profile) are placed across the top of the matrix, one character per matrix column. Likewise, the other sequence (or matrix profile) is placed along the left edge of the matrix. The matrix is then filled with values moving from one corner of the matrix to the other corner in a systematic fashion as depicted in Figure 5.1. The value of each cell is determined by the three adjacent cells that were solved one step previously. At every cell, the cost of substituting residues or adding/extending a gap is assessed. These costs are determined by the choice of substitution matrix and affine gap penalties. Every cell is assigned a value in order to maximize the local benefit (or minimize the cost) of moving from one of the three adjacent cells to the current cell, and is computed using the following recurrence relationship:

$$F(i,j) = \max \left\{ \begin{array}{l} F(i-1,j-1) + s(x_i,y_j), \\ F(i-1,j) - d, \\ F(i,j-1) - d. \end{array} \right\}$$

$F(i,j)$ is the current cell in the DP matrix F at position i,j . The cost function $s(x,y)$ is a lookup into a substitution table describing the cost of substituting symbol x_i with symbol y_i . d represents the cost incurred for adding a gap. Substitution matrices for nucleotides typically reflect

differences in transversion/transition ratios, while substitutions for proteins are commonly based on empirically derived, probabilistic models such as PAM [22] and BLOSUM [39]. *Affine* gap costs are often used, which assigns a fixed gap penalty for starting a new gapped region, and a smaller penalty for extending an existing gapped region [40].

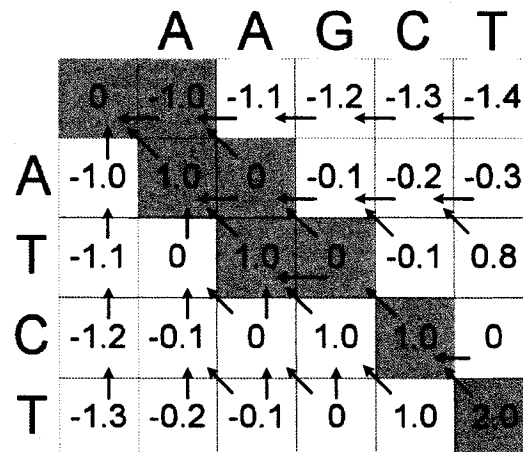


Figure 5.1: The Needleman-Wunsch dynamic programming matrix. The cells of the matrix are initially computed based on specific substitution rates and gap penalties. The algorithm then backtracks from the lower right to the upper left cells to reconstruct the alignment in reverse. In this example, there are three ties along the backtracking path. These backtracking ties lead to three equally optimal pairwise sequence alignments. We used a simple substitution/gap model where matches = 1, mismatches = 0, initial gaps = -1.0 and gap extensions = -0.1.

When the entire DP matrix is filled, the path that led from the starting point to the end point is uncovered through a process known as *backtracking*. Backtracking is an algorithmic procedure that identifies the valid steps at each cell that could have led to the cell being filled with the locally optimal value that it ultimately received.

Identifying an optimal sequence alignment for a set of input sequences is known to be NP-Hard [41, 10]. Dynamic programming techniques sequence alignment compute the provably optimal alignment for n sequences of mean length k in time $O(k^n)$. In other words, optimal simultaneous multiple sequence alignment via dynamic programming quickly becomes computationally intractable as the number of input sequences increase, even when the dynamic programming matrix is evaluated only within pre-computed bounds [13].

5.3.1 Dynamic Programming Ties

Dynamic programming *ties* can and do occur. Ties happen when the value of a cell can be computed with the identical values from more than one adjacent cell. Due to the presence of these ties, the path through the dynamic programming matrix is not a single, branchless, linear path, but a directed acyclic graph (DAG). In the presence of ties, the backtracking algorithm changes from a simple linear traversal to a graph traversal.

The backtracking step can break ties arbitrarily and deterministically, or break ties in a wholly stochastic, unbiased way. In the interest of speed and simplicity (and perhaps naiveté), most progressive multiple sequence alignment algorithms break these ties in an arbitrary fashion, severely biasing the resultant alignment. Other algorithms break the tie stochastically by assigning each valid path equal probability. For example, by giving each of two valid paths a 50% chance of consideration. However, this method of tie resolution is also flawed.

It is obvious that when ties are encountered along the backtracking path, the path ceases to be strictly linear and should really be considered a directed acyclic graph (DAG). The backtracking DAG represents a set of possible, equally optimal paths through the DP matrix. One enumerates the set of optimal pairwise alignments by exhaustively traversing all paths through the DAG.

In many cases, traversing all possible paths through the backtracking DAG is prohibitively expensive, and branching due to arithmetic ties in the DAG is resolved arbitrarily or in a stochastic, but biased fashion.

5.3.2 Progressive Multiple Sequence Alignment

Feng and Doolittle proposed the progressive multiple sequence alignment method [42] as a practical alternative to the intractable simultaneous alignment of n input sequences using n -dimensional dynamic programming. PMSA represents a pragmatic tradeoff between alignment accuracy and speed. While the final alignment is no longer guaranteed to be optimal, it is computed relatively quickly. Instead of using dynamic programming techniques on all n sequences simultaneously, PMSA applies *pairwise* dynamic programming to optimally align pairs of sequences and subalignments.

This method reduces the amount of computation from $O(k^n)$ in the case of optimal simultaneous alignment to $O(nk^2)$ in the case of progressive alignment. In short, PMSA

represents a dramatic asymptotic runtime improvement with respect to both the average length and number of input sequences.

PMSA uses estimated evolutionary distances between all sequence pairs to construct a *guide tree*. The guide tree is typically constructed using a distance-based tree inference algorithm such as Neighbor-joining [5,29]. Guide trees dictate the order of the pairwise DP alignments. The guide tree is an estimate of the phylogeny of the input sequences as determined by the relatedness of the sequences to one another. By aligning more similar sequences prior to more distant sequences, one limits the introduction of excessive and immutable gaps early in the alignment process. As sequences are added to a progressive alignment, previously introduced gaps cannot be retroactively altered. This leads to the PMSA phenomenon of “once-a-gap, always-a-gap”.

Arithmetic ties encountered during dynamic programming have a particularly pronounced effect in PMSA algorithms. This effect is discussed in detail in Section 5.6.

5.4 The Frequency of Ties

Ties occur frequently in dynamic programming. The frequency of ties is determined by several factors, including the content and relatedness of the sequences and alignment parameters such as choice of substitution matrix and affine gap costs. By simulating pairs of nucleotide sequences under specific evolutionary models, we demonstrate possible scenarios and patterns of tie frequency as a function of sequence similarity.

We use distinct simulated sequence pairs stratified into different levels of sequence similarity, from low to high. The sequences are generated by simulation under the F84 model of sequence evolution [43] using the ROSE program [44]. By using otherwise fixed alignment parameters with these sequences, we pairwise align the sequences with the Needleman-Wunsch algorithm and count the frequency of dynamic programming ties across different levels of sequence relatedness.

We hypothesize that DP ties become more frequent as sequence similarity increases. The results summarized in Figure 5.2 below provide support for this hypothesis.

We found that when aligning very similar sequences, the path through the backtracking DAG has fewer equally optimal branches. By contrast, highly divergent sequences include many ties along the DAG. This may indicate that DP matrices for highly divergent sequences simply have

less inherent signal, and therefore DP ties occur more frequently due to chance. By contrast, closely related sequences may contain a stronger DP signal where chance ties are less likely.

The most common type of tie involves two-way ties with the diagonal direction. Several examples of such ties are shown in Figure 5.1. Ties between horizontal and vertical directions are very rare, as are three-way ties between all adjacent nodes.

Figure 5.3 indicates that the number of DP ties through the *entire* matrix remains fairly constant, independent of sequence relatedness. Ultimately, the size and branching of the DAG shrinks as sequence relatedness increases, despite a fairly constant overall number of ties throughout the matrix.

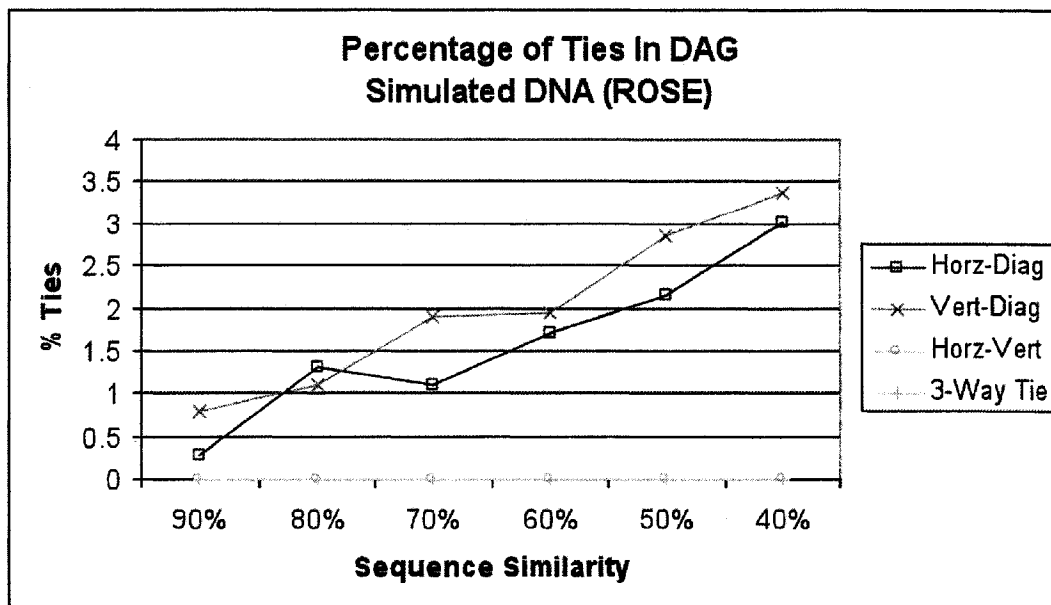


Figure 5.2: Arithmetic tie frequency along the directed graph during backtracking. Here we present an example of the number of arithmetic ties encountered during backtracking. All four distinct kinds of ties are counted separately. The figure shows averages computed over multiple replicates. The 1 Kb nucleotide sequence pairs were simulated with ROSE under the F84 model. We computed percent sequence identity post-alignment using CLUSTAL W. Here, our DP algorithm used match=+1, mismatch=0, gap open = -1, gap extend = -0.1.

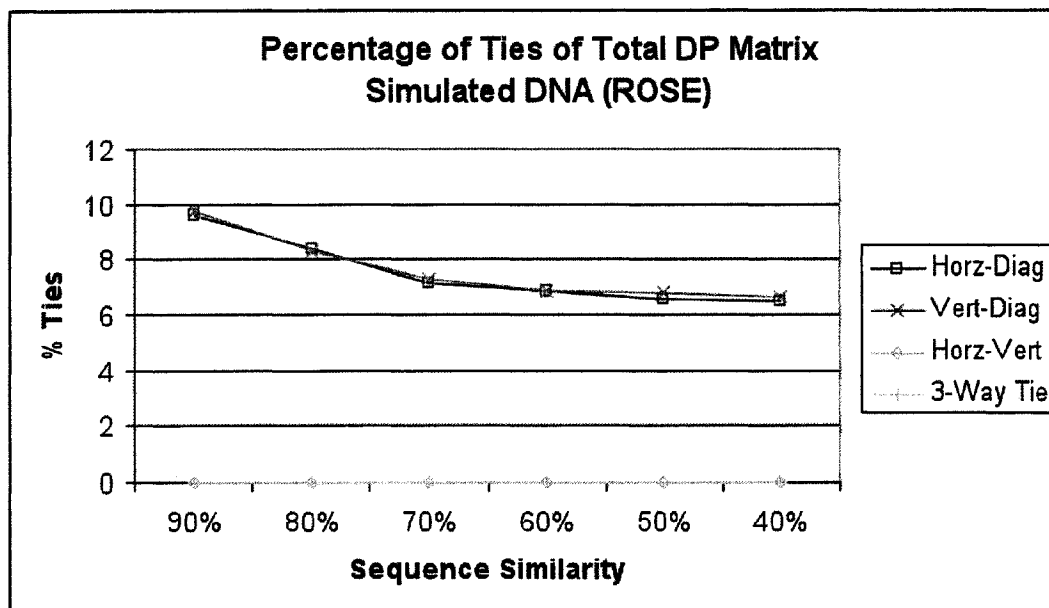


Figure 5.3: Counting ties in DP matrix as function of sequence similarity. We present the tie count averaged over multiple replicates. The sequences were simulated as in Figure 5.2. The full DP matrix is 1000x1000, of which up to 10% of the cells contain at least one kind of tie. The correlation of tie frequency to sequence similarity is significantly different from the strong linear correlation found along the actual backtracking DAG as shown previously in Figure 5.2.

5.5 Removing Bias from Dynamic Programming

Most existing dynamic programming implementations for molecular sequence alignment break ties in an arbitrary and deterministic way that introduces bias. Superficially, this appears justifiable since there seems to be no reason to prefer one equally optimal pairwise alignment over another. However, in the context of progressive multiple sequence alignment, where the particular output of one stage becomes the input to the next, choosing between equally optimal pairwise alignments becomes vitally important.

In the case of pairwise DP alignment, ties involving the diagonal are sometimes resolved by simply choosing the diagonal path. This move results in no immediate gap insertions. At first glance, this technique appears to reduce the total number of gaps in the alignment since diagonal moves represent substitutions and do not add gaps to an alignment. In reality, this arbitrary tie-breaking technique simply delays gap insertions until later backtracking stages,

systematically biasing gap placement along the aligned sequences. Finally, there is no mathematical or biological justification to arbitrarily prefer diagonal paths over other, equally optimal paths.

Another common approach to breaking DP ties is to essentially “flip a coin” wherever a tie is encountered during backtracking. While neither arbitrary nor deterministic, this stochastic approach remains biased because there are often an unequal number of ways to get to any particular cell along the backtracking DAG. Flipping a coin during backtracking, disregarding the number of paths leading to the current cell, results in choosing some paths with higher likelihood than other valid paths. Since there is no objective justification for choosing one equally valid backtracking path over another, this biased tie resolution strategy artificially constrains the alignment.

We propose an extension to traditional dynamic programming algorithms that allows for stochastic, unbiased tie resolution during backtracking. Our extended DP algorithm impacts both the matrix fill and backtracking steps. The algorithm stores not only the value at each cell, but also a list of the valid directions that lead to that cell. We also store the total number of paths that lead to that cell. The total number of paths for any particular cell is computed as the sum of the number of paths from the adjacent cells in the valid direction list. For example, if a cell's value can be computed from two adjacent cells, then there is a two-way tie. The number of paths leading to the current cell is the sum of the number of paths leading from the two adjacent, tied cells.

During backtracking, when a tie is encountered, we stochastically resolve the tie by choosing a valid random direction *with weights proportional to the number of paths from each direction*. By proportionally weighting our backtracking directions, all valid paths to the current cell are considered with equal likelihood. An example of this extended dynamic programming algorithm is shown in Figure 5.4 below.

