

**Phylogenetic Pipeline for the Detection of
Horizontal Gene Transfer**

by

Julia Goodrich

Acknowledgements

I would first like to thank my advisor Rob Knight. Over the past couple of years he was a wonderful mentor and has provided me with an amazing opportunity and invaluable guidance. Rob has supported me through my time in his lab, and I would not be where I am today without him. My time in his lab has changed my entire plans, I had never considered graduate school until experienced the endless possibilities of research by being in his lab. I will always remember what he has done for me.

I would also like to thank the entire Knight lab for their support and assistance during my time there. Everyone in the lab was extremely helpful and made my experience in the lab more incredible than I could have ever imagined. Specifically, I want to thank Daniel McDonald and Jesse Zaneveld. Daniel McDonald, aside from being a good friend, has helped me with many of the research projects I have worked on. He worked with me on the phylogenetic reconstruction project in chapter two and decorating a tree with a seven level taxonomy in chapter three. Jesse Zaneveld was always available for my questions about horizontal gene transfer, without him I would not have gained the knowledge I now have about horizontal gene transfer.

Next, I would like to thank the rest of my thesis committee, Debra Goldberg and Henry Tufo, for their time and assistance during my thesis research. Debra

Goldberg in addition to being on my committee gave me guidance on the development of a new horizontal gene transfer detection method described in chapter seven. The work done in chapter two was in collaboration with Henry Tufo, so I would like to thank him for support during that project.

Another person I want to thank is Ryan Walters for his help in analyzing the data I generated in chapter eight. Others that deserve thanks are the members of my group in my User Interface Design class; Anuraag Chintalapally, Matthew Ripley, and Corey Teffetator. All three were very supportive of me while I was doing my thesis, and helped to develop the horizontal gene transfer visualization software briefly mentioned in chapter six.

I would also like to thank my parents, Jim and Karen Goodrich, for all of their support during this process. From listening to my presentations and reading parts of my thesis to calming me down when I was stressed or frustrated, they were always there for me. My fiancé and sister were also support systems for me, encouraging me to keep writing and putting up with many sleepless nights.

There were also grants and scholarships that helped to fund my research. They are the Undergraduate Research Opportunities Program (UROP), HHMI, the Goldwater Scholarship, NASA ROSES Astrobiology award #EXOB07-0094 and NIH R01HL90480.

Contents

Chapter

1	Introduction	1
1.1	Basic Biology Overview	1
1.2	Phylogenetics	2
1.3	Horizontal Gene Transfer	3
2	Benchmarking of Different Methods of Building Phylogenetic Trees	7
2.1	Motivation	7
2.2	Phylogentic Reconstruction Methods	8
2.3	Simulations	8
2.4	Data Collection and Organization	11
2.4.1	Tree Inference	11
2.4.2	Post Processing	11
2.4.3	Data Management	12
2.5	Visualization and Interpretation of Results	13
3	Building a Seven Level Taxonomy	21
3.1	Motivation	21
3.2	NCBI Taxonomy	22
3.3	Designed Objects	22
3.4	Assignment of Ranks and Common Names	24

3.5	Cleanup of Missing Ranks	26
3.6	Cleanup of Missing Names	28
3.7	Choose from Duplicate Ranks	29
3.8	String Representation of Tree	29
4	Review of Phylogenetic Algorithms for Detecting Horizontal Gene Transfer	31
4.1	Subtree Prune and Regraft	31
4.2	Maximum Agreement Subtree	32
4.3	Maximum Likelihood and Maximum Parsimony	33
4.4	Available Methods	34
5	Double Birth-Death Model	35
5.1	Importance	35
5.2	Algorithm Implementation	36
6	Infrastructure for Running High Throughput Tests	39
6.1	Gaining Knowledge of the System	39
6.2	Simulating Appropriate Datasets	40
6.3	Running Detection Programs on Datasets	41
6.4	Data Management and Analysis	42
6.5	HGT Visualization	42
7	A New Network Based Method for Detecting Horizontal Gene Transfer	44
7.1	Data Collection	44
7.2	Clustering	45
7.3	Building the Network	46
7.4	Interesting Results	46
7.5	Algorithm to Automate HGT Detection	48

8	Compositional Methods for Detecting Horizontal Gene Transfer: Stand Alone Version of CodonExplorer	54
8.1	Codon Explorer	54
8.2	Rationale	55
8.3	Challenges	57
8.4	Results	58
9	Conclusions and Future Directions	65
	Bibliography	68

Tables

Table

2.1	Phylogenetic reconstruction parameters	10
6.1	Queues on Steele cluster	40
7.1	Simulated HGT events	48

Figures

Figure

2.1	Three backbone topologies used for simulations	9
2.2	Tip-to-tip distances and monophyletic recovery ratios for reconstruction methods	14
2.3	Tip-to-tip distances by sequence length	15
2.4	Tip-to-tip distances and by the number of tips	16
2.5	Tip-to-tip distances for rooted and un-rooted trees	17
2.6	Tip-to-tip distances for different rate matrices	18
2.7	Tip-to-tip distances of ClearCut and FastTree by method parameters	20
3.1	Seven levels of Taxonomy	23
3.2	Demonstration of maximum tip-to-tip distance calculation	25
3.3	Addition of missing ranks using dummy nodes	27
3.4	Solution to potential problem when adding missing ranks	28
4.1	Illustration of a single SPR move	32
4.2	Example of finding the maximum agreement subtree	33
5.1	Example of double birth-death algorithm	38
6.1	Screenshot of HGT visualization tool	43
7.1	Network produced from clustering gene sequences	47

7.2	Species tree from double birth-death model	49
7.3	Illustration of finding last common ancestor of cluster	50
7.4	Illustration using tip-to-tip distance to distinguish HGT event from related species	51
7.5	Illustration of finding the direction of an HGT event	52
7.6	Output from new HGT detection method	53
8.1	CodonExplorer analysis of <i>E. Faecalis</i> full genome	60
8.2	CodonExplorer tRNAs and ribosomal RNAs outliers from <i>E. Fae-</i> <i>calis</i> full genome	61
8.3	CodonExplorer: <i>E. Faecalis</i> genes that are involved in the glycoly- sis pathway	62
8.4	CodonExplorer: <i>E. Faecalis</i> gene belonging to the merR family . .	63

Chapter 1

Introduction

Completing large-scale phylogenetic analyses is becoming increasingly important as the number of biological sequences grows. In particular, datasets are larger than ever before because of high-throughput sequencing technologies, such as pyrosequencing [31], making analyses correspondingly more computationally intensive. The evolutionary history of horizontal gene transfer (HGT) has important implications for human and ecosystem health. Finding the most efficient and accurate workflow for detecting HGT events is crucial to understanding the role HGT has played in evolution. HGT detection using phylogenetic methods is extremely sensitive to the inference of the phylogenetic trees used for detection. The tree inference itself is dependent on the parameters used and the dataset being analyzed. To improve HGT detection we need to optimize the results of reconstruction methods. Computational requirements for phylogenetic reconstruction scale poorly as larger datasets are used.