The extended dynamic programming algorithm for aligning sequences X and Y using substitution matrix S and gap penalty D :

The Modified Matrix Fill Algorithm (for global alignment)

```

FOR EACH row  $i$  AND column  $j$  in DP matrix  $F$  DO:
   $F_{i,j}.val \leftarrow \max(F_{i-1,j-1} + S(X_i, Y_j), F_{i,j-1} - D, F_{i-1,j} - D)$ 
   $F_{i,j}.dirs \leftarrow \text{find\_directions}()$  // including tied dirs
  FOR EACH direction  $r$  in  $F_{i,j}.dirs$  DO:
     $F_{i,j}.npaths \leftarrow F_{i,j}.npaths + F_r.npaths$ 
  DONE
DONE

```

The Modified Backtracking Algorithm (for global alignment):

```

 $i = \text{length}(X)$  // start at lower-right corner
 $j = \text{length}(Y)$ 
WHILE  $i > 0$  AND  $j > 0$  DO:
   $wt.up \leftarrow F_{i-1,j}.npaths / F_{i,j}.npaths$ 
   $wt.left \leftarrow F_{i,j-1}.npaths / F_{i,j}.npaths$ 
   $wt.diag \leftarrow F_{i-1,j-1}.npaths / F_{i,j}.npaths$ 
   $newdir \leftarrow \text{rand\_weight}(wt.up, wt.left, wt.diag)$ 
DONE

```

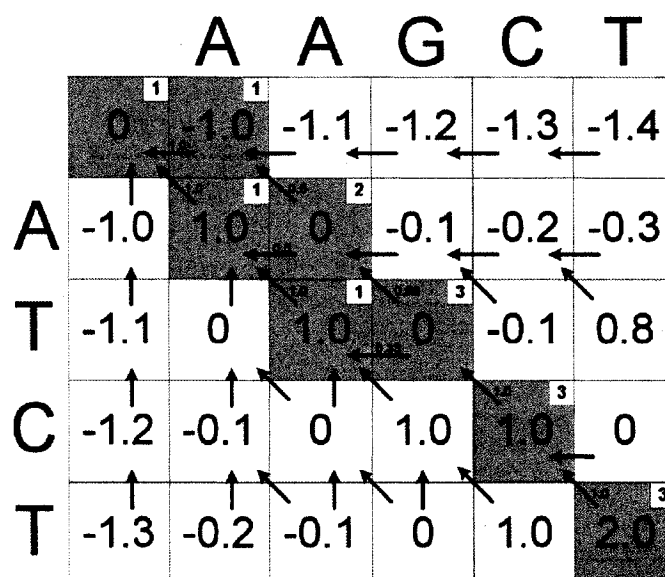


Figure 5.4: Extending dynamic programming: unbiased, stochastic backtracking. For every cell, we track the number of paths leading to that cell (shown in upper-right squares in every cell along the backtracking graph). When randomly resolving ties during backtracking, our choice is weighted based on this path count, providing equal weight to every path through the matrix. In this example, the weights are depicted on the backtracking path lines.

5.6 One Guide Tree, Many Alignments

The current dogma with progressive multiple sequence alignment implies that there exists a one-to-one relationship between guide trees and alignments. That is, a set of input sequences, an evolutionary model (*e.g.* alignment parameters) and a guide tree determines the single resulting alignment. However, the presence of arithmetic ties during dynamic programming in progressive alignment contradicts this dogma. The goal of a good PMSA algorithm is to search for and find the best possible multiple sequence alignment achievable through progressive sequence alignment (*i.e.* *the globally optimal progressive alignment*).

During pairwise alignment, DP ties result in a set of equally optimal pairwise alignments. This has a *combinatorial effect* during progressive MSA, as one *set* of possible pairwise alignments could be aligned with yet another set of possible pairwise alignments. The choice of which optimal alignments to align next during PMSA constrains the possibility in subsequent alignment steps during the progressive alignment. Unfortunately, there is no information present to inform the choice of which optimal alignments to align at each step. While every pairwise alignment step results in a set of one or more equally optimal alignments, the stepwise choice of alignments/profiles impacts the ability to find the globally optimal progressive alignment. Ultimately, every tie-breaking decision constrains the search for the globally optimal MSA. Since the choice of alignments at each pairwise step is uninformed, it is rare that a PMSA algorithm achieves the globally optimal PMSA for any given set of inputs.

A guide tree does not correspond directly to a single, deterministic multiple sequence alignment. A guide tree, together with input sequences and alignment parameters, results in a *set* of possible complete progressive multiple alignments. This set of progressive alignments is rarely equally optimal. In reality, a single guide tree represents an entire set of possible multiple alignments and a corresponding *distribution* of alignment scores.

Figure 5.5 shows the output of a simulation experiment wherein we fixed the parameters to the progressive alignment algorithm and sampled the alignments produced through unbiased tie resolution as described previously. We simulated 10 sequences using ROSE [44] under the F84 model. ROSE introduced indels of different sizes in the output sequences, and outputted the “true” alignment. We fixed the guide tree and other alignment parameters and sampled 1,000 alignments by simply resolving dynamic programming ties in different ways. We scored the alignments by measuring their distance from the true alignment output by ROSE.

We align the same set of sequences using the same inputs via CLUSTAL W [2] to demonstrate how a deterministic progressive alignment algorithm performs relative to the distribution of possible alignment scores. We find that sometimes CLUSTAL W performs relatively well, sometimes it performs poorly, but it always misses the complete picture (the full distribution of alignments) and will almost never arrive at the best possible progressive alignment for any particular set of inputs.

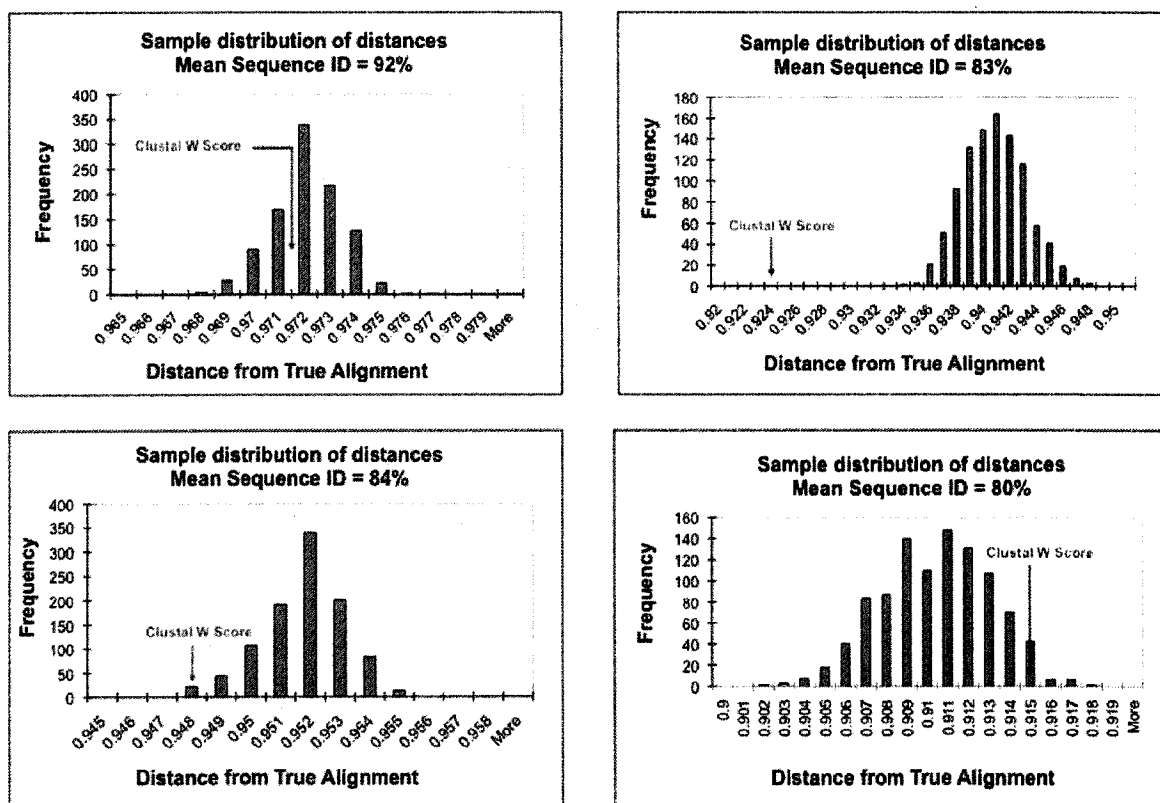


Figure 5.5: Example alignment score distributions, given a single guide tree. We show four distributions of 1,000 alignment scores each. For each distribution, we aligned 10 simulated nucleotide sequences and resolved ties in an unbiased, stochastic fashion. Within each sequence set, the alignments were derived from the same fixed guide tree and model parameters. All differences in alignments and scores within each example are entirely due to different output due to stochastic tie breaking. In each example, we show where the alignment generated by CLUSTAL W scored with respect to the complete distribution.

5.7 A Progressive Alignment and Dynamic Programming Counterexample

Progressive multiple sequence alignment suffers from a fundamental limitation known as “once-a-gap, always-a-gap” [42]. Sequences and alignment profiles are pairwise aligned in the order dictated by a guide tree using only the local information present in the sequences at hand. Most PMSA algorithms use dynamic programming to achieve locally optimal pairwise alignments. Subsequent alignment steps cannot or do not use new information to retroactively inform and optimize the relative gap positions for sequences that have already been aligned. Resolved homology and relative gap placement are immutable between already-aligned sequences.

The key problem with the current PMSA paradigm is that optimal pairwise alignment at each step of the progressive alignment process constrains the search for the globally optimal alignment achievable through PMSA. PMSA algorithms are greedy because they always choose the locally optimal alignment at each step, even if choosing a suboptimal pairwise alignment at an earlier step may subsequently result in a globally optimal progressive alignment.

While this limitation of progressive multiple sequence alignment is well known, we present a concrete, visual example of the effect in Figure 5.6. In this example, we progressively align three short DNA sequences. In one case, we align the sequences in the traditional fashion by consistently choosing from locally optimal pairwise alignments. In the other case, we vary from the optimal dynamic path slightly, resulting in a locally sub-optimal pairwise alignment but a globally superior overall progressive alignment.

In this specific example, we first pairwise align Seq1 and Seq2 and take the resulting alignment profile and align it with Seq3. There is one possible optimal pairwise alignment of Seq1 and Seq2 (*i.e.* there are no dynamic programming ties), and in this case the optimal backtracking path is precisely along the diagonal. If we align Seq1 and Seq2 optimally, and take the resulting alignment profile and optimally align it with Seq3, the result is globally suboptimal. This is demonstrated by aligning Seq1 and Seq2 together in a demonstrably suboptimal way by diverging slightly from the diagonal. When we take this locally suboptimal pairwise alignment and align it with Seq3, the result is globally superior to the first example.

It is infeasible to try all possible optimal and even slightly suboptimal paths in all combinations for large datasets. However, we believe this counterexample points to clear

possibilities for future improvements for progressive alignment algorithms. Progressive algorithms could sample alignments, not only from optimal backtracking paths, but from paths in close proximity to the optimal path. It is possible to sample with trial and error from such alignments to incrementally approximate the globally optimum progressive alignment. Such sampling could continue as long as time permits, as determined by the user of the alignment algorithm.

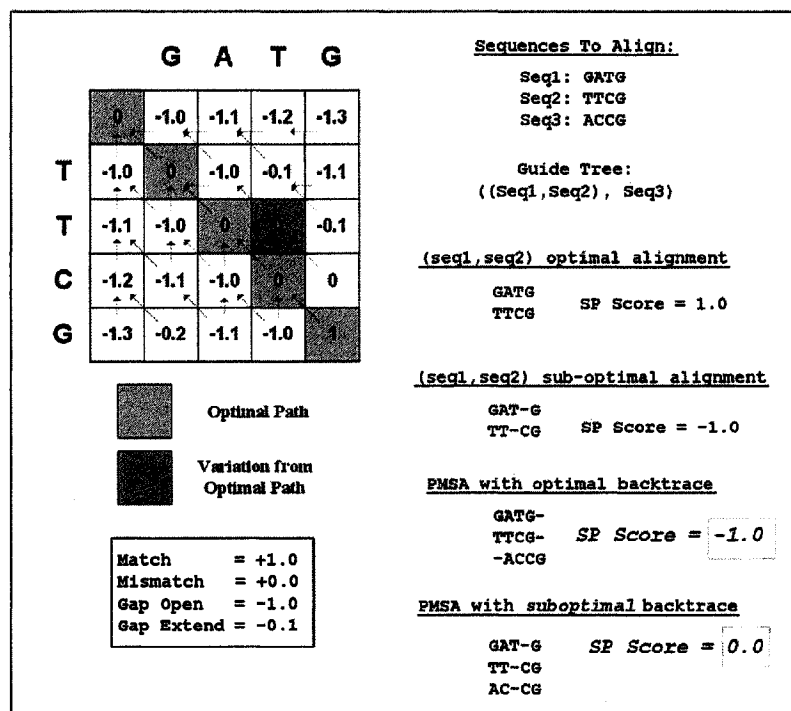


Figure 5.6: PMSA counterexample: Optimal DP leading to suboptimal progressive alignments. This example clearly depicts a key limitation in progressive multiple sequence alignment. We progressively align 3 short DNA sequence fragments. In one case, we produce the typical result by following the optimal DP backtracking path during the alignment of the first two sequences. In the other case, we vary slightly (in red) from the optimal backtracking path. By wandering off of the optimal backtracking path, we get a sub-optimal pairwise alignment of the first two sequences. Aligning the third sequence with the sub-optimal alignment of the first two sequences, we find a globally better final alignment. By using alignments dictated by locally optimal backtracking paths, we significantly constrain the search for the best possible overall sequence alignment. Future PMSA algorithms could also attempt sub-optimal backtracking paths near the optimal path for a more complete search.

5.8 Discussion

We demonstrated that dynamic programming ties are common in practice. In our simulation example, we showed that the frequency of DP ties increase as a function of the relatedness of the input sequences. In the case of simple pairwise alignments, dynamic programming ties result in a set of equally probable and equally optimal alignments. However, in the case of progressive multiple sequence alignment, the pairwise DP ties result in a combinatorial explosion of possible outputs. Since the way in which the sets of alignments are paired and aligned during the progressive alignment step impacts the final alignment output, these distinct outputs result in a distribution of alignment scores. Without information guiding the selection of which alignments to use at each progressive step, the globally optimal complete progressive alignment is almost never realized. We demonstrated how conventional, deterministic progressive alignment algorithms fall far short of finding the globally optimal progressive alignment, leaving considerable room for improvement to existing progressive alignment and dynamic programming approaches.

We also demonstrated an extension to the traditional DP approach that removes bias during stochastic backtracking. This modification to the common Needleman-Wunsch or Smith-Waterman algorithms is simple to implement and relatively inexpensive in both space and time.

It is typically too computationally expensive to enumerate all possible combinations of DP backtracking paths in order to identify the specific combination of paths resulting in the globally optimal progressive multiple sequence alignment. Breaking ties in an unbiased, stochastic manner is the only practical way to fairly explore the space of path combinations. Repeatedly and fairly sampling from the complete set of DP paths and constructing a sample distribution of alignment scores allows one to characterize the search space and choose paths which are notably better than the average alignment.

We demonstrated that deterministic programs with arbitrary tie-breaking mechanisms produce alignments that score significantly lower than the globally optimal progressive alignment.

Finally, we presented a concrete visual example of the core limitation of progressive multiple sequence alignment: the well known “once-a-gap, always-a-gap” phenomenon. Traditional progressive alignment algorithms are greedy in the sense that only optimal pairwise alignments are considered, while it is trivial to identify cases where *locally suboptimal pairwise alignments can be used to achieve superior overall alignments*. We recommend that future progressive

alignment algorithms sample from both optimal and slightly suboptimal backtracking paths in an attempt to identify globally superior alignments.

Chapter 6

Quantifying The Correlation of Alignment Accuracy and Guide Tree Quality

6.1 Preface

This chapter derives from a manuscript that remains unsubmitted for publication at the time of the writing of this report. We currently plan to submit this paper to the Pacific Symposium on Biocomputing (PSB '09). This chapter describes a large simulation study, empirically exploring the critical aspect of progressive multiple sequence alignment: the relationship of guide trees to alignment quality.

6.2 Abstract

Evolutionary biologists commonly use progressive multiple sequence alignment to identify positional homology in regions of molecular sequences. This progressive alignment approach infers a guide tree from input sequences and then aligns the sequences based on the order dictated by the guide tree. It is reasonable to assume that true phylogenies makes the best possible guide trees, resulting in alignments which best reconstruct the true positional homology of the input sequences. In this study, we use both real-world data and simulated data to empirically quantify and characterize the correlation between guide tree quality and alignment quality for several datasets. For both alignments and trees, we define quality as the proximity from the true alignment or tree. We find strong evidence that alignment quality is negatively correlated to the distance between a given guide tree and the true phylogeny. In many cases, this correlation appears logarithmic, indicating that the largest deltas in alignment

accuracy occur within a few steps of the true tree. Our results inform alignment algorithm designers about the advantage of constructing extremely accurate guide trees.