1.1 Basic Biology Overview

The central dogma of biology is that information flows from DNA to RNA to protein. HGT events occur at the DNA level. DNA, deoxyribonucleic acid, is

a nucleic acid molecule containing the genetic information in all of our cells [50]. This genetic information provides instructions that determine our specific traits that make us different from one another. The building blocks of DNA are the four nucleotides adenine (A), cytosine (C), guanine (G) and thymine (T). In 1953 Watson and Crick proposed that DNA was structured as a double helix, which looks similar to a spiral staircase. Each side of the double helix contains the same set of repeating nucleotides, but they are in opposite directions from one another, or anti-parallel. This occurs because of base pairing, A pairs with T and G pairs with C. So if there is an A on one side of the double stranded DNA it pairs with a T on the other strand.

Ribonucleic acid (RNA) is made from DNA by a process called transcription [50]. RNA is also a carrier of genetic material. RNA is built from three of the same nucleotides as DNA, but thymine is replaced by uracil (U). Most RNA is then translated into proteins, but some remains as RNA. These proteins and functional RNAs perform the tasks specified by our genes. A gene is a section of DNA that encodes for a specific RNA or protein needed by the organism. As species evolve, genetic material changes to code for this evolution. We refer to the change of a nucleotide in DNA as a substitution or mutation.

1.2 Phylogenetics

Phylogenetics is the study of the relationships between organisms based on their evolutionary history; phylogenetic reconstruction is a subset of phylogenetics. Phylogenetic reconstruction methods are used to try and determine the historical relationships among organisms given their biological sequences. The species are

leaves of the tree while all the connecting branches show how those species are related. An ongoing effort in biology is to reconstruct the “Tree of Life,” which is an evolutionary tree relating all species on earth to one another. This obviously requires methods that can reconstruct accurate phylogenies for a large number of taxa in a reasonable amount of time. The two general categories of reconstruction methods are distance-based and character-based [10].

Distance-based methods such as Neighbor-Joining [41] use a matrix of distances between pairs of taxa in a sequence alignment to build a tree. NJ methods are rapid, but since it is usually a fast method since it only examines the distances some of the information that is important in determining evolution is lost. Maximum likelihood [14] and maximum parsimony [11] are both character-based methods, these methods generate all possible topologies and statistically determine which topology is the best given each column of the sequence alignment. Character-based methods are typically more accurate, but they are very slow since all possible trees need to be considered.

1.3 Horizontal Gene Transfer

Horizontal gene transfer (HGT) is the movement of genetic material from one strain or species to another. Bacteria are asexual, and it was thus once believed that genes in bacteria were simply transferred vertically with each lineage, but we now know they can also be transferred horizontally. In 1958 Joshua Lederberg was awarded the Nobel Prize for discovering conjugation, a mechanism of bacterial gene exchange that is a major cause of HGT. HGT is most common in bacteria and archaea, but studies have shown that it also occurs in some eukaryotes [19].

Over the years many researchers have attempted to determine the extent of HGT throughout the “Tree of Life”, but these studies have yielded conflicting results, leaving unanswered the question of how pervasive HGT is in biology. Although the extent of HGT is not known, comparisons that have been done on genomes of closely related species suggest an alternative to the “Tree of Life”. For example the completely sequenced genomes of three strains of *E. coli* were compared and shown to have only 39.2% of their combined sets of genes in common [51]. Ge *et al.* [15] suggests that there is still the underlying “Tree of Life” representing the vertical transfers, but it contains small “cobwebs” that represent all of the horizontal gene transfers without changing the underlying tree.

HGT has profound implications on ecology, evolution, and medicine [36]. It has played an important role in bacterial evolution and diversity by allowing species to acquire the necessary genes to adapt to new environments. HGT also has important medical implications because it provides a means for bacteria to acquire resistance to antibiotics. Since HGT has had a great impact on the world, further study of HGT and its consequences is essential.

Developing new computational techniques to detect genes that have been transferred among species will guide future biological experiments to understand the diverse effects of HGT on biology. Having the ability to detect HGT events accurately will improve our ability to infer the evolutionary history of species, which is a major ongoing effort in biology. This research could have an ecological impact by providing insight into new ways to adapt bacteria to new environments in which they otherwise could not thrive, but where their presence would cause improvement. Alternatively, if certain bacteria has invaded an environment and caused problems, identifying and understanding HGT could help us find ways to

rid the bacteria from the environment. HGT detection could also lead to new testable predictions about the functions of genes in multiple species. Moreover, new understandings of HGT that follow from research in this area could lead to the identification of new drug targets, thereby advancing medical research. This in turn will help with fighting infections that otherwise might not be cured due to the vast problems resulting from bacterial resistance to current antibiotics. It is apparent that there is significant motivation for the development of new methods for HGT detection.

Detection of HGT events has been studied extensively, but no method can accurately detect all transfers that have occurred [39, 40]. HGT detection methods can be split into two different categories: compositional and phylogenetic. Compositional methods are those that examine the genomic sequence to look for atypical regions. For example, assessing GC content to find genes that exhibit a difference from the bulk of the genome [26]. Phylogenetic methods seek to find disagreements within the topology of a phylogenetic tree by comparing several gene trees with a species tree or with other gene trees. In doing so, these methods attempt to find the HGT events that might have caused this disagreement to occur. When comparing a gene tree and species tree, if the gene tree has a different topology than the species tree then it could mean that the gene was transferred from one species to another at some point in time. However, inferring gene trees correctly can be very challenging because fast-evolving genes have a poor signal deep in the tree. Hence, there may be stochastic variation in gene trees due to noise around this poor signal that can be mistaken for HGT. The species trees are also imperfectly known, they are usually determined with inference methods that use a sequence alignment to infer the tree; however, no inference method is exact.

The overall aim of this thesis is to provide additional insight into the areas of computational biology involving phylogenetics and HGT. More specifically, benchmarking phylogenetic reconstruction methods, developing new tools for detecting HGT, using existing methods of HGT detection, and putting together a framework to help users of phylogenetic detection methods. The necessary first step was to obtain a detailed characterization of the commonly used phylogenetic reconstruction programs. This was important to give us an understanding of how well tree inference methods perform, and which perform better overall. This knowledge is important in deciding the appropriate reconstruction method to use when building the trees for phylogenetic HGT detection.

Chapter 2

Benchmarking of Different Methods of Building Phylogenetic Trees

2.1 Motivation

The goal for this project was to discover what if any conditions yield high quality phylogenies despite the limitations of the current inference methods. Specifically, we were interested in how the sequence length, the representation of individual families, and total number of sequences affect the reconstruction methods. To answer this question, we generated over 15,000 phylogenetic trees and simulated about 200,000 sequence alignments from these trees. Then the alignments were used to infer trees with three widely used inference methods producing around 2,100,000 results. The resulting trees were compared to the original trees by their tip-to-tip distance as well as the ratio of the number of correctly recovered monophyletic groups. This took approximately 2 months using 750,000 CPU hours on Frost, a 8,192 processor BlueGene/L system at the National Center for Atmospheric Research (NCAR). The project was done in collaboration with Norm Pace's lab in the Molecular, Cellular, and Developmental Biology Department and Henry Tufo's lab in the Computer Science Department. I specifically put together the code to run the simulations and generate the graphs. I helped with code for the metrics of comparison and the design of the database.

2.2 Phylogentic Reconstruction Methods

The three applications used in this analysis were FastTree, ClearCut, and RAxML [38, 45, 47]. FastTree and Clearcut use modified versions of the Neighbor-Joining algorithm. Both of these programs have relatively light computational requirements. Clearcut has a memory footprint of $O(N^2)$ while FastTree is only $O(NLa)$, where N is the number of sequences, L is the width of the alignment and a is the size of the alphabet. RAxML is a maximum likelihood-based method that is computationally heavy and also has large memory requirements, but it has been known to produce high quality results.

2.3 Simulations

Determining the optimal conditions for reconstructing phylogenies required the use of simulated sequences over a controlled set of parameters defining their generation. To do so required the construction of biological-like trees in which sequences could be derived from. These sequences were then used for phylogenetic reconstruction, the result of which could be directly compared to the original tree from which the sequences were based on. Five datasets varying several parameters were simulated (see Table 2.1). To begin we used sequences from the small subunit rRNA alignment in the Greengenes database with about 125,000 sequences from the Guerrero Negro(GN) microbial mat [27] added to the alignment. We also used a phylogenetic tree from the ARB [29] database that is based on parsimony

insertion of the GN sequences to the baseline tree that was released by Greengenes [9]. The subtrees for all of the experimentally determined divisions with over 1,000 sequences were extracted from the tree. Only the species with over 1,250 nucleotides in the alignment were used for the study. The topologies of subtrees, 20 total, were then “grafted” onto varying backbone topologies providing many simulated phylogenetic trees. The backbone tree topologies were star, comb, and balanced as shown in Figure 2.1.

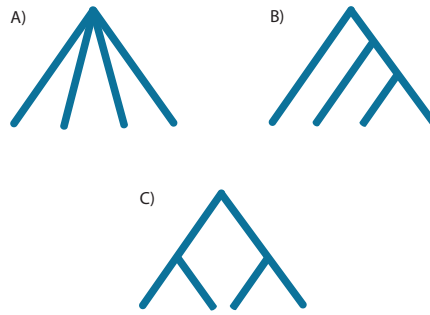


Figure 2.1: We “grafted” experimentally determined divisions on to. A) star, B) comb, C) balanced.

We then varied the number of divisions “grafted” onto the backbone topology using sampling with replacement. The number of species on the tree was another parameter that we chose to look at; the species were sampled using both random and balanced sampling. For random sampling, each of the divisions was “grafted” onto the backbone once and n number of species was randomly chosen from all the species on the tree and a subtree was formed containing only those species. Balanced sampling was performed by randomly choosing the n/k sequences from each of the divisions where n is the total number of species and k is the number of divisions on the tree. The final parameter varied for the construction of simulated trees was the internal branch length on the backbone topology.

Seven different internal branch lengths were used: 0.01, 0.02, 0.03, 0.04, 0.05, 0.075, 0.10.

After generating over 15,000 phylogenetic trees we used the PyCogent [25] SeqSim module that utilizes an evolutionary model to simulate biological-like sequences based on the phylogeny of each tree. The sequences were simulated with different lengths, as the quality of inferred trees likely depends on the length of the sequences; longer sequences means more data explaining the evolutionary history. Additionally, we used different evolutionary rate-matrix strategies for simulating the sequences. The strategies were: (i) a single empirically determined matrix, propagated through the tree; (ii) a single empirically determined matrix for each division and propagated through the division; (iii) a matrix at each tip that is empirically determined from the division that tip is in, propagated back through the tree; (iv) a randomly generated matrix at each node. Some of the datasets also contained multiple replicate alignments for each tree. Since the inference programs had difficulties with the names of the species in the sequences, we remapped them to simple names and created a mapping file to relate the species names on the simulated tree with the corresponding names on the inferred trees. All of the parameter information is stored in the filename for the tree and the filename for its alignment for easy parsing and file identification. Code was written to automate the generation of the simulated trees and their alignments.

Table 2.1: Description of the parameters that were varied in each of the datasets. cc= ClearCut, ft = FastTree, rx = RAxML

Dataset	Sequence Length (nt)	Number of Species	Rooted	Methods	Number of Replicates
1	250, 400, 1250	128, 256	No	cc, ft, rx	1
2	250, 400, 1250, 2500, 10000	512, 1024	No	cc, ft	1
3	250, 400, 1250	128, 256	Yes	cc, ft	50
4	250, 400, 1250	128,256	Yes	rx	1
5	250, 400, 1250, 2500, 10000	512, 1024	Yes	cc, ft	10

2.4 Data Collection and Organization

2.4.1 Tree Inference

Next, the alignments generated for each simulated tree were used to obtain an inferred tree. If the inference methods were perfect then we would observe the inferred tree to be identical to the simulated tree that the alignments were evolved from. However, the inference methods are not perfect, which leads to discrepancies between the two trees. The reconstruction programs we used are: RAxML, ClearCut, and FastTree. Each program also has different parameters depending on the specific method you wish to use so we varied these as well. For ClearCut we used 6 different parameters: Neighbor-Joining (NJ) and relaxed NJ with no distance corrections, NJ and relaxed NJ with Jukes-Cantor correction, NJ and relaxed NJ with Kimura correction. There were four parameters for FastTree in the study: no distance correction and Jukes-Cantor correction for each of regular NJ and BIONJ. Since RAxML has large memory and computational requirements we only ran it with its default parameters using the GTRMIX model. The alignments are then passed to each program for each parameter combination. The result filenames contain the original tree filename plus the method used and the method parameters.

2.4.2 Post Processing

Post-processing scripts were written to compare the original “true trees” to the inferred trees and create friendly tab delimited files to be loaded into a database. The inferred trees were compared to the simulated trees using two

scoring metrics: Tip-to-Tip Distance and Monophyletic Recovery Ratio. The Tip Distance Score is based on the Pearson correlation coefficient between the two trees tip-to-tip distance matrices, the resulting correlation is coerced into a distance where 0 is identical and 1 is completely different. A lower score indicates that the true branch lengths were recovered well. The Monophyletic Recovery Ratio compares the monophyletic groups of both trees and represents the average monophyletic recovery in the result tree. A monophyletic group, specifically, is a subtree in which the tips are all of a common family. A high score indicates that the monophyletic groups were recovered well. PyCogent contains code that calculates the Tip Distance Score.

2.4.3 Data Management

Since the scale of this project is very large it would have been impossible to look through all of the data by hand. For this reason, we created a database with MySQL where all the data is stored in two tables. One of the tables contains all of the information for the original trees and the other table stores the result trees parameters and the different scoring metrics. These tables were then joined using private keys and foreign keys for efficient searching of the database for results matching any of the parameters in either table. After preparing and populating the database we turned to the Python interface with MySQL to automate queries so that they could easily be varied by all of the parameters. This helped explain how each of the parameters affected the accuracy of each of the inference methods, by allowing me to directly build graphs from the queries.

2.5 Visualization and Interpretation of Results

From the analysis I observed that the method that performed the best at recovering branch lengths, the one with the lower Tip-Distance Scores, was RAxML slightly, but FastTree performed similarly. Clearcut does not appear to recover branch lengths well as the majority of its results fell above a Tip Distance Score of 0.1, which seems to be the cut-off for both RAxML and FastTree. However, when looking at ClearCut's performance on monophyletic recovery, it did do better than RAxML and FastTree. This leads to the conclusion that ClearCut does not effectively recover tip-distances, but if only the topology of the phylogenetic tree is important it might be the best choice for analysis. Although RAxML is the best choice for recovering tip-distances and fairs well against the others on recovering monophyletic groups, it has much greater computational requirements than the other two. FastTree compared nicely to RAxML and ClearCut on both scoring metrics and has light computational and memory requirements.