6.3 Introduction

Multiple sequence alignments represent the positional homology of molecular sequences. Nucleotides or residues are arranged by inferred homology into columns, while “gaps” are inserted to represent insertion/deletion (indel) events. The final output is often visualized as a rectangle, containing two or more molecular sequences with indel regions interspersed to correctly align columns and maximize the overall positional homology of the original input sequences.

Many researchers rely on the technique of progressive multiple sequence alignment (PMSA) in order to produce both fast and accurate multiple alignments. Commonly used PMSA programs include CLUSTAL W [2], T-Coffee [45] and MUSCLE [46]. All three of these programs infer a *guide tree* using fast and simple distance-based algorithms such as Neighbor-Joining [5] or UPGMA [47]. Once the guide tree is constructed based on the pairwise distances of the input sequences, the sequences are then aligned to each other in the order specified by the guide tree. A guide tree built by joining neighboring taxa in a pairwise distance matrix represents an ordering in which similar sequences are aligned prior to more distant sequences. Among other benefits, this approach limits the introduction of immutable gaps early in the alignment process, leading to relatively accurate alignments with an overall smaller number of gapped regions. Some PMSA algorithm developers claim that guide trees based on minimum pairwise edit distance produce more accurate alignments than guide trees based on the estimated evolutionary distance between taxa.

How important is guide tree selection to progressive alignment? Most PMSA algorithms quickly construct a distance-based guide tree and simply use that ordering. Some newer PMSA programs spend additional time refining guide trees to further optimize multiple alignments. MUSCLE initially infers a UPGMA tree and *draft alignment*, but subsequently uses this draft alignment to construct a Kimura-corrected [48] pairwise distance matrix that is the basis for an improved UPGMA-derived guide tree. MUSCLE further tries to refine the overall progressive alignment by iteratively and systematically re-rooting the guide tree. EVALYN [9] is a genetic

algorithm that iteratively performs crossover and mutation operations to optimize a population of guide trees. EVALYN measures the fitness of any individual guide tree by aligning the input sequences based on the guide tree and scoring the alignment using a sum-of-pairs (SP) metric.

While both MUSCLE and EVALYN spend significant compute resources to find guide trees that result in better overall multiple alignments, it is unclear if the gains in guide tree quality are worth the extra effort. MUSCLE performs relatively well in benchmark alignment tests [46], indicating that additional guide tree optimization may be measurably beneficial. However, questions remain concerning the maximum and expected gains from guide tree refinement as well as the correlation between guide tree quality and alignment quality.

Nelesen *et al.* [49] concluded that alignment quality and guide tree quality are *not* strongly correlated. While we agree with their results given their experimental design, their experimental design is limited in a number of key ways. Namely, their experiment is strictly a simulation study wherein they test the performance of a number of common PMSA programs under a small handful of similar methods for inferring guide trees. By contrast, our experimental design uses both biologically derived data as well as simulated sequences. We emphasize a novel, systematic, fine-grained approach to quantifying the quality of guide trees and its correlation to alignment quality.

We hypothesize that under PMSA, alignment quality is correlated to guide tree quality. While this correlation may seem obvious, the topic is largely absent in the literature. Quantifying the relationship between guide tree selection and alignment accuracy is an important ancillary goal.

6.4 Methods

This experiment systematically computes alignment accuracy as a function of guide tree quality. We quantify any found correlation and perform a hypothesis test by assessing the statistical significance of that correlation.

With progressive alignment algorithms, the output alignment \hat{A} is a function, $pmsa()$, of the input sequences S and the guide tree T .² Most progressive alignment algorithms infer T from S

² A progressive alignment is also a function of the alignment parameters such as gap costs and substitution matrix. This experiment uses default program parameters and we therefore exclude them in our function description.

prior to computing the alignment, but pre-computed guide trees can be provided as optional input:

$$\hat{A} = pmsa(S, T)$$

First, we define A_Q as the accuracy or quality of \hat{A} as compared against A^* using some arbitrary comparison function *aligncompare*:

$$A_Q = \text{aligncompare}(A^*, \hat{A})$$

A_Q is expressed as a normalized accuracy measurement in the interval $[0,1]$ of a test alignment with respect to a reference alignment. $A_Q=1$ indicates that the two alignments are identical, while a score of $A_Q=0$ means that \hat{A} and A^* share no common features under the distance evaluation function. The *aligncompare()* function can be implemented in a variety of ways. Minimum edit distance is an ideal measure of alignment similarity, but it is usually prohibitively expensive to compute for realistically sized datasets. Instead, we use both the sum-of-pairs (SP) accuracy rate and the total-column (TC) accuracy rate metrics.

Both SP-accuracy and TC-accuracy represent the percentage of correct alignment *features* in \hat{A} , as compared against the reference alignment A^* . The sum-of-pairs (SP) metric compares alignments by summing all correctly aligned nucleotide pairs in \hat{A} with respect to A^* , and then dividing that sum by the total number of nucleotide pairs in A^* . SP-error is the complement of SP-accuracy.

The total column (TC) score compares alignments at the column level by summing the correctly aligned columns in \hat{A} (given A^*) and dividing this integer value by the total number of columns in A^* . Since a column with a single nucleotide mismatch is considered incorrectly aligned, TC is obviously more sensitive than SP.

We define alignment distance A_{dist} as alignment *error* and it is the complement of alignment accuracy:

$$A_{dist} = 1 - A_Q$$

Similarly, we define tree distance T_{dist} in terms of the distance of a given tree T from the true tree, T^* :

$$T_{dist} = \text{treedistance}(T^*, T)$$

The function *treedistance()* returns a non-negative integer in the interval $[0, \infty)$, representing the edit distance between T and T^* . In our particular experimental design, we actually create guide trees at known edit distances from T^* , conveniently eliminating the need to estimate minimum edit distances.

Since we define *quality* as the proximity to the true alignment or tree, the goal of this experiment is to quantify the correlation between A_Q and T_{dist} under this distance-based definition of quality. Based on prior assumptions about progressive alignment, we expect A_Q to decrease as T_{dist} increases and therefore be negatively correlated.

6.4.1 Degrading Guide Trees

We iteratively and randomly apply tree edit operators to an initial guide tree T^* using either Nearest-Neighbor Interchange (NNI) or Tree Bisection and Reconnection (TBR) [36]. Every repeated application of NNI or TBR *degrades* the true tree by some known upper-bound estimate of edit distance.

The NNI operator swaps neighboring branches in a tree. Each internal branch of an unrooted bifurcating tree connects four other branches. These four branches are each other's nearest neighbors, connected to one another via the internal branch. NNI swaps nearest neighbors on internal branches of the given tree. For each random NNI operation, we randomly choose one internal branch and one pair of neighbors to interchange.

TBR bisects the guide tree at some branch, resulting in two distinct unrooted trees. The two unrooted trees are then reconnected at two reconnection points on both trees, resulting in a rearranged tree. Typically TBR tries all possible bisection and reconnection points between the two trees and evaluates that topology under some objective function. We define a single random TBR step to mean that we bisect the guide tree at one randomly selected branch and reconnect the two trees at a randomly chosen reconnection point on each tree. Notably, TBR is a strict superset of NNI, and TBR almost always introduces larger topological changes at each step.

For each input dataset, we independently degrade guide trees up to 50 edit operations from T^* , separately under both the NNI and TBR operators. We generate 100 guide trees at each of the 50 degradation levels. Every tree at every level is degraded independently, starting from T^* , to avoid autocorrelation.

The algorithm for the experiment is:

```

FOR EACH true tree  $T^*_i$  DO
  FOR EACH edit operator  $OP_j \in \{NNI, TBR\}$  DO
    FOR EACH degradation level  $L_k \in \{1, 2, \dots, 50\}$  DO
      FOR EACH replicate  $R_l \in \{1, 2, \dots, 100\}$  DO
         $T_{ijkl} \leftarrow OP_j(T_i)$ 
         $\hat{A}_{ijkl} \leftarrow pmsa(S_i, T_{ijkl})$ 
         $A_{Q,ijkl} \leftarrow aligncompare(A^*_i, \hat{A}_{ijkl})$ 
      DONE
    DONE
  DONE
DONE

```

CLUSTAL W (with default settings) aligns the input sequences, using every degraded guide tree as an input. The QSCORE program³ computes the SP and TC accuracy scores by comparing the alignments derived from every degraded guide tree with the known true alignment. A_Q is simply the SP-accuracy or TC-accuracy score returned from QSCORE. Results are averaged across all 100 replicates for each guide tree degradation level. Since T_{dist} is explicitly known for every T and T^* , we can easily compute the correlation between A_Q and T_{dist} for all alignments and guide trees in the experiment. For the simulated studies, we replicate this experiment five times using different randomly generated starting true trees and simulated true alignments.

6.4.2 Experimental Input Data

This experiment uses naturally evolved trees and alignments as well as sequences and trees simulated under evolutionary parameters estimated from those natural data. Naturally evolved molecular sequences rarely provide us with known true alignments and/or trees. However, sequences simulated over known trees conveniently provide both true trees and true alignments, enabling the explicit definition of *quality* in terms of *distance* from the known “truth”.

Simulated evolution involves simplistic pseudo-stochastic models of molecular sequence evolution. Because of this, there is inherent risk with all simulation studies that attempt to generalize an algorithm’s behavior from simulated to natural cases. For this reason, this experiment analyzes data from both natural and simulated evolution. To better compare results from each kind of data, we directly estimate the simulation model parameters from the natural datasets.

³ QSCORE is available from R.C. Edgar at www.drive5.com

6.4.3 TreeBASE Alignments and Trees

We used TreeBASE [50] to identify naturally evolved sequences with published alignments and phylogenies. TreeBASE is an annotated online database of peer-reviewed phylogenies and their corresponding sequence alignments. We arbitrarily chose three distinct TreeBASE studies containing sufficiently challenging alignments and trees.

In this article, we refer to these datasets by their TreeBASE matrix accession number, thereby allowing easy lookup via TreeBASE of the specific NEXUS file containing both the putative alignment and phylogeny. The first dataset, M2045 [51] contains both the 22-taxon alignment from fungal 18S ribosomal RNA and the corresponding maximum likelihood tree. A 16-taxon dataset, M3016 [52], contains longer, chloroplast sequences and a putative maximum parsimony phylogeny. The M2783 dataset [53] is a pathogenic plant fungus study, consisting of an alignment of 33 short, well-conserved 5.8S ribosomal RNA sequences. Interestingly, the published maximum likelihood tree derived from this alignment is highly pectinate, as the sequences represent sequentially derived *Ceratobasidium* strains.

Table 6.1: The selected TreeBASE datasets.

TreeBASE Matrix Accession #	# Taxa	Average Sequence Length	Alignment Length	% ID	Gene	Method	Comment
M2045	22	2618	2723	88.63%	18S ribosomal RNA	Maximum Likelihood	fungus
M3016	16	3695	3977	88.84%	rbcl (Chloroplast)	Parsimony	plant
M2783	33	562	605	96.70%	5.8S ribosomal RNA	Maximum Likelihood	Fungus Plant pathogen

These alignments and trees in TreeBASE are computed using conventional methods such as progressive alignment, maximum parsimony, and maximum likelihood. Both the alignments and trees probably contain errors. However, for the purposes of this study, we treat the TreeBASE alignments and trees as true and correct. This allows us to use the TreeBASE data as a reference set against which we can measure accuracy in our experiments. In order to test our

methods with known true alignments and phylogenies, we simulate sequences and alignments over known trees.

6.4.4 Simulated Sequences and Trees

Simulated evolution concurrently provides the precise true tree and the precise true alignment. We used ROSE [44] to simulate molecular evolution based on model parameters directly estimated from the three TreeBASE studies. We chose to base the simulation parameters on the three TreeBASE datasets in order to ground our simulations in reality and better contrast any differences between the simulated cases and the non-simulated cases.

For each TreeBASE dataset, we provide ROSE with estimates of sequence length, equilibrium nucleotide frequencies, indel probabilities, indel size probability distributions, as well as 16-parameter, time irreversible nucleotide substitution matrices. For each of the three TreeBASE datasets, ROSE simulated 5 replicate datasets, each with its own stochastically generated phylogeny. ROSE generated unique random phylogenies for all five replicates, but these starting trees were identical across TreeBASE cases. In other words, there were only five simulated starting trees in the whole experiment, but each of these 5 starting trees was used in the M2045, M3016, and M2783 based simulations.

6.4.5 Measuring Alignment Quality

The ultimate goal of multiple sequence alignment is to accurately reconstruct the series of insertion, deletion, and mutation events that occurred via evolution. For most naturally occurring molecular sequences, we can never know the true alignment with absolute certainty. Using simulation, however, we know the true alignment and can use various metrics to assess the distance between any alignment and the true alignment. This distance from the true alignment is the best, least-biased measure of alignment quality.

We use the QSCORE program to measure the distance between reference alignments and true alignments. QSCORE uses four comparison functions and returns normalized accuracy estimates, valued between 0 and 1.

6.5 Results

Overwhelmingly, we found a strong relationship between alignment accuracy and amount of tree degradation across all three TreeBASE-derived datasets and all replicates as shown in Figures 6.1-6.3. This apparent correlation was found in all but a small minority of cases.

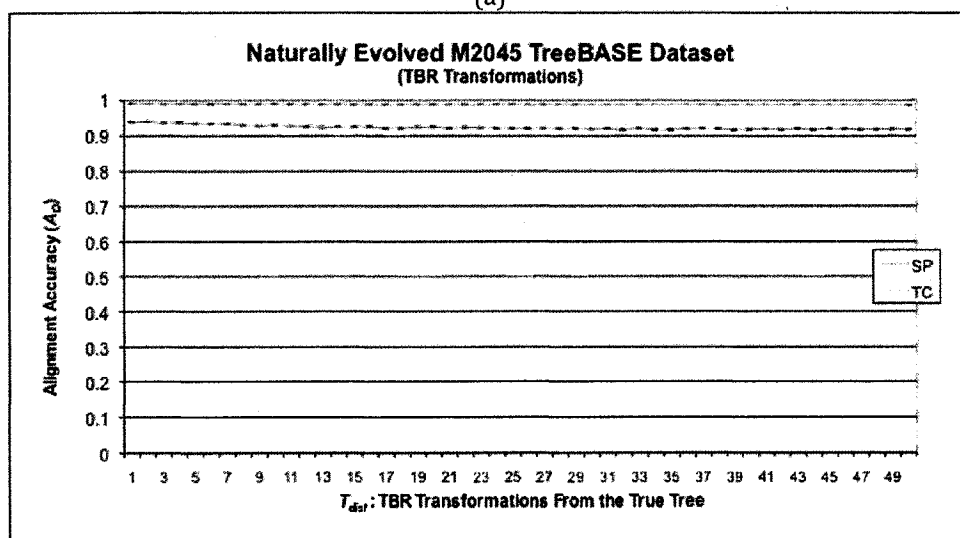
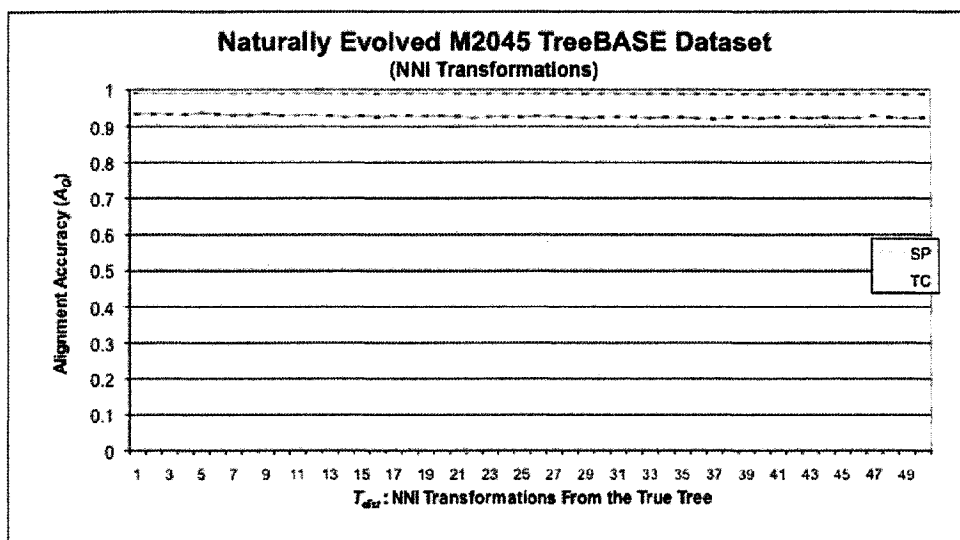


Figure 6.1: Naturally evolved M2045 fungal 18S rRNA TreeBASE study results. These results demonstrate that degrading the guide trees up to 50 random independent NNI and TBR operations has little absolute effect. However small the effect, the correlation between alignment accuracy score (A_Q) and distance from the true tree (T_{dist}) remains strong and statistically significant. *SP* represents the Sum-of-Pairs accuracy metric, while *TC* represents the Total Column accuracy metric.