Both of the scoring metrics indicate that longer sequences improve the reconstruction across all methods (Figure 2.3). This is to be expected since longer sequences allow for more data describing the evolutionary relationships between the species. Unfortunately, RAxML's computational requirements limited us to sequences up to a length of 1250 nucleotides, but there is a significant enough difference between the lengths we do have to suggest that the pattern would continue similarly to the other two methods. Although longer sequences clearly help the inference methods, sequences as large as 10,000 are not possible right now. Most sequencing techniques are providing many more sequences than they used to, but the length of the sequences remains relatively short. The results show

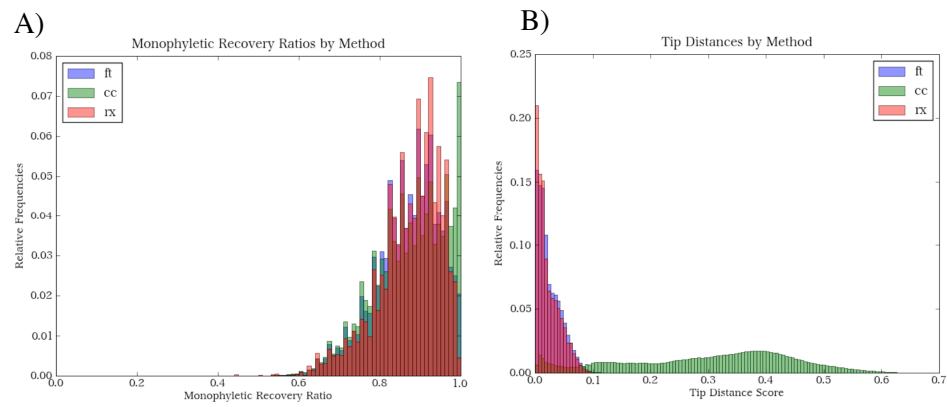


Figure 2.2: A) Monophyletic scores by method. ft = FastTree, cc= ClearCut, and rx = RAxML. RAxML and FastTree performed similarly. ClearCut performed the best at recovering monophyletic groups. B) Tip-to-tip scores by method. RAxML and FastTree tended to perform similarly over the metric, while ClearCut was not able to recover branch lengths well.

how important the length of the sequence really is to inferring phylogenies and how the current lengths of sequences limit our ability to effectively reconstruct phylogenetic trees.

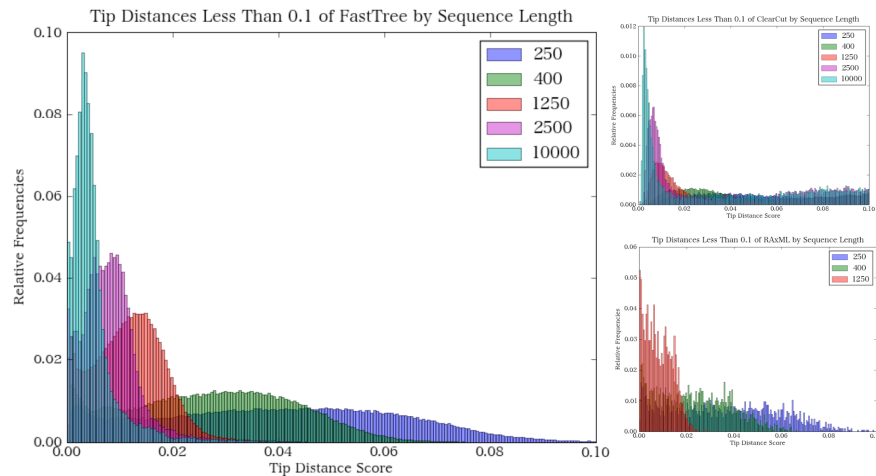


Figure 2.3: Effect of sequence length on tip-to-tip distance. As the length increases, the quality of the resulting tree improves across all methods.

I also found that independent of the method used, the number of sequences does not affect the reconstruction of the phylogenetic tree. This finding is contrary to popular belief, it is thought that an increase in sequences causes not only longer running times in a program, but also decreased accuracy. Figure 2.4 is only a subset of all of the data, to prevent bias since not all of the numbers of taxa are present in the variations of sequence length. The sequence lengths that are present are 150, 400, and 1250. We see that the number of sequences added to the tree does not appear to affect the results of these methods. Since RAxML has large memory requirements and due to memory constraints on Frost we were

unable to run RAxML on trees larger than 256 taxa, but we plan to use other computational resources to complete these results.

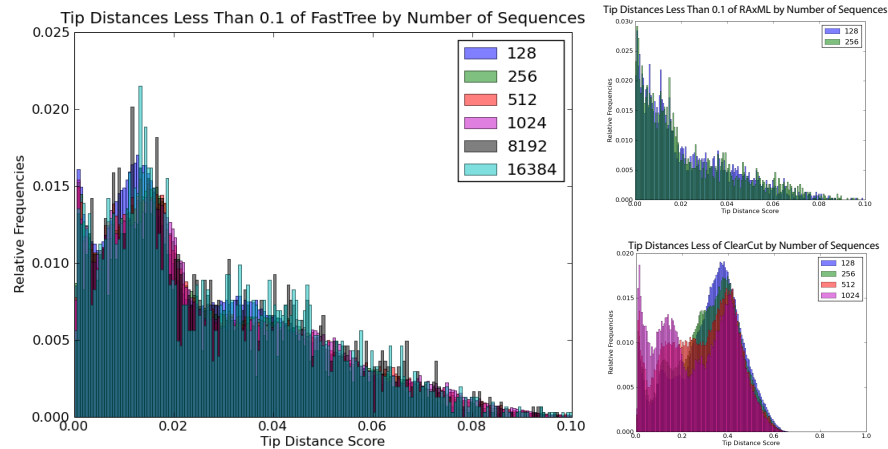


Figure 2.4: Effect of the number of taxa on tip-to-tip distance. As the number of taxa increases, the ability to correctly infer branch lengths does not change.

From the current results it seems that un-rooted trees produce higher quality phylogenetic trees, which is counter to expectations. We see that un-rooted trees recovered branch lengths better than rooted trees for both FastTree and Clearcut, but there is not a significant difference between the two in the RAxML results. The same pattern was observed with the Monophyletic Recovery Ratio. Rooting the simulated trees was thought to improve the inference since it defines the direction of change. This allows us to make better predictions about monophyletic groups so that a group does not appear to be split simply because the root is not defined. Closer examination of this finding is needed to understand why it is being seen.