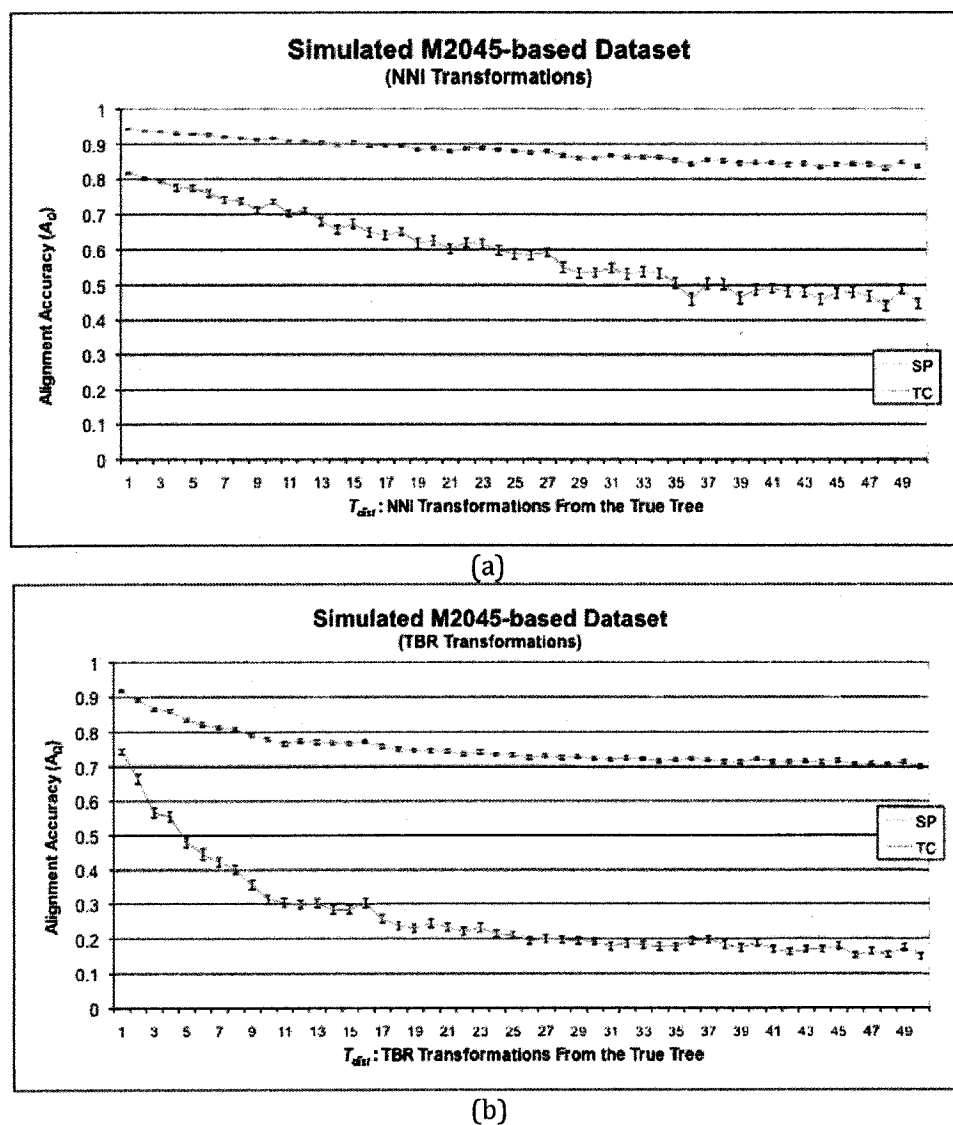


Figure 6.2: Results of simulations based on M2045 fungal 18S rRNA TreeBASE study. In these representative simulated results, the effect of tree degradation on alignment quality is much more pronounced than with the naturally evolved datasets. The correlation between A_Q and T_{dist} is significant, negative, and non-linear. Error bars represent \pm one standard error from the sample mean. *SP* represents the Sum-of-Pairs accuracy metric, while *TC* represents the Total Column accuracy metric.

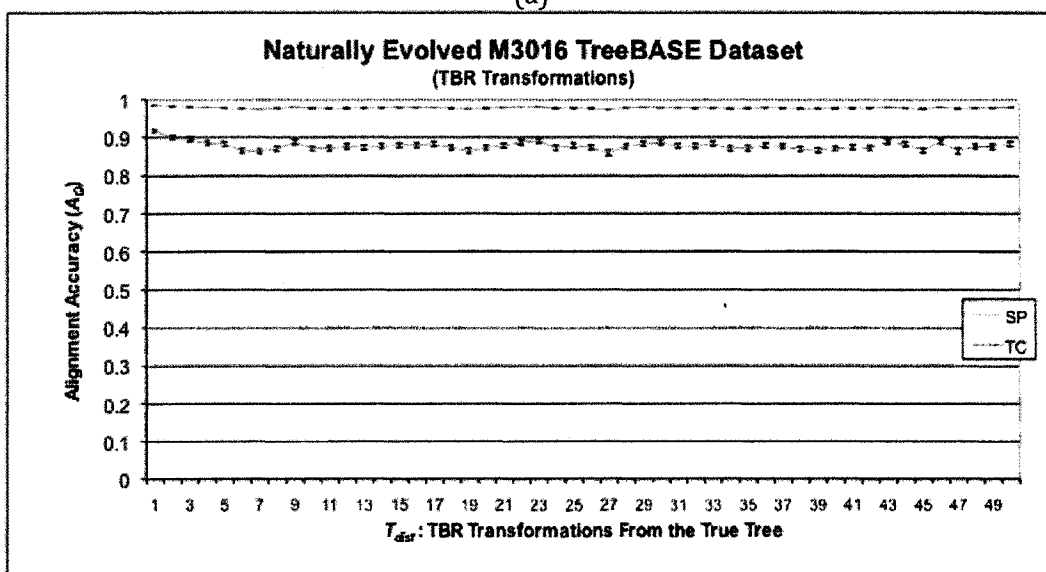
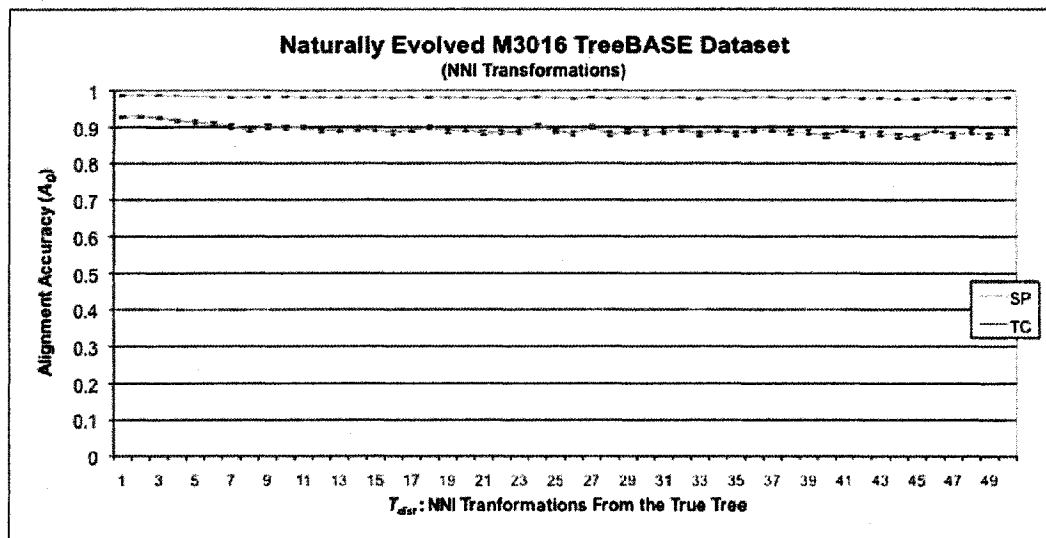
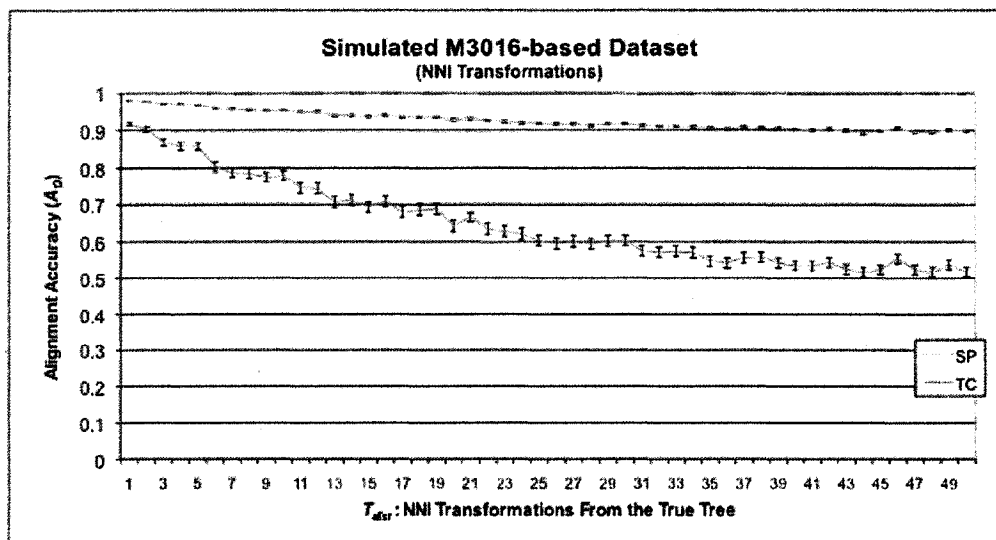
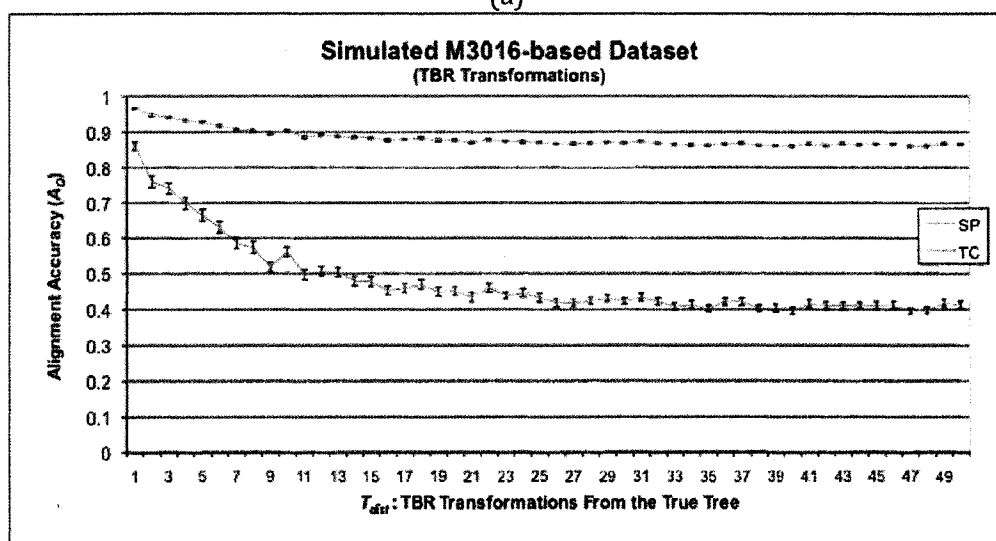


Figure 6.3: Results for the natural M3016 *rbcl* Chloroplast study. This figure shows that degrading the guide trees up to 50 random independent NNI and TBR operations has little absolute effect for the naturally evolved alignments and trees shown in (a) and (b). In (b), the correlation between T_{dist} and A_Q is *not* statistically significant. *SP* represents the Sum-of-Pairs accuracy metric, while *TC* represents the Total Column accuracy metric.



(a)



(b)

Figure 6.4: Results for the simulated M3016 *rbcL* Chloroplast study. Alignment accuracy scores in the simulated cases demonstrate a notably higher overall sensitivity and correlation to T_{dist} than in the natural cases shown in Figure 6.3. *SP* represents the Sum-of-Pairs accuracy metric, while *TC* represents the Total Column accuracy metric.

Alignment accuracy dropped faster with TBR degradation than with NNI degradation due to the relatively higher destructiveness of the TBR operator. Also, the net measured effect of tree degradation was significantly more pronounced under the TC-accuracy metric compared to SP-accuracy. This also was anticipated, due to the relatively higher sensitivity of TC to differences between alignments.

The overall impact of guide tree degradation on alignment accuracy was relatively small, especially as measured under SP-accuracy. Across our entire study, we found that guide tree degradation had a median effect of less than 5% on alignment scores with SP, and a median of 25% effect under the more sensitive TC metric.