When simulating the trees I varied the rate matrices that are used for evolu-

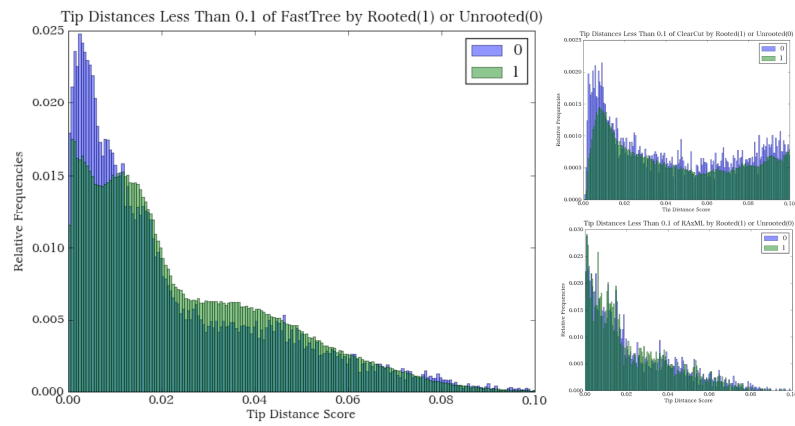


Figure 2.5: Effect of rooted vs. un-rooted trees on tip-to-tip distance. Rooting the tree did not impact the ability to correctly infer branch lengths.

ing sequences using a Markov model. I used some rate-matrices that were empirically determined and others that were random. Both the Tip Distance Score and the Monophyletic Recovery Ratio show the four strategies do not affect the reconstruction, which is a surprise. The expectation was that the empirical matrices would increase the accuracy of the inference methods when compared with random matrices.

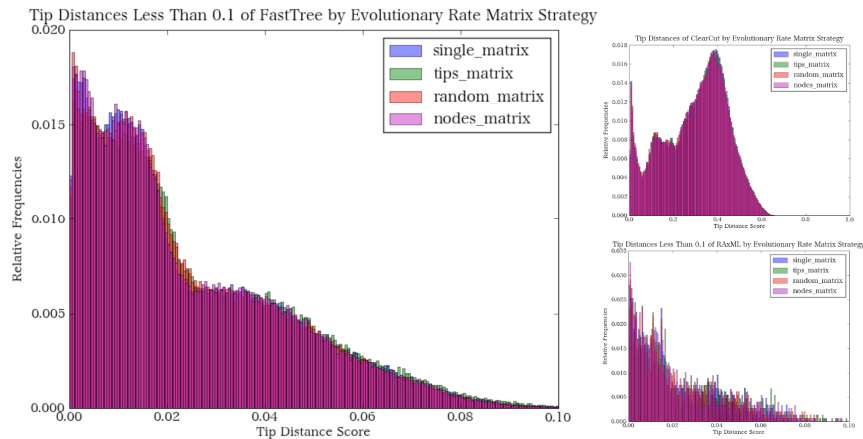


Figure 2.6: Effect of the rate matrix strategy on the tip-to-tip distance score. There is not a noticeable difference in the strategies regardless of the method.

We looked at the effect of each of the methods own parameters on the Tip Distance Score since each of the methods have parameters that use distance corrections. It was thought that there would be a significant advantage for the parameters with distance corrections to perform well in comparison to those without corrections because this score is based off of distances. FastTree’s BIONJ and regular NJ with Jukes-Cantor correction produced superior results to BIONJ and

NJ with no corrections. Looking at ClearCut's parameters all of the methods that used NJ exceeded those that used relaxed NJ performed. This is due to the fact that relaxed NJ takes shortcuts in the NJ algorithm to speed up tree inference. The distance correction methods showed improvement over their corresponding parameters without the corrections.

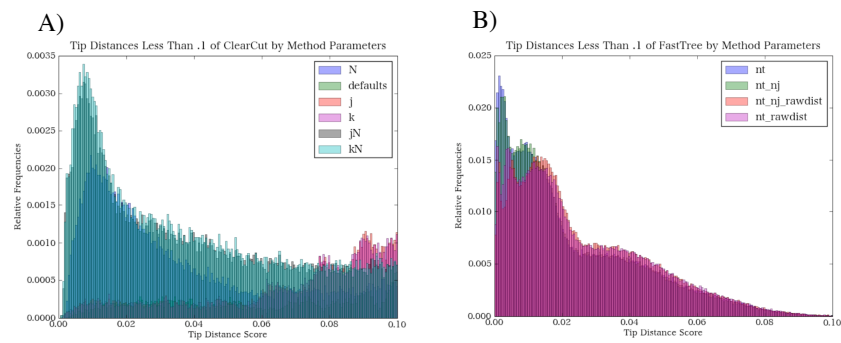


Figure 2.7: A) ClearCut parameters affect on branch length recovery. defaults = relaxed Neighbor-joining (NJ) with no distance corrections, N = NJ with no distance corrections, j = relaxed NJ with Jukes-Cantor correction, jN = NJ with Jukes-Cantor correction, k = relaxed NJ with Kimura correction, and kN = NJ with Kimura correction. B) FastTree parameters affect on branch length recovery. nt = BIONJ with Jukes-Cantor correction, nt_nj = NJ with Jukes-Cantor correction, nt_nj_rawdist = BIONJ with no correction, and nt_rawdist = NJ with no correction.

Chapter 3

Building a Seven Level Taxonomy

3.1 Motivation

After performing the phylogenetic reconstruction benchmark, we were presented with an additional project involving the greengenes and Guerrero Negro tree. We wanted to apply the knowledge we gained from the results of the phylogenetic reconstruction benchmark to further analyze the Guerrero Negro biological system. The samples that were collected from the Guerrero Negro microbial mat, the most diverse microbial community known on earth, produced about 100,000 16S rRNA sequences. From the reconstruction analysis we found that the reconstruction program FastTree gives equally good or better results and has less memory and time requirements than the other programs in the comparison. So the Guerrero Negro sequences along with the greengenes sequences were aligned using Infernal [35], and a phylogenetic tree was built using FastTree. Then our aim was to develop an algorithm with the ability to decorate the seven level taxonomy onto any tree given a mapping of taxonomic classification for each of the tips. Daniel McDonald and I worked on this project together and in collaboration with Phil Hugenholtz at DOE Joint Genome Institute.

3.2 NCBI Taxonomy

The first step in decorating a seven level taxonomy onto a given tree is to use a taxonomy classification to assign a taxonomy to each of the tips on the tree. The seven levels of taxonomy in order from the most general to most specific are: Kingdom, Phylum, Class, Order, Family, Genus, Species (Figure 3.1). The tips on the greengenes tree are all named with their prokMSA ID (prokaryotic multiple sequence alignment), this ID is simply an identifier for a 16S rRNA sequence that can be found in the prokMSA database, a database of prokaryotic 16S rRNA. Greengenes provides a mapping from this ID to the NCBI (National Center for Biotechnology Information) taxonomy ID, allowing the taxonomy to be easily retrieved from NCBI. Each NCBI taxonomy is given as a simple string separating each level with a semicolon, i.e. Bacteria; Firmicutes; Clostridia; Clostridiales; Veillonellaceae; Selenomonas; Selenomonas genomosp. C2.