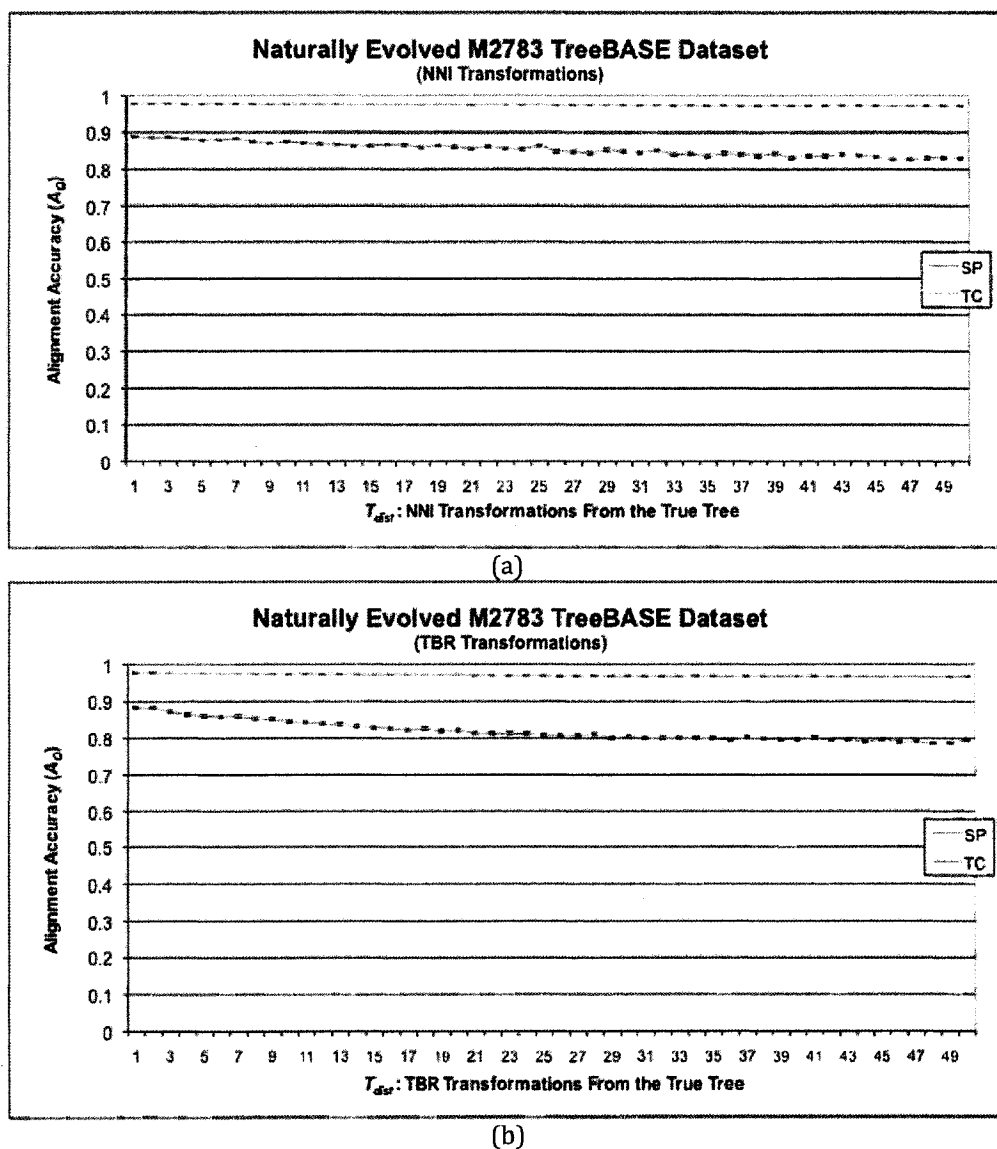
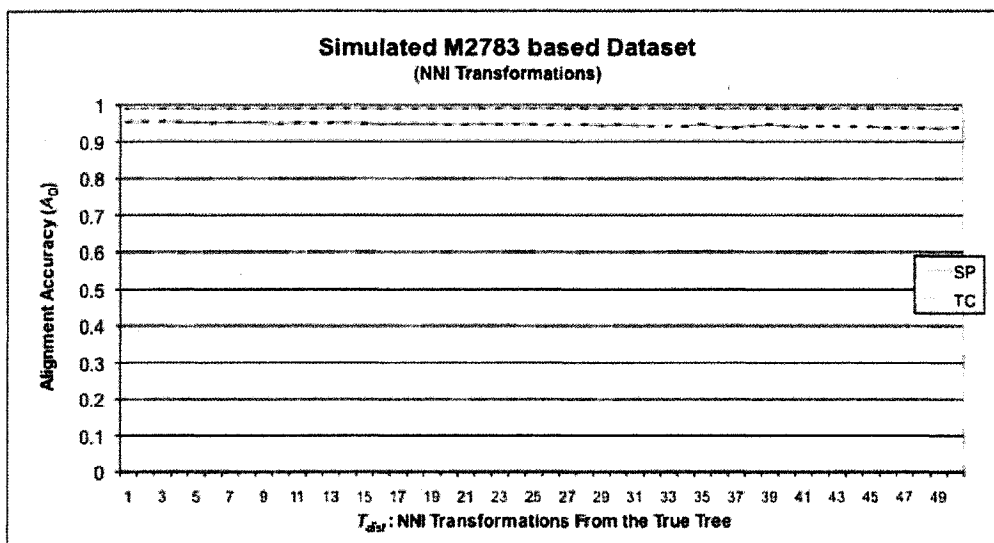
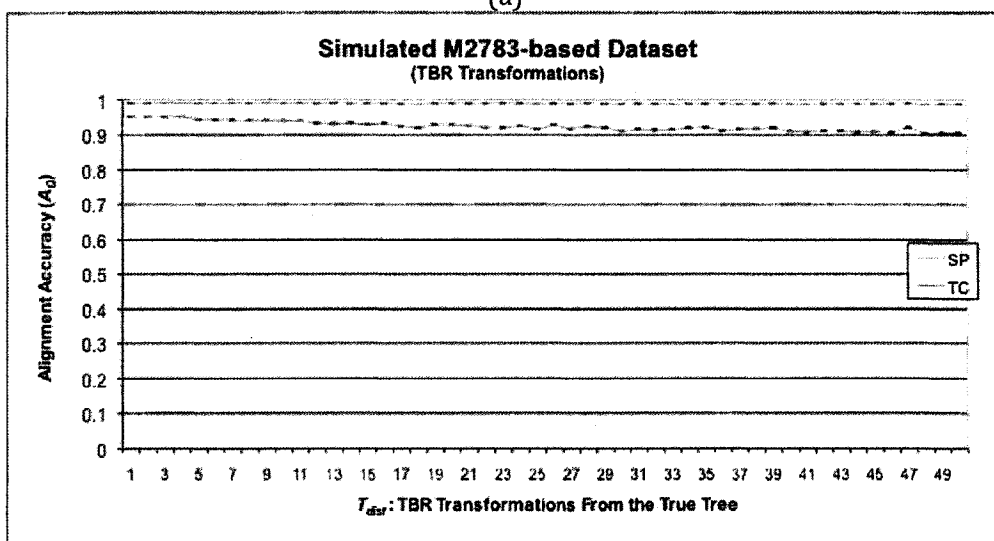


Figure 6.5: Results using the shorter 5.8S rRNA M2783 fungal pathogen study (natural). With this dataset, the absolute effect of guide tree degradation is very small, yet statistically significant. These shorter sequences are highly similar with a 96.7% identity and a very pectinate true tree. *SP* represents the Sum-of-Pairs accuracy metric, while *TC* represents the Total Column accuracy metric.



(a)



(b)

Figure 6.6: Results using the shorter 5.8S rRNA M2783 fungal pathogen study (simulated). With this dataset, the absolute effect of guide tree degradation is very small, yet statistically significant. These shorter sequences are highly similar with a 96.7% identity and a very pectinate true tree. *SP* represents the Sum-of-Pairs accuracy metric, while *TC* represents the Total Column accuracy metric.

This small effect, shown in Figures 6.5 and 6.6, is inline with the results presented by Nelesen *et al.* Alignment accuracy as a function of tree degradation is apparently non-linear. In fact, it appears to be logarithmic in nature, indicating a larger measurable effect closer to the true tree. Since the Nelesen experiment used a handful of similar guide trees (*e.g.* those inferred

with UPGMA, Neighbor-Joining, etc.), this may contribute to their conclusion that there is no strong correlation between alignment and guide tree quality.

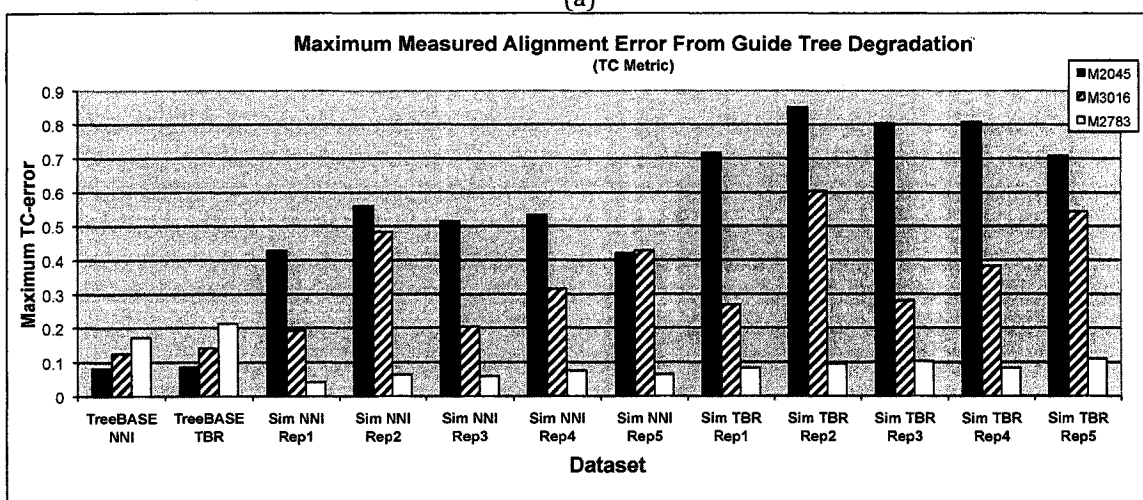
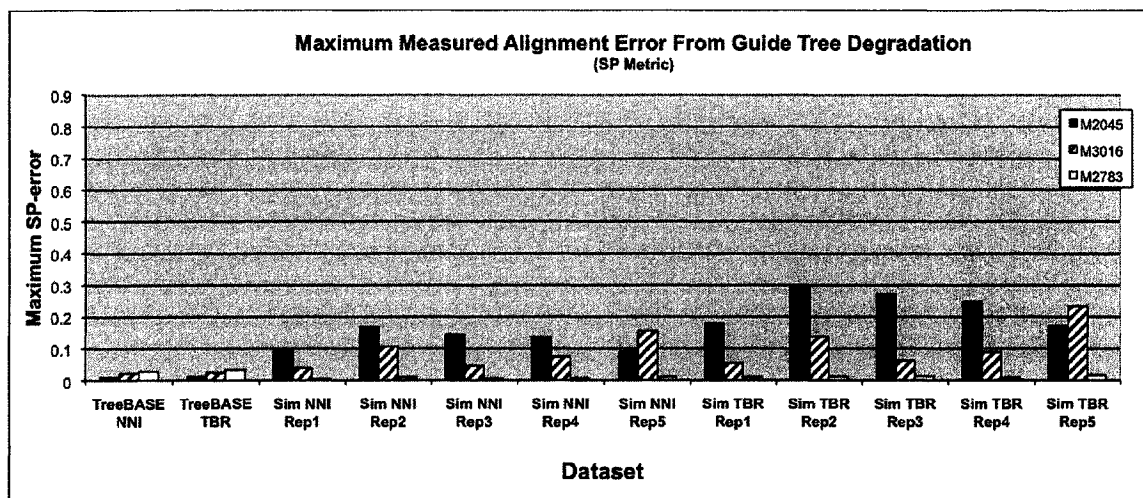


Figure 6.7: The largest measured effect for each dataset for both SP-error (a) and TC-error (b). As expected, the overall effect of degrading the guide tree is largest as measured by the more sensitive TC metric. Also, the magnitude of the effect is largest when degrading trees with the more disruptive TBR operator. Unexpectedly, simulated data sets are generally more sensitive to poor guide trees.

We found a strong correlation in most experimental runs, despite the small overall effect of guide tree selection on alignment accuracy.

The datasets derived from the M2783 alignments and trees were far less sensitive to guide tree degradation as shown in Figures 6.5 and 6.6. Yet even this dataset indicated a statistically significant correlation. These 5.8S rRNA sequences are highly conserved with 96.7% sequence identity. We hypothesize that since the input sequences start so close to the true alignment, the effect of guide tree selection is significantly reduced. Other factors may also explain this result: this dataset had more taxa and shorter sequences than the others.

Unexpectedly, tree degradation produced a larger measurable effect on alignment quality with the simulated cases than with the naturally evolved cases. This is most apparent in the M2045 and M3016 derived datasets.

6.5.1 Significance of Correlation

Precisely stated, we hypothesize that alignment accuracy decreases as the distance of the guide tree from the true tree increases. Since our figures indicate a non-linear correlation, we use the Spearman rank correlation coefficient and formulate a statistical hypothesis test under the null hypothesis that this correlation coefficient is 0:

$$H_0: r = 0$$

$$H_a: r \neq 0$$

While the T_{dist} values are always in ranked order, A_Q scores were ranked in descending order. We computed the Spearman coefficient r on these rankings and computed t as:

$$t = r \sqrt{\frac{n-2}{1-r^2}}$$

We then performed the appropriate t test for each of these coefficients at $p=0.01$ with 48 degrees of freedom. The results under both the SP and TC metrics are shown in Figure 6.5.

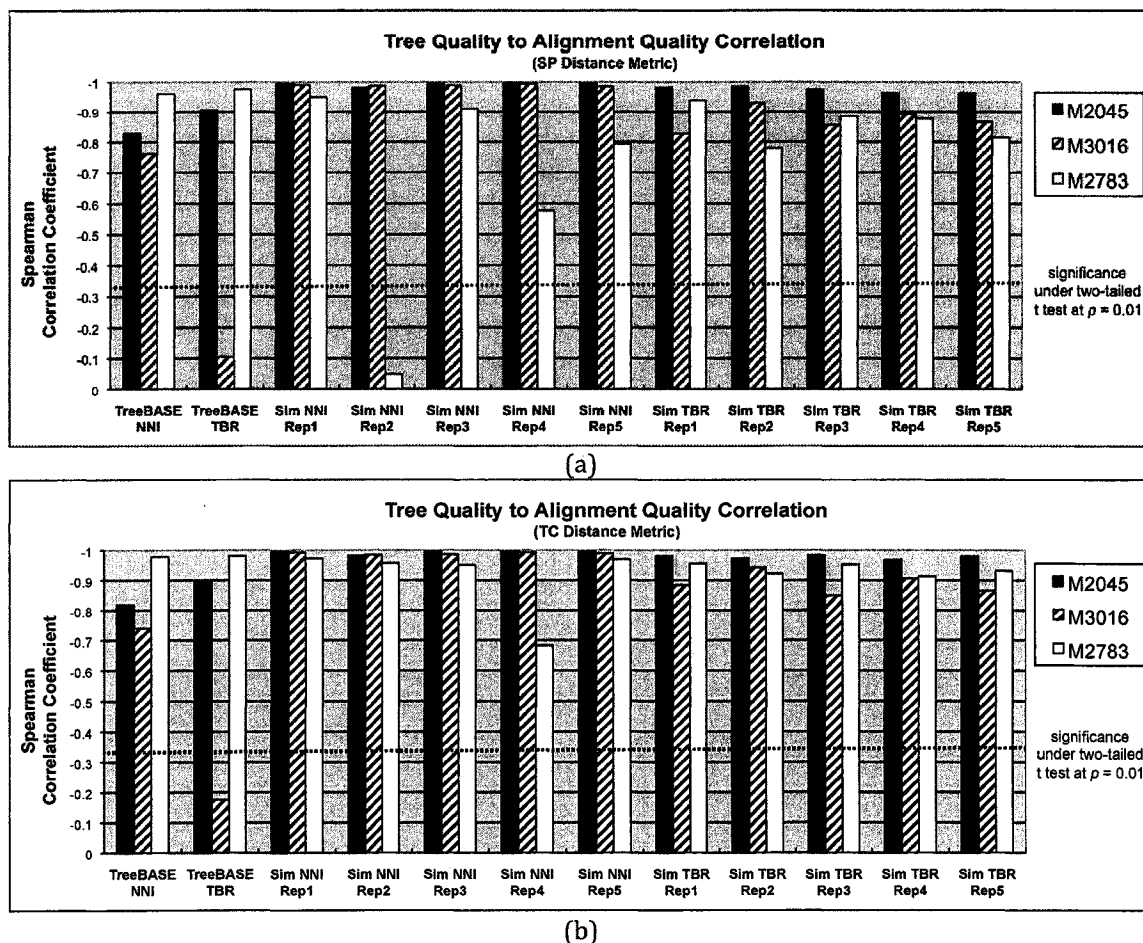


Figure 6.8: The Spearman rank coefficient for alignment quality and tree degradation. Most datasets show a strong negative correlation. The horizontal line represents the value under which we reject the null hypothesis. We clearly reject H_0 for the large majority of datasets in this study.

6.6 Discussion

Across all simulated and biological datasets, we found a strong, statistically significant correlation between guide tree quality and alignment quality. While TC scores were more sensitive than SP scores to the use of degraded guide trees, our overall conclusion is independent of our choice of alignment distance metric. Furthermore, our conclusions are independent of tree transformation operator.