3.3 Designed Objects

Since the overall expectation of this research was to decorate the seven level taxonomy onto a tree, there were some objects we developed to aid in efficiency. We developed a taxonomy tree object that subclasses PyCogent's RangeNode object to store essential properties, and perform functions that the base object is unable to do. The taxonomy node object is a single node of a tree that has knowledge of its parent and children in the tree. This node is aware of all of the tips that descend from it, and provides many of the functions needed for assigning the taxonomy, such as iterating over all the tips and storing their NCBI taxonomy

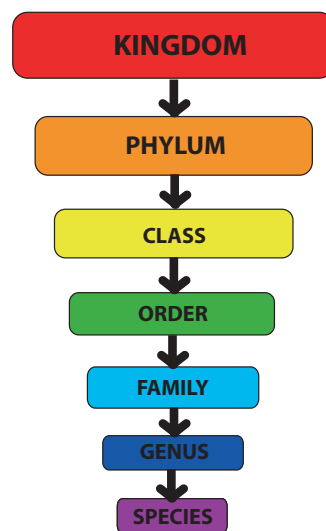


Figure 3.1: Starting from the top we have the kingdom level, which is the most general level, meaning that many organisms belong to the same kingdom, less belong to the same phylum, and species is the most specific of the seven levels.

for later use. We are also dealing with many taxonomy consensus strings, so we created an object to effectively store these strings to allow for easy retrieval of names at specific ranks as well as other basic abstract functionality such as determining the number of ranks present. This object is essentially a list, but is more suited to our application than just a list. It has knowledge of the levels, allowing for easy mapping between indices and taxonomic levels. It also has the ability to convert to and from an NCBI consensus string. Another object that was important to the overall program is a distribution object that contains the tip-to-tip distributions for the seven levels. This object can then return a z-score per rank for a given branch length. The z-score for a number gives an indication of how far it deviates from the mean of a distribution.

3.4 Assignment of Ranks and Common Names

First, we map the prokMSA ID for each tip to their NCBI taxonomy consensus string, and add a consensus object for this string as a property to each tip. Now that all of the tips have their Consensus property set, we perform a postorder traversal on the tree to cache the descending consensus objects for all nodes. Next, we do another postorder traversal to set the maximum tip-to-tip distance for each node. The maximum tip-to-tip distance is simply the longest path of all paths from one tip to another tip that descends from the node in question. For example, in Figure 3.2 node X has descendants B, C, D, E, and F. To find the maximum tip-to-tip distance we examine the paths between every pair of these nodes. Doing this we find that the longest path is the path from node C to node D, which is 0.7. Performing this calculation over and over for our large tree gets very computationally and time intensive, therefore it is important to provide a

method to cache this information. This was done in one postorder traversal where we keep track of the longest node-to-tip path at each node, and then calculate the maximum tip-to-tip distance using this information from its children nodes. This eliminates the need to re-examine paths during caching, which is another large speed up to the algorithm.

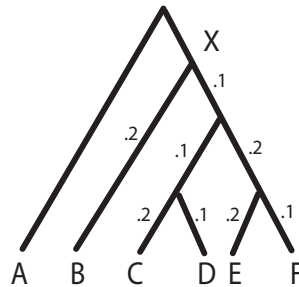


Figure 3.2: This is a simple tree that has its branch lengths labeled. Node X has a maximum tip-to-tip distance of .7, which is the distance from node C to node E.

After caching necessary information on each node of the tree we now do a preorder traversal of the tree to assign a rank to the nodes based on the taxonomy rank distributions. A node is assigned a certain rank determined by calculating z-scores. A z-score is calculated at each rank using the distribution of maximum tip-to-tip distances at that rank and the maximum tip-to-tip distance of the tips descending from that node. The rank that has the smallest of these z-scores, is the distribution that best fits the maximum tip-to-tip distance and is therefore used as the rank at that given node. This calculation is performed for every internal node in the tree.

Using the ranks and descending consensus objects assigned at each node, a common name is chosen for it. The first step is to find the highest supported

common name at the nodes rank. Given the three consensus strings: (A; B; C; D; E; F; G), (A; B; C; W; X; Y; Z), (A; B; C; D; S; T; U). If we are looking at the order level we get that the highest supported name at that rank is D with a support of 66%. The threshold of support that we used was 95%, so this node would be left without a common name. We have now set a rank on every node and a common name on nodes where the name has a support greater than or equal to 95%.

3.5 Cleanup of Missing Ranks

Now that every node has been assigned a rank based on the maximum tip-to-tip distributions, we need to check that for any path from a tip to the root of the tree every level is on the tree. This is done using a preorder traversal that examines each node except the root. If the current node has a rank that is over one rank more specific than the rank of its parent, then there are missing ranks between these nodes. For instance, if the node has the rank family and the nodes parent has the rank class then the order level is missing.

After we identified the missing ranks we needed to decide how to add them to the tree. This was done by first using the maximum tip-to-tip distance of the descendant node to calculate the z-score at all of the missing ranks. Then the list of missing ranks was split according to the rank that is closest in tip-to-tip distance, which is the rank with the lowest z-score. Next, the descendant node is given the rank that has the closest fit, and dummy nodes with a branch length of zero are created for the other ranks. The dummy nodes with more specific ranks than the descendant node are placed below it and the node with the most

specific rank contains the descendant nodes children. The dummy nodes with more general ranks than the descendant node are placed below the ancestor node. Figure 3.3 illustrates this process of adding dummy nodes. The ancestor node has the Kingdom (K) rank and the descendant node has the rank genus (G), so the missing ranks are phylum (P), class (C), order (O), and family (F). If we say that the descendant node fits the order tip-to-tip distribution the best, then we will observe the addition of dummy nodes seen in the figure.

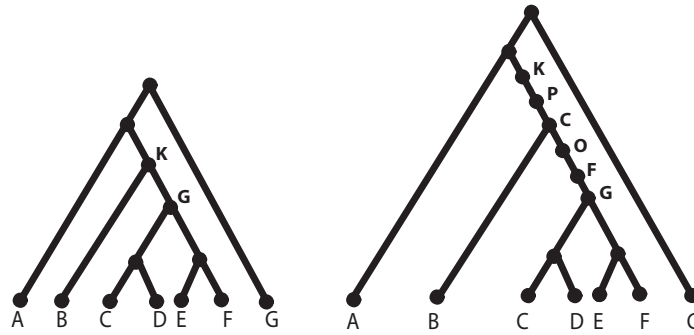


Figure 3.3: These trees demonstrate the addition of dummy nodes to include missing ranks in the design of the seven level taxonomy.

Applying this procedure alone leaves the possibility of more specific ranks being put above more general ranks, to prevent this case a check is performed before splitting the list of missing ranks. If the most general rank of the ancestor nodes children is greater than the rank identified as the closest match for the descendant rank, then that rank is used as the splitting rank. Figure 3.4 gives an example of this case. We see a problem if we are adding missing ranks between the node with the kingdom rank and the node with the genus rank, and the rank that fits the descendent nodes tip-to-tip distribution best is order. The problem is that splitting at the order rank will create a situation where ranks are out of

order. Therefore we split at the phylum level instead since it is the highest rank that we can split at without creating this problem.

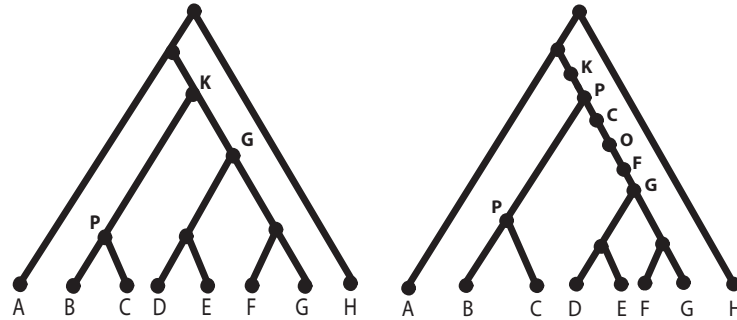


Figure 3.4: Here we show a problem that can occur when adding missing ranks. There is a possibility of adding a rank out of order into the tree, when the z-score is used to determine the proper split of ranks. The tree on the right shows that you need to look at rank order before splitting the ranks.