In many cases, especially those with the more disruptive TBR operator, the effect of guide tree quality to alignment quality appears non-linear and logarithmic in nature. In other words,

the largest effect on alignment quality almost always occurs within 5-10 steps from the true tree, and then gradually tapers off. This result may be of interest to PMSA algorithm developers, as it informs us as to the relative importance of choosing a good guide tree. It may often be worth the additional computational time to optimize a good guide tree even further. We see measurable performance gains for exactly this reason from programs such as MUSCLE that infer a quick guide tree using distance methods and then improve upon the initial tree to produce better alignments.

Superficially, our results contradict the findings in Nelesen *et al.*, where they performed a similar study looking for correlation between guide tree quality and alignment quality. Contrary to our results, they showed that there was no strong correlation between alignment quality and guide tree quality.

We do ultimately refute Nelesen *et al.*, concluding instead that there *is* a significant correlation between alignment quality and guide tree quality. However, we do agree with Nelesen that the overall effect of guide tree selection to alignment quality is usually very small, especially as measured under the SP metric. We also agree that the choice of alignment algorithm and associated parameters is usually more important than the choice of guide tree. Ultimately, the Nelesen experiment is able to show that the effect of guide tree selection on alignment quality is very small, but their method is not sensitive enough to actually quantify the correlation of the subtle effect.

First, our study presents a more systematic approach for generating guide trees at different, measurable distances from the true tree. By contrast, Nelesen *et al.* used a handful of distinct, but similarly generated guide trees and then compared the effect of using these trees on overall alignment scores. Second, their study used only the SP metric, which is not as sensitive to errors as the TC metric. Our study uses both metrics. While TC produces much larger, measurable effects, we found similar correlation results independent of the method for measuring alignment distance.

We did find that the overall effect of guide tree selection on alignment quality is relatively small under the SP metric. Therefore, we broadly concur with Nelesen *et al.* that "... despite the large differences in topological accuracy of the guide trees, alignment accuracy (measured using SP-error) for a particular alignment method varies relatively little between alignments estimated from different guide trees." Across our entire study, we found that guide tree degradation had a median effect of less than 5% on alignment scores under SP, and a median of 25% effect under the more sensitive TC metric. However, small effects do not preclude a

correlation and we did find a statistically significant nonlinear correlation between guide tree quality and alignment quality.

For at least two datasets (M2045 and M3016), there was a large difference in sensitivity to guide tree selection between the simulated and non-simulated cases. This sensitivity difference was less noticeable in the M2783-derived datasets. Determining why simulated datasets seem more sensitive than non-simulated datasets to this experimental approach is an area of potential future work.

The overall net effect of guide tree degradation in the M2783 TreeBASE study was significantly smaller than the other studies. In this dataset, sequences in the true alignment were highly conserved with a 96.7% sequence identity. We initially conclude that highly conserved molecular sequences are far less sensitive to guide tree selection under PMSA. Our general experimental method lends itself to a future systematic analysis of the correlation between percent identity and guide tree sensitivity under PMSA.

6.7 Future Work

We would like to extend this experimental method to other PMSA programs such as MUSCLE, MAFFT [54] and T-Coffee. This would help exclude any kind of implementation-specific bias to our results.

The distance of a progressive alignment to the true alignment seems to be a nonlinear, logarithmic function of the distance of the guide tree to the true tree. Determining a closed-form description of this relationship may prove advantageous.

We found evidence that progressive alignment with well-conserved input sequences is relatively insensitive to guide tree topology. This result was largely anticipated, since the input sequences are initially closer to the true alignment. However, other variables may affect the amount of overall sensitivity to guide tree selection including sequence length, nucleotide frequencies, number of taxa, and topology of the true tree. We propose a broader future study quantifying the correlation between these variables and sensitivity to guide tree selection. An adaptive PMSA algorithm might pre-compute these sequence statistics to estimate the optimal amount of time to spend on guide tree refinement.

Appendix A

Relaxed Neighbor-Joining: A Fast Distance-Based Phylogenetic Tree Construction Method

- [27] Evans, J., L. Sheneman, and J.A. Foster (2006) Relaxed Neighbor-Joining: A Fast Distance-Based Phylogenetic Tree Construction Method, *J. Mol. Evol.* 62:785-792.

A.1 Preface

As second author of this paper, I did not include this article as a chapter in this report. As co-author and significant contributor, I included it as an appendix.

The novel Relaxed Neighbor-Joining algorithm stemmed from collaborative work with Jason Evans of the University of Idaho. While working on optimizing the performance of the classic Neighbor-Joining algorithm, Evans invented Relaxed Neighbor-Joining. The Clearcut program (Chapter 4) implements Relaxed Neighbor-Joining. Both the Clearcut program and the Clearcut publication ultimately came from the algorithm design work described in this paper.

A.2 Abstract

Our ability to construct very large phylogenetic trees is becoming more important as vast amounts of sequence data are becoming readily available. Neighbor-Joining (NJ) is a widely used distance-based phylogenetic tree construction method that has historically been considered fast, but it is prohibitively slow for building trees from increasingly large datasets. We developed a fast variant of NJ called Relaxed Neighbor-Joining (RNJ) and performed experiments to measure the speed improvement over NJ. Since repeated runs of the RNJ algorithm generate a superset of the trees that repeated NJ runs generate, we also assessed tree

quality. RNJ is dramatically faster than NJ, and the quality of resulting trees is very similar for the two algorithms. The results indicate that RNJ is a reasonable alternative to NJ and that it is especially well suited for uses that involve large numbers of taxa or highly repetitive procedures such as bootstrapping.

A.3 Introduction

The Relaxed Neighbor-Joining (RNJ) algorithm is a distance-based phylogenetic tree construction method that is similar to the Neighbor-Joining (NJ) algorithm [5,29]. RNJ and NJ have the desirable property that they are consistent estimators of phylogeny if the distances in a dataset are purely additive [28]. An additive dataset is one for which there exists a tree with nonnegative branch lengths that perfectly represents all of the pairwise distances. In practice, datasets are rarely additive, but NJ still works well as long as distances are nearly additive [55,56]. NJ is one of the more commonly used tree construction methods, and in most cases it generates useful results. NJ tree construction requires pairwise distances as input. Users of NJ typically start by calculating the magnitudes of differences between DNA sequences for the taxa under consideration. Those magnitudes are then treated as distances, and the NJ algorithm attempts to construct a tree that encodes all of the pairwise distances. The quality of results depends heavily on the accuracy of the distances, and several researchers have addressed this issue. At the most basic level, distances can be corrected to account for multiple mutations at the same DNA site [48]. Felsenstein uses a likelihood-based model for calculating distances [36]. The Weighbor variant of the NJ algorithm reweights distances in an attempt to improve results when distantly related taxa are included in the dataset [57]. The BIONJ variant of NJ takes into account the variances and covariances of the distances and minimizes these at each step during tree construction [58]. NJ is generally regarded as a fast reconstruction method, but its runtime complexity is $O(n^3)$, which means that for large datasets, reconstruction is impractical. Mailund and Pedersen developed QuickJoin [31], which implements a heuristic method that avoids considering pairs of nodes when they are known to fall outside bounds that are calculated from previous passes through the matrix of pairwise distances. Runtime results are good for this heuristic, as is shown later, but substantial extra space is required for auxiliary data structures, which limits QuickJoin to approximately 8000 taxa on modern 32-bit computer systems. As such, this heuristic approach is only compelling for a limited range of input sizes.

Typical NJ implementations do not make use of such heuristics. Their runtimes are proportional to n^3 for all inputs, and they require space proportional to n^2 . RNJ typically requires time proportional to $n^2 \lg n$, without using any more space than NJ. Unlike simpler distance-based tree construction methods, NJ is able to deal with varying rates of evolution, so the resulting trees need not be ultrametric. This is accomplished by making join decisions based on transformed distances that take all taxa into consideration. A transformed distance T_{AB} between taxon A and taxon B is calculated as

$$T_{AB} = D_{AB} - \frac{\sum_{i=1, i \neq B}^n D_{Ai} + \sum_{i=1, i \neq A}^n D_{Bi}}{n-2}$$

where D_{xy} is the distance between taxon x and taxon y , and n is the total number of taxa. The fractional sums represent the average distances from A and B to all other taxa. Note that the divisor is $n-2$ because D_{AA} and D_{BB} are always zero, and D_{AB} is excluded, which means that two distances are effectively excluded from each sum. There are two components that contribute to a minimal transformed distance. A small absolute distance between A and B contributes, but it is also important that, on average, A and B are farther from other taxa than they are from each other. By joining two taxa with globally minimal transformed distance between them at each step of tree construction, NJ builds the tree starting from the leaf nodes, without risk of joining taxa that are not immediate neighbors. Figure A.1 helps to illustrate why this is so. D_{CD} is the minimum pairwise distance, but T_{AB} and T_{EF} are the minimal transformed distances. This is because C and D are closer to the center of the tree than are the taxa with minimal transformed pairwise distances. We modified the NJ algorithm in order to improve tree construction speed, with the additional goal of maintaining the quality of generated trees. This paper describes the algorithmic modifications that are the basis of RNJ, then presents experiments that measured tree construction speed and tree quality. The experimental results indicate that the RNJ algorithm is very fast, and that RNJ trees are of very similar quality to NJ trees

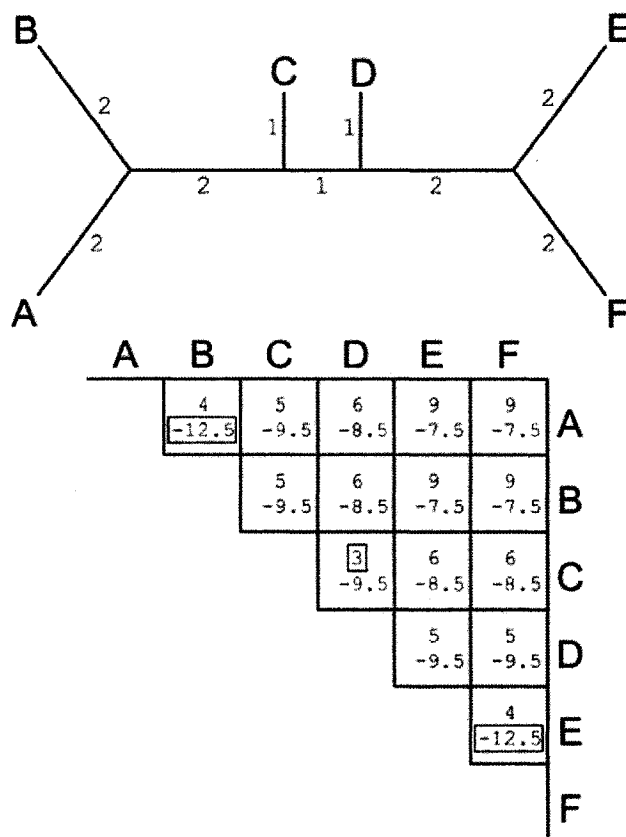


Figure A.1: Phylogenetic tree and corresponding patristic distance matrix. Each matrix cell contains the absolute pairwise distance (top) and transformed distance (bottom). D_{CD} is the minimum pairwise distance for this tree, but the associated transformed distance T_{CD} is not minimal; T_{AB} and T_{EF} are minimal. The NJ algorithm will first join A and B or E and F .

Like NJ, RNJ uses transformed distances when making join decisions, but rather than looking for a minimum among all transformed distances, RNJ looks for two taxa that have minimal transformed distance between them as compared to their transformed distances to all other taxa. Whereas NJ only joins pairs of neighboring leaf nodes that are minimally distant, RNJ can join any pair of plausible neighboring leaf nodes. There are typically many more pairs of neighboring leaf nodes than there are neighboring leaf nodes with minimal transformed distance, and RNJ is usually able to find such pairs without having to calculate transformed distances for all taxon pairs. We refer to the algorithm as “relaxed” NJ because it does not search for the globally minimal transformed distance; RNJ employs a less stringent, relaxed join criterion. Following is a general description of the RNJ algorithm, which operates on an input matrix of pairwise distances:

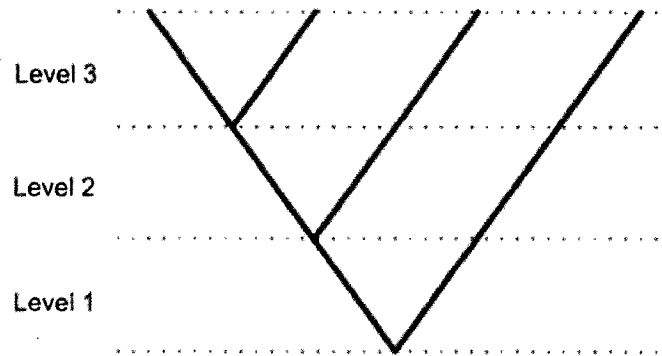


Figure A.2: A pectinate tree with $x = 4$ leaf taxa. The tree has $x - 1 = 3$ levels, and branches span from 1 to $x - 1 = 3$ levels.

Where \mathfrak{R} is the set of rows in the matrix of pairwise distances:

1. While $|\mathfrak{R}| > 2$:
 - (a) Choose a row $A \in \mathfrak{R}$.
 - (b) Choose a row $B \in \{\mathfrak{R} \mid B \neq A\}$.
 - (c) Calculate the set of transformed distances $T_A = \{T_{AR} \mid \forall R \in \{\mathfrak{R} \mid R \neq A\}\}$
 - (d) Calculate the set of transformed distances $T_B = \{T_{BR} \mid \forall R \in \{\mathfrak{R} \mid R \neq B\}\}$
 - (e) If $T_{AB} \in \{\min(T_A)\}$ and $T_{BA} \in \{\min(T_B)\}$
 - i. Create a new node X , and join it to the nodes represented by A and B .
 - ii. Remove A and B from \mathfrak{R} .
 - iii. Insert a row that corresponds to node X into \mathfrak{R} .
2. Join the nodes represented by the remaining two rows.

There is a situation in which RNJ could mistakenly join two nodes, unless an additional check is performed. For example, nodes C and D in Figure A.1 could be mistakenly joined, since T_{CD} is minimal compared to the transformed distances in the matrix rows and columns that correspond to nodes C and D . However, there is a simple way to always recognize and avoid such errant joins when distances are additive. (For non-additive distances, conflicting data lend partial support to such joins, so they are not necessarily in error.)

Consider that if the path $A \leftrightarrow B$ includes an internal branch, there exists a node R such that the internal branch is a component of the path $A \leftrightarrow R$. In such a case, joining A and B would remove the internal branch, which would change T_{AR} . In fact, the way that branch lengths are calculated during joins changes the distance from A to all other nodes (except B) if there should be an internal branch between A and B . Therefore, when distances are additive, errant joins can be avoided by making sure that T_{AR} does not change for an arbitrary choice of R other than B .

RNJ's algorithmic complexity is $O(n^3)$, but typical performance is much better. The runtime of RNJ tree construction is primarily determined by the number of transformed distances that must be calculated.

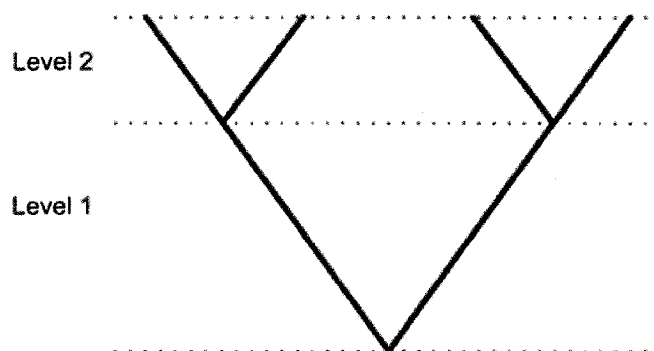


Figure A.3: A perfect tree with 2^{x-2} leaf taxa. The tree has $x = 2$ levels, and all leaf taxa are $x = 2$ branches away from the root.

There is some probability that step (1a) of the RNJ algorithm will choose a taxon that has an immediate neighbor. That probability ranges from $4/n$ to 1; it is minimal for pectinate (maximally deep) trees such as that in Figure A.2 and maximal for perfect (fully balanced) trees such as that in Figure A.3. For the extreme case of pectinate trees, RNJ typically affords only a constant (though substantial) speedup over NJ, but for trees that are even somewhat balanced, RNJ performs approximately proportional to $n^2 \lg n$.