3.6 Cleanup of Missing Names

At this point we have a tree that has a rank on every node, and any path from a tip to the root of the tree will contain all of the seven levels. There are also common names on some nodes of the tree, but we now need to add a common name to all of the nodes. We do this by finding the next named node descending from the unnamed node that has the most leaves. This name is then used as the common name for that node, and the rank remains the same. After this step we have a fully named and ranked tree containing duplicate ranks.

3.7 Choose from Duplicate Ranks

Since on the path from any tip to the root of the tree there may be duplicate ranks, meaning one node has the rank order and its parent also contains the rank order, we must define methods for choosing the rank and name to keep. For this step the same rules apply as for the other steps. We must make sure that any node has a more general rank than any of its descendants, and a more specific rank than any of its ancestors. Another constraint is that all seven levels must remain present from the tips to the root. A few methods were developed to choose from these duplicate ranks. The first is to simply choose the rank that is found deepest in the tree, which is the node that is closest to the root. A second method is to choose the shallowest node possible without breaking any of the rules stated above. The third method is to choose the node that has the best z-score for its tip-to-tip distance relative to the empirical distribution of tip-to-tip distances for that rank. The last method is to choose the node with the longest branch length. Any of these methods would be possible for the user to choose from.

3.8 String Representation of Tree

Now that we have a tree object containing all of the information we want, we need an effective way of showing this information in a string format that is readable by many of the available tree visualization programs. This is done by naming the internal nodes with the information about the ranks and the common names. Each node has its rank prepended as the first letter, e.g. s for species, g for genus, etc. Then the rank and the name are separated by ‘_’, so each node

that has a name and rank will be printed as ‘r__CommonName’ where r is the rank and CommonName is the name the node was given from the consensus string. If the taxon was already used in the tree a number is appended onto the name, e.g. ‘r__CommonName_1’, ‘r__CommonName_2’, etc. The number is determined based on the confidence value at each node, the node with the greatest confidence has ‘_1’ appended to it the next has ‘_2’ continuing until the node with the lowest confidence. The final Newick string, which is the format used in phylogenetics to represent a tree as a string, will contain these names as the internal node names, the tips will still be named with their prokMSA ID, and the confidence values and branch lengths will remain on the tree.

Chapter 4

Review of Phylogenetic Algorithms for Detecting Horizontal Gene Transfer

The phylogenetic HGT detection methods use different algorithms to attempt to explain the discrepancies between a species tree and a gene tree. Most of the algorithms that are used for detecting HGT are also those used to calculate distance scores for comparing how similar one tree is to another. The methods that are often used for tree comparison are Subtree Prune and Regraft (SPR), Maximum Agreement SubTree (MAST), and Nearest Neighbor Interchange (NNI). Of these comparison metrics the one that is most commonly used for the detection of HGT is SPR. Other algorithms that have been used for detecting HGT events include maximum likelihood and maximum parsimony.

4.1 Subtree Prune and Regraft

The subtree prune and regraft algorithm has applications for HGT detection as well as tree comparison [53]. The SPR distance that is used for tree comparison is calculated by finding the minimum number of subtree prune and regrafts it takes to change one tree into the same topology as the tree it is being compared to. This minimum number of movements is the SPR distance, the lower the number the

closer the two trees are to having the same topology. Or thinking about it in terms of HGT, the SPR distance is the minimum amount of transfers that could have occurred to cause discrepancies between a gene tree and a species tree if the transfers are thought of as the movement of a subtree from one branch of the tree to another. Figure 4.1 shows an illustration of a single SPR move. Determining SPR distance has been shown to be NP-hard, and as a result many heuristics have been developed.

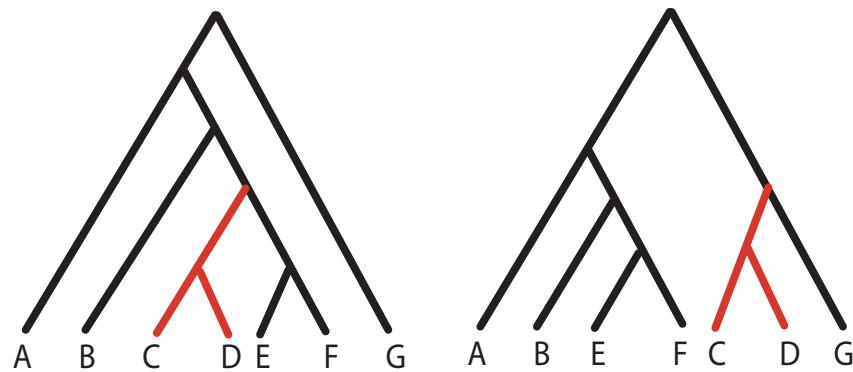


Figure 4.1: The subtree containing nodes C and D is pruned from its connection on the tree. This subtree is then grafted onto the branch between the root of the tree and the node G. This can also be thought of as a single HGT event, where the ancestor of C and D experienced an HGT event for the gene represented by this tree.

4.2 Maximum Agreement Subtree

To define the maximum agreement subtree we will start with two trees T_1 and T_2 [48]. We define L_1 as a subset of the leaves in T_1 and L_2 as a subset of the leaves in T_2 . Now T_1' is the subtree of T_1 that contains only the leaves in L_1 and

$T2'$ is the subtree of $T2$ that contains only the leaves in $L2$. Another definition we need is the definition of isomorphic. A tree $T1'$ is isomorphic to a tree $T2'$ if there exists a rearrangement of $T1'$ such that it has the same topology and order of leaves as $T2'$. If $L1$ and $L2$ contain the same leaf labels and $T1'$ and $T2'$ are isomorphic, this is an agreement subtree. The maximum agreement subtree is an agreement subtree where $L1$ and $L2$ are the largest subsets of all possible subsets that make up an agreement subtree. Figure 4.2 shows the maximum agreement subtree between two trees. Finding the maximum agreement subtree for two trees is another NP-hard problem for which there exists many heuristics.

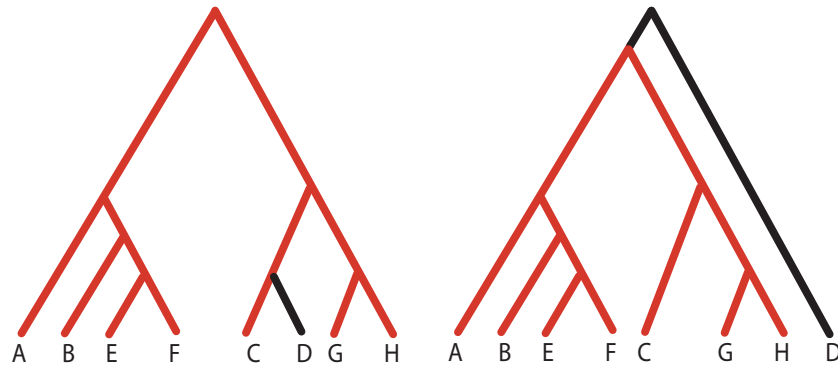


Figure 4.2: The maximum agreement subtree between these two trees is the subtree colored in red.