A.4 Experiment

We performed three experiments, which looked at (1) algorithmic correctness, (2) speed, and (3) quality of results. The correctness and speed experiments used additive distance matrices as input, in order to make validation possible and performance comparisons fair.

The experiments used four distinct tree shapes, and branch lengths were based on random sampling from the gamma distribution. The gamma distribution was chosen because it provides a simple mechanism for generating trees with varying degrees of branch length variation, where all branches have nonnegative lengths. In all cases, we chose parameters such that

$\lambda = \frac{1}{\alpha}$ where λ is the scale parameter and α is the shape parameter, so that the expected value was always 1. We chose values of $\alpha \geq 2$, so that all distributions had the same basic shape. Specifics of how trees of the four shapes were generated follow: Pectinate. A pectinate tree with x leaf taxa has $x-1$ levels. For a given x , there is only one such tree shape, assuming unordered node adjacencies (Figure A.2). Each branch is assigned a length by summing the results of iteratively multiplying some constant branch length scalar C with a number that is drawn from the gamma distribution: $\sum_{i=L_{min}}^{L_{max}} C \times D_i$, where $D_i \sim \Gamma(\alpha, \lambda = 1/\alpha)$. L_{min} and L_{max} are the minimum and maximum levels that the branch spans. The root node is implicit, so the branch that contains the implicit root has a length that is the sum of that branch's implicit component branches.

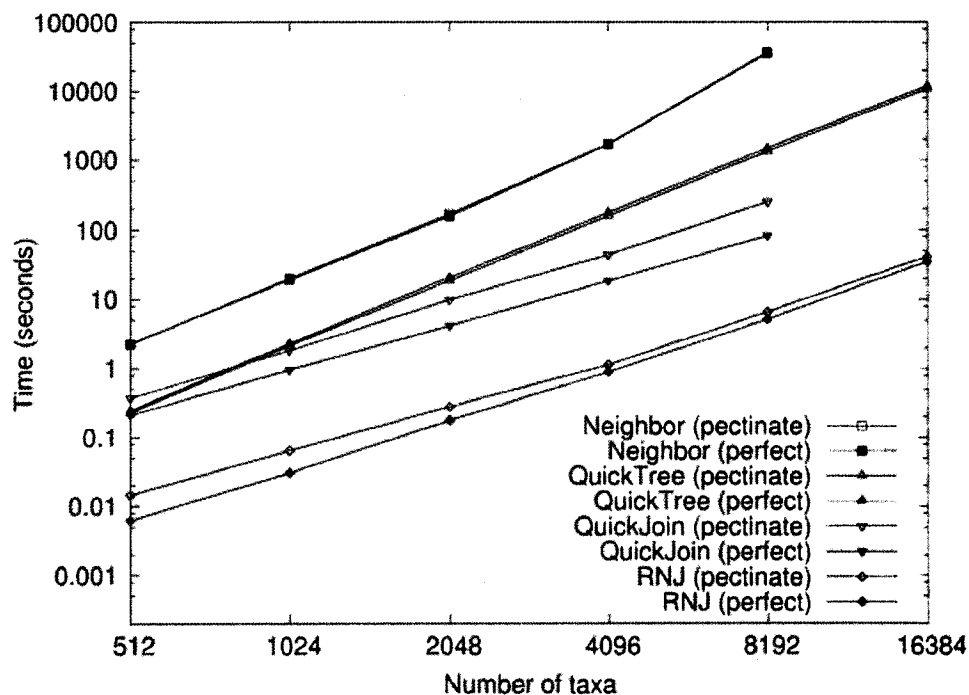


Figure A.4: Speed comparison of NJ/RNJ tree construction for two tree shapes (pectinate and perfect) and varying numbers of taxa. RNJ vastly outperforms the NJ implementations of QuickJoin, QuickTree, and PHYLIP neighbor for all tree shapes and sizes.

Trezilla

Trezilla trees are based on a single 500-taxon tree that is based on data from Chase *et al.* [59].

These trees differ only in their branch lengths. Each branch is assigned a length by multiplying some constant factor with a number that is drawn from the gamma distribution: $C \times D$, where $D \sim \Gamma(\alpha, \lambda = 1/\alpha)$.

Random

A random tree is generated via star decomposition. Each branch is assigned a length using the same procedure as for treezilla trees.

Perfect

A perfect tree always has $2x$ leaf taxa, for some positive integer x . There are x levels of the tree, and every leaf taxon is x branches away from the root (Figure A.3). Each branch is assigned a length using the same procedure as for treezilla trees. The root node is implicit, so the branch that contains the implicit root is on average twice as long as the other branches. For purposes of tree generation, that branch's length is assigned as though it is two separate branches.

A.4.1 Correctness

We generated a total of 10 random trees for each of the following numbers of taxa: 3 to 50, and $100x$, where $x = [1..100]$. Branch lengths were gamma-distributed: $\sim \Gamma(\alpha = 2, \lambda = 1/2)$. For each of these trees we generated the corresponding additive distance matrix, then used RNJ to reconstruct a tree. In every case, RNJ succeeded in recovering the original tree.

A.4.2 Speed

We generated trees of the two extreme shapes: pectinate and perfect. Branch lengths were gamma-distributed: $\sim \Gamma(\alpha = 2, \lambda = 1/2)$. The trees had $2x$ taxa, where $x = [9, 14]$. From these trees, we generated additive distance matrices, which were randomly shuffled in order to avoid inputs that favored a particular search order. We then compared the runtime of RNJ to the runtimes of PHYLIP neighbor [30], QuickTree [24], and QuickJoin [31]. The experiments were run on an Intel Pentium-4 3 GHz Linux system, and all four programs were compiled with the same optimization flags. Figure A.4 summarizes the results.

A.4.3 Quality

We used ROSE [44] to simulate true alignments and true trees for 512 taxa under the F84

model of molecular evolution [60, 43] for four tree shapes (pectinate, treezilla, random, and perfect), ranges of sequence length (250 to 2500, in increments of 250), divergence time (0.25 to 2.5, in increments of 0.25), and level of evolutionary rate heterogeneity ($\alpha = 2^x$ for x from 1 to 10). Rose was run with a mean mutation rate of 0.01342302. Mutation rate is meaningful only in the context of time; we chose the mutation rate and time intervals such that the full range of useful divergence was simulated. We used insertion/deletion thresholds of 5.0×10^{-6} , and insertion/deletion function vectors of [2,3,4,4,3,2,1]. We chose these insertion/deletion settings in order to add a level of biological realism to the problem of calculating pairwise sequence distances. We used PHYLIP's dnadist program to estimate pairwise sequence distances according to the F84

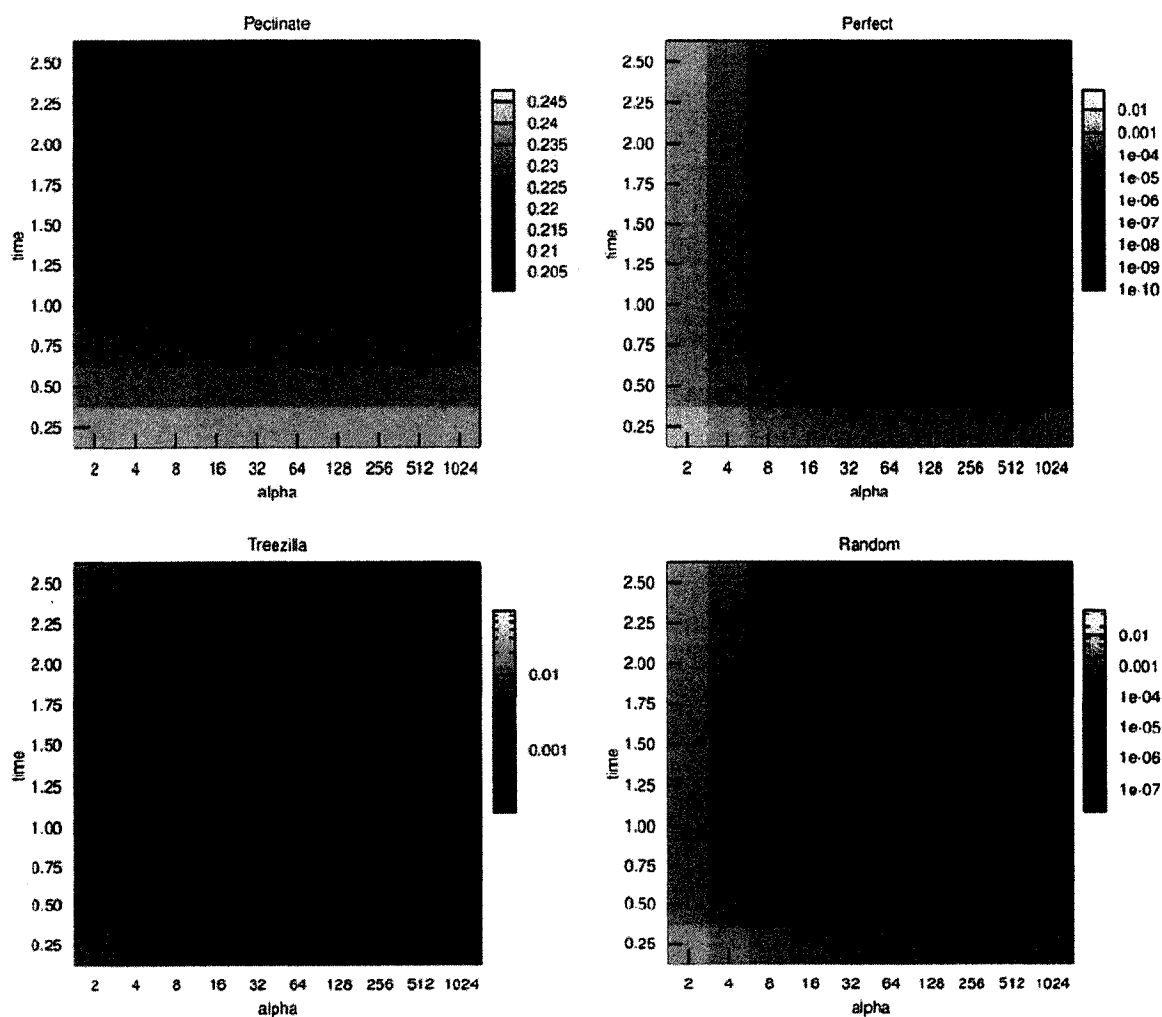


Figure A.5: RNJ quality results (mean MSE of Robinson-Foulds distances) for all four tree shapes (pectinate, treezilla, random, and perfect), where sequence length is 1500. The results are worst for pectinate trees and best for perfect trees.

model. We used QuickTree to generate NJ trees and used the Robinson-Foulds distance measure [35], specifically as described by Moret *et al.* [61], to calculate the distance of each resulting tree from the corresponding true tree. The Robinson-Foulds distance measure represents the proportion of branches in two trees that induce bipartitions unique to one tree or the other.

Experiments were replicated on two levels.

1. One hundred RNJ and 100 NJ trees were created from each distance matrix, and the mean squared error (MSE) was calculated for the resulting Robinson-Foulds distances. The distance matrix was randomly shuffled before each NJ replicate, so that ties would be broken approximately randomly.
2. At a higher level, each experimental configuration was replicated 100 times and the mean and variance of the value mentioned in (1) were calculated. These two levels of replication were necessary in order to avoid spurious results due to stochasticity of RNJ, NJ, or the simulations. A total of 80 million trees were generated and analyzed in these tests. All of the quality experiments were run on a 220-processor Beowulf supercomputer.

Due to the large volume of the raw results, we were only able to fit representative samples here, along with descriptions of the general trends. In general, the quality of the RNJ and NJ results was very similar. Both algorithms performed best on perfect trees, nearly as well on random trees, somewhat worse on treezilla trees, and very poorly on pectinate trees. Figure A.5 shows one slice of the results for RNJ. We do not present the variance of the mean squared error because for all experiments, a lower mean equated to a lower variance. The general patterns are very similar for RNJ and NJ; the only differences are variations in magnitude. As sequence length increased, results universally improved.

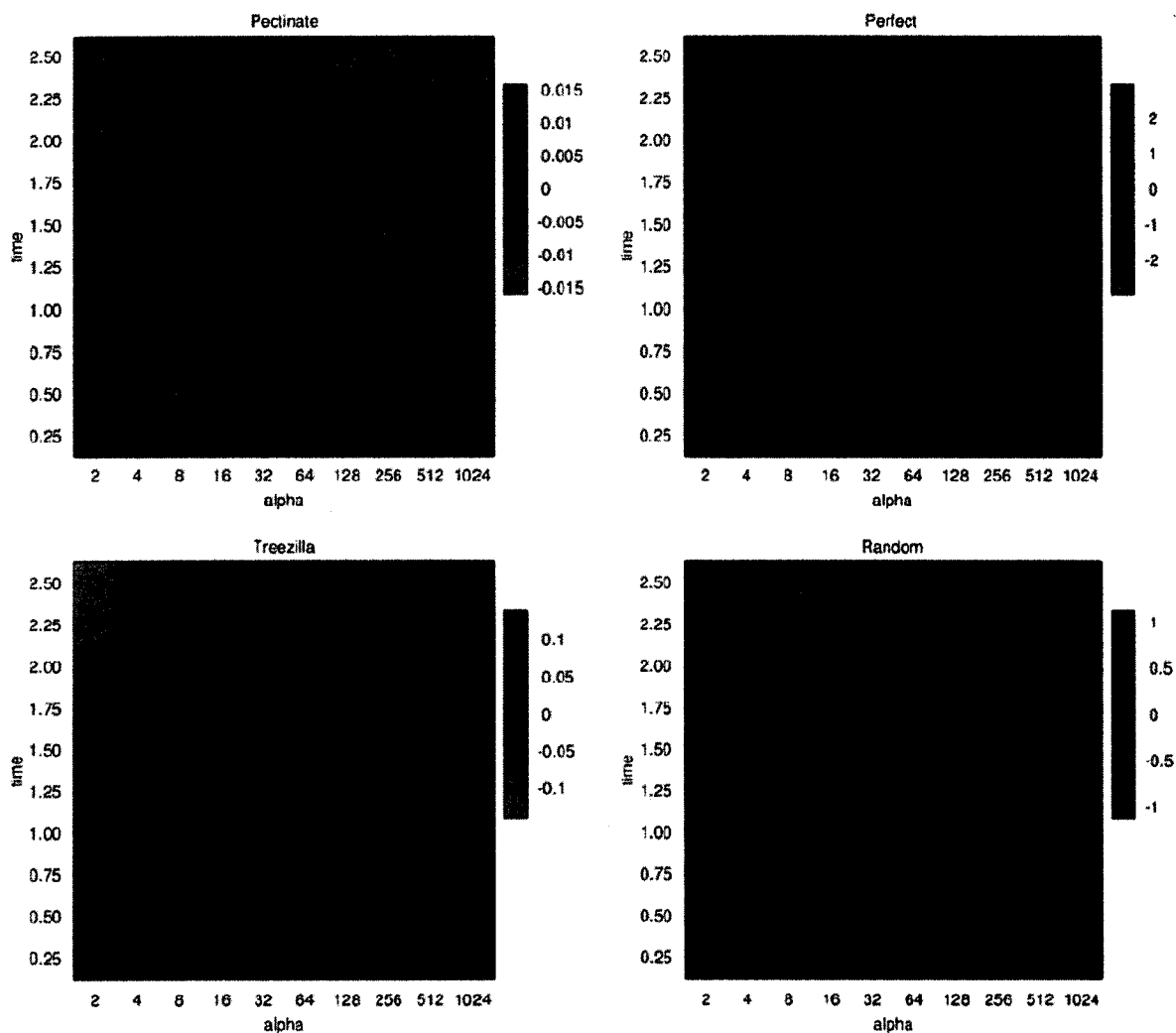


Figure A.6: Quality results of RNJ versus NJ, for all four tree shapes (pectinate, treezilla, random, and perfect), where sequence length is 1500. The plots depict relative quality using the formula $\log_{10} \text{MMSENJ} - \log_{10} \text{MMSE}_{\text{RNJ}}$, where MMSE is the mean MSE of the Robinson-Foulds distances to the true trees. The formula calculates orders-of-magnitude differences between RNJ and NJ. Positive values mean that RNJ performed better than NJ. For the dark cells in the perfect trees plot, NJ always recovered the true tree topology.

Results were best for $1:00 \leq \text{time} \leq 1:75$, depending on the tree shape. This was expected, because we chose the range of time such that at the extremes, PHYLIP dnadist was barely able to compute pairwise distances. As α increased (molecular clock rate heterogeneity decreased), results generally improved. For pectinate trees however, results were uniformly bad, regardless of α ; RNJ and NJ generated poor trees because the variations in the long branches overwhelmed the short branches along the “spines” of the trees. This is an inherent aspect of pectinate trees, which calls into question the general usefulness of RNJ and NJ on pectinate

trees. Nonetheless, we included pectinate trees because they represent one of the two extreme cases of tree topology, and because some previous NJ quality experiments included them [54]. Figure A.6 shows the relative performance of RNJ and NJ for one slice of the quality experiments. With few exceptions, NJ generated slightly better trees than RNJ did for all configurations of the experiments that were based on perfect, random, and pectinate trees. For the treezilla-based experiments, however, NJ only did slightly better for the shortest sequences, and the RNJ trees quickly surpassed those of NJ as the sequence length increased. It is important to note that even in this case, the algorithms produced trees of very similar quality.

A.5 Discussion

NJ favors joins for which there is maximal agreement about whether the nodes under consideration are neighbors, as evidenced by transformed distances. RNJ treats all plausible joins as equally good. As such, RNJ trees tend to vary more than NJ trees. NJ is maximally greedy at each join, whereas RNJ is less greedy. NJ's greediness can cause systematic bias, which leaves open the risk of uniformly poor results for certain classes of input. Although RNJ is also potentially prone to bias, that bias is of a less troubling nature; all join operations for which the distance matrix contains support are given approximately equal opportunity, whereas NJ may completely exclude potential joins for which support is low. Were it possible to accurately quantify evidence for potential joins, an unbiased algorithm would randomly choose from the possibilities proportional to their levels of support.

The quality experiments were constructed such that the algorithm that performs better tends to have lower variance, because outliers have a large effect on the summary statistics. RNJ can construct a superset of the trees that NJ can construct, and this higher variance could be an advantage in some cases. Consider that the simulations used the same model of molecular evolution for both simulation and pairwise distance estimation. There was no mismatch between the true model and the inference model, so lower variance generally meant better overall results, but if there were a model mismatch, as is the case for biological data, RNJ's higher variance would improve the chances of capturing the true tree in its distribution of possible resulting trees. Thus RNJ is more robust than NJ in such a case.

We demonstrated that RNJ is substantially faster than NJ, which makes RNJ compelling for certain uses. For example, heuristic searches for optimal trees often start with NJ trees and try to improve from there. For large datasets, the substantial time that RNJ saves compared to NJ

can instead be used for the heuristic search, which should allow more starting points to be considered in the same amount of time. Another application is that of guide tree creation for progressive multiple sequence alignment (MSA), as implemented by programs such as CLUSTAL W [2]. NJ is the most algorithmically complex step of progressive MSA, so for large numbers of sequences, using RNJ instead of NJ can have a substantial impact on total program runtime.

As for overall quality of results, neither algorithm is clearly superior. NJ clearly produces better trees on average for perfect trees, but RNJ produces better trees on average for treezilla-based trees. That RNJ does better than NJ for the experiments that were based on biological data leaves us to wonder if this is peculiar to the treezilla data or if RNJ will generate better trees than NJ for a wide range of biologically based data.

Bibliography

- [1] Shyu, C., L. Sheneman, J.A. Foster (2004) Multiple Sequence Alignment with Evolutionary Computation, *Genetic Programming and Evolvable Machines special issue on Biological Applications of Evolutionary Computation*, (5)2:121-144.
- [2] Thomson, J.D., Higgins D.H., 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.
- [3] Needleman, S.B. and Wunsch, C.D. (1970) A general method application to the search for similarities in the amino acid sequences of two proteins, *Journal of Molecular Biology*, 48(3):443-53.
- [4] Smith, T.F., Waterman, M.S. (1981) Identification of Common Molecular Subsequences, *Journal of Molecular Biology*, 48:443-453.
- [5] Saitou, N., and Nei, M. (1987) The Neighbor-Joining method: a new method for reconstructing phylogenetic trees", *Journal of Molecular Biology Evolution*, 4:406-425.
- [6] Kececioğlu, J.D., and Zhang, W. (1998) Aligning Alignments, In *Proceedings of the 9th Annual Symposium on Combinatorial Pattern Matching (CPM '98)*.
- [7] Jukes, T.H., and Cantor, C. (1969) Evolution of protein molecules, In H.N. Munro (Ed), *Mammalian Protein Metabolism*, Volume III, Chapter 24, pp. 21-132, Academic Press, New York.
- [8] Thompson, J.D., Plewniak, F., and Poch O. (1999) BALiBASE: a benchmark alignment database for the evaluation of multiple alignment programs, *Bioinformatics*, 15(1):87-88.
- [9] Sheneman, L., J.A. Foster (2004) Evolving Better Multiple Sequence Alignments, In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004)*, Seattle, WA.
- [10] Just, W. (2001) Computational Complexity of Multiple Sequence Alignment with SP-Score. *Journal of Computational Biology*. 8(6):615-623.
- [11] Notredame, C. (2002) Recent progresses in multiple sequence alignment: a survey. *Pharmacogenomics*, 2002. 3(1).
- [12] Notredame, C., Higgins D.G. (1996) SAGA: sequence alignment by genetic algorithm. *Nucleic Acids Research*. 24(8):1515-1524.
- [13] Carillo, H., Lipman, D. (1988) The multiple sequence alignment problem in biology. *SIAM Journal on Applied Mathematics*. 48(5):1073-1082.

- [14] Thomsen, R., Fogel, G.B., Krink, T. (2002) A CLUSTAL Alignment Improver using Evolutionary Algorithms, In *Proceedings of the Fourth Congress on Evolutionary Computation (CEC-2002)*, vol. 1, pp 121-126.
- [15] Higgins, D.G., Bleasby A.J., Fuchs, R. (1992) CLUSTAL V: improved software for multiple sequence alignment. *Comput Appl Biosci.* 8(2):189-191.
- [16] Lewis, P.O. (1998) A genetic algorithm for maximum-likelihood phylogeny inference using nucleotide sequence data. *Molecular Biology Evolution.* 15(3):277-283.
- [17] Matsuda, H. (1996) Protein phylogenetic inference using maximum likelihood with a genetic algorithm. In *Proceedings of the Pacific Symposium on Biocomputing (PSB '06)*. World Scientific, London.
- [18] Congdon, C.B. (2002) Gaphyl: An Evolutionary Algorithms Approach for the Study of Natural Evolution. In *Proceedings of the Genetic Evolutionary Computation Conference (GECCO '02)*. Morgan Kaufmann.
- [19] Fitch, W.M. (1971) Toward defining the course of evolution: Minimum change for a specific tree topology. *Systematic Zoology.* 20:406-416.
- [20] Koza, J. (1992) *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press.
- [21] Lewis, P.O. (1998) A Genetic Algorithm for Maximum Likelihood Phylogeny Inference Using Nucleotide Sequence Data, *Mol. Biol. Evol.* 15(3):277-283.
- [22] Dayhoff, M.O., Schwartz, R.M., and Orcutt, B.C. (1978) A model of evolutionary change in proteins., in M. O. Dayhoff, editor, *Atlas of Protein Sequence and Structure*. Vol.5, Suppl. 3, chapter 22, pages 345–352. National Biomedical Research Foundation.
- [23] Henikoff, S., Henikoff, J.G. (1992) Amino acid substitution matrices from protein blocks. in *Proceedings of the National Academy of Sciences of the USA*, 1992. 89(22):10915–10919.
- [24] Howe, K., Bateman, A., Durbin, R. (2002) QuickTree: building huge Neighbor-Joining trees of protein sequences. *Bioinformatics.* 18(11):1546-1547.
- [25] Felsenstein, J. (1981) Evolutionary trees from DNA sequences: A maximum likelihood approach. *Journal of Molecular Evolution*, 17:368-376.
- [26] Sheneman, L., Evans, J. and Foster, J.A. (2006) Clearcut: a fast implementation of Relaxed Neighbor-Joining. *Bioinformatics.* 22(22):2823-4.
- [27] Evans, J., Sheneman, L. and Foster, J.A. (2006) Relaxed Neighbor-Joining: A Fast Distance-Based Phylogenetic Tree Construction Method, *J. Mol. Evol.* 62(6):785-92.
- [28] Waterman MS, Smith TF, Singh M, Beyer WA (1977) Additive evolutionary trees. *J Theor Biol.* 64:199–213.
- [29] Studier, J. A. and Keppler, K.J. (1988) A note on the Neighbor-Joining algorithm of Saitou and Nei. *Mol. Biol. Evol.* 5:729–731.

- [30] Felsenstein, J. (2004) PHYLIP (phylogeny inference package) version 3.6. Distributed by the author, Department of Genome Sciences, University of Washington, Seattle.
- [31] Mailund, T. and Pedersen, C.N.S. (2004) QuickJoin – fast neighbour-joining tree reconstruction. *Bioinformatics*, 20:3261–3262.
- [32] Sheneman, L., Foster, J.A., (2006) Estimating the Destructiveness of Crossover on Binary Tree Representations, In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '06)*, Seattle WA., July 8th - July 13th, 2006.
- [33] R. Poli and W.B. Langdon (1998) Schema theory for genetic programming with one-point crossover and point mutation. *Evolutionary Computation*, 6(3):232-252.
- [34] O'Reilly, U.M., Oppacher, F. (1995) The troubling aspects of a building block hypothesis for genetic programming. In L. Darrell Whitley and Michael D. Vose, editors, *Foundations of Genetic Algorithms 3*, pages 73-88, Estes Park, Colorado, USA, 31 July - 2 August 1994. Morgan Kaufmann.
- [35] Robinson, D.R., and Foulds, L.R. (1981) Comparison of phylogenetic trees. *Mathematical Biosciences*. 53: 131-147.
- [36] Felsenstein, J. (2004) *Inferring Phylogenies*, Sinauer Associates, Sunderland, Massachusetts.
- [37] Bellman, R. (1957) *Dynamic Programming*. Princeton, NJ: Princeton Univ. Press.
- [38] Gotoh, O. (1982) An improved algorithm for matching biological sequences. *J. Mol. Biol.* 162:705-708.
- [39] Henikoff, S. and J. Henikoff (1993) Performance evaluation of amino acid substitution matrices. *Proteins: Structure, Function, and Genetics*, 17:49–61.
- [40] Gotoh, O. (1990) Optimal sequence alignment allowing for long gaps. *Bull. Math. Biol.*, 52:359-373.
- [41] Wang, L. and T. Jiang (1994) On the complexity of multiple sequence alignment. *J. Comput. Bio.*, 1:337-348
- [42] Feng, D.F. and R.G. Doolittle, (1987) Progressive sequence alignment as a prerequisite to correct phylogenetic trees. *J. Mol. Evol.*, 25:351–360.
- [43] Felsenstein, J., and G. A. Churchill. (1996) A hidden Markov model approach to variation among sites in rate of evolution. *Mol. Biol. Evol.* 13:93–104.
- [44] Stoye, J., D. Evers, F. Meyer (1998) ROSE: generating sequence families, *Bioinformatics*, 14(2):157-163.
- [45] Notredame C, Higgins DG, Heringa J. (2000) T-Coffee: A novel method for fast and accurate multiple sequence alignment., *J Mol Biol.* 302(1):205-17.
- [46] Edgar, Robert C. (2004) MUSCLE: multiple sequence alignment with high accuracy and high throughput, *Nucleic Acids Research* 32(5), 1792-97.

- [47] Sneath & Sokal (1973) *Numerical Taxonomy*. W.H. Freeman and Company, San Francisco, pp 230-234.
- [48] Kimura M (1980) A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *J. Mol. Evol.* 16:111-120.
- [49] Nelesen, K. LIU, C.R. Linder, and W. Warnow (2008) The effect of the guide tree on multiple sequence alignment and subsequent phylogenetic analyses, In *Proceedings of the Pacific Symposium on Biocomputing (PSB '08)*, 13:25-36.
- [50] Sanderson, M. J., M. J. Donoghue, W. Piel, and T. Eriksson (1994) TreeBASE: a prototype database of phylogenetic analyses and an interactive tool for browsing the phylogeny of life. *Amer. Jour. Bot.* 81(6):183.
- [51] Takamatsu, S., U. Braun, and S. Limkaisang (2005) Phylogenetic relationships and generic affinity of *Uncinula septata* inferred from nuclear rDNA sequences, *Mycoscience*, 46:9-16.
- [52] Jiao, Z. and J. Li. (2006) Phylogeny and biogeography of intercontinental disjunct Gelsemiaceae inferred from chloroplast and nuclear DNA sequences. *Systematic Botany*, 32(3):617-627.
- [53] Cardinale, F., L. Ferraris, D. Valentino, and G. Tamietti (2007) Induction of systemic resistance by a hypovirulent *Rhizoctonia solani* isolate in tomato. *Physiological and Molecular Plant Pathology*, 69(4-6):160-171.
- [54] Katoh, M., Kuma, M. (2002) MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform, *Nucleic Acids Res.* 30:3059-3066.
- [55] Saitou N, Imanishi T (1989) Relative efficiencies of the Fitch- Margoliash, maximum-parsimony, maximum-likelihood, minimum- evolution, and Neighbor-Joining methods of phylogenetic tree construction in obtaining the correct tree. *Mol. Biol. Evol.* 6:514-525.
- [56] Kuhner, M.K., Felsenstein, J. (1994) A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates. *Mol Biol Evol.* 11:459-468.
- [57] Bruno, W.J., Socci, N.D., Halpern, A.L. (2000) Weighted Neighbor-Joining: a likelihood-based approach to distance-based phylogeny reconstruction. *Mol. Biol. Evol.* 17:189-197.
- [58] Gascuel, O. (1997) BIONJ: an improved version of the NJ algorithm based on a simple model of sequence data. *Mol. Biol. Evol.*, 14:685-695.
- [59] Chase MW, Soltis DE, Olmstead RG, Morgan D, Les DH, Mishler BD, Duvall MR, Price RA, Hills HG, Qiu YL, Kron KA, Rettig JH, Conti E, Palmer JD, Manhart JR, Sytsma KJ, Michael HJ, Kress WJ, Karol KG, Clark WD, He´dren MH, Gaut BS, Jansen RK, Kim KJ, Wimpee CF, Smith JF, Furnier GR, Strauss SH, Xiang QY, Plunkett GM, Soltis PS, Swensen SM, Williams SE, Gadek PA, Quinn CJ, Equiarte LE, Dolenberg E, Learn GH Jr, Graham SW, Barrett SCH,

- Dayandan S, Albert VA (1993) Phylogenetics of seed plants: An analysis of nucleotide sequences from the plastid gene *rbcl*, *Ann. Mo. Bot.*, 80:528–580.
- [60] Kishino, H., Hasegawa, M. (1989) Evaluation of the maximum likelihood estimate of the evolutionary tree topologies from DNA sequence data, and the branching order in Hominoidea. *J. Mol. Evol.*, 19:170–179.
- [61] Moret BME, Nakhleh L, Warnow T, Linder CR, Tholse A, Padolina A, Sun J, Timme R (2004) Phylogenetic networks: modeling, reconstructibility, and accuracy. *IEEE/ACM Trans Comput Biol Bioinform*, 1:13–23.