4.3 Maximum Likelihood and Maximum Parsimony

In phylogenetics, maximum likelihood and maximum parsimony are commonly used as methods for inferring phylogenetic trees. The maximum likelihood algorithm searches for the tree that maximizes the probability of having a certain

set of species on the tree given a set of sequences [14]. Maximum parsimony is a character-based method that looks at the possible substitutions that can be made to link all of the species together in a phylogenetic tree [11]. This method searches for the simplest way to explain the set of sequences, or that with the least amount of evolutionary change. In other words this will be the tree with the smallest length, and therefore the least number of substitutions. Finding the minimum is done by searching all possible tree topologies, and then choosing the smallest tree.

4.4 Available Methods

The HGT detection software Phylonet uses a polynomial-time heuristic for the SPR problem along with maximum agreement subtree to detect transfers [49]. They achieve this by solving the relaxed problem not the exact problem, therefore they do not guarantee that they find the smallest set of HGT events that describe the incongruence between trees. EEEP also uses an SPR heuristic, and it includes parameters for adjustment of how strict the method is [2]. The idea behind their algorithm is to use methods to discontinue searching for HGT events down a path that seems unfavorable. Horizstory implements an approximation of SPR [30]. They use efficient methods to collapse subtrees with identical topologies, thereby reducing the search size before performing SPR. Lattrans is another method using an SPR heuristic; it provides parameters for algorithm time constraint, and checks for time violations when considering a set of HGT events [1]. Nepal implements a maximum likelihood method and a maximum parsimony method. Both methods try to augment a species tree into a phylogenetic network so it fits the evolution of the gene sequence data [21, 22].

Chapter 5

Double Birth-Death Model

5.1 Importance

The double birth-death model is an algorithm that simulates phylogenetic trees with evolutionary histories that are biologically relevant to the study of horizontal gene transfer. A simple birth-death model simulates a single tree by modeling the births and deaths that species undergo through time, simulating the evolutionary history relating all of the species that exist at the final iteration. The double birth-death model is a model that simultaneously simulates a species tree and multiple gene trees. This model also allows for the simulation of trees with known horizontal gene transfer events. These simulated trees can then be used to benchmark phylogenetic HGT detection methods, because of their ability to produce datasets where we know the correct answer. Another way commonly used to assess the accuracy of phylogenetic HGT detection methods is to start with a single simulated tree and generate the gene trees by simply moving subtrees from the branch they reside on to another branch in the tree. This method, however, does not account for the births and deaths that can occur in both the gene trees and the species during evolution, leading to a more incomplete biological model. It is therefore important to use the double birth-death model to effectively compare the HGT detection methods.

5.2 Algorithm Implementation

The current implementation in PyCogent, which I implemented a significant section of, allows the user to simulate a species tree, gene trees and horizontal transfer events given their specific parameters. The user has the ability to set parameters such as the number of gene trees, the number of species found on the species tree, the rate of transfer, and several others. The model starts with a root node for the species tree and a root node for each gene tree. Then, at each time-step (iteration) all of the species nodes at that current time-step are given the chance to either live, die, or split (birth). The gene trees also evolve at each iteration. If a species node splits then the corresponding nodes in the gene tree will also split, and if a species node is deleted then the corresponding nodes in the gene tree will also be deleted. However, all of the current gene nodes are also given the opportunity to live, die, split, or be transferred to another species at each time-step. If every current gene node only lives, meaning there are no transferred, duplicated, or deleted genes at any iteration of simulation, then the gene trees will all be identical to each other and the species tree. As each iteration is performed, the current number of time-steps determines the branch length of each node. The user specifies the amount of branch length that is added at each time-step. Figure 5.1 gives an example of the double birth-death algorithm using two gene trees.

- At time step 0 we start with a root node for the species tree and a root node for each of the two gene trees. (Figure 5.1 A)
- At time step 0 gene tree 1 node 0 has a birth. (Figure 5.1 B)
- At time step 0 species node 0 has a birth. (Figure 5.1 C)
- At time step 1 gene 1 node 2(on right) dies. (Figure 5.1 D)
- At time step 1 species node 1 and 2 both have births. (Figure 5.1 E)

· At time step 2 gene 2 node 4 transfers to node 6 (transfer of gene 2 from species 4 to 6). (Figure 5.1 F)

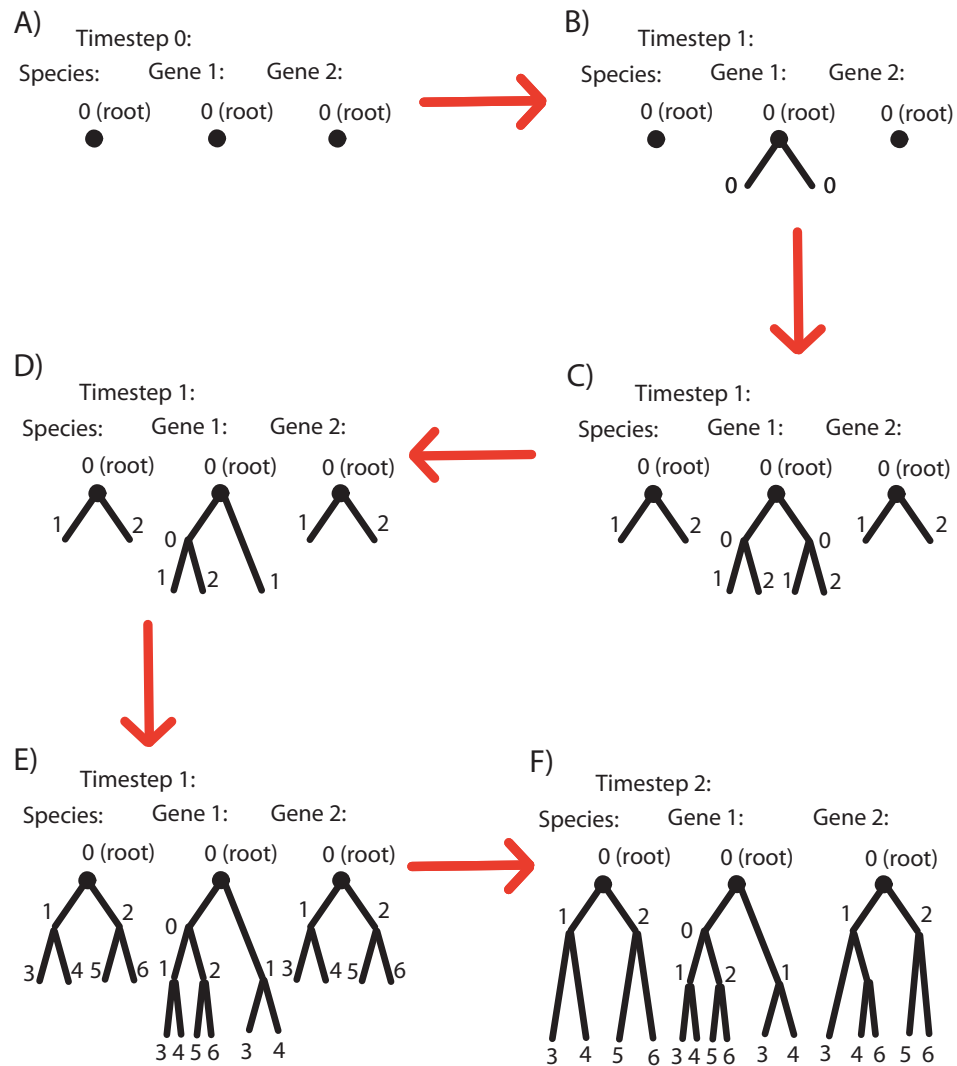


Figure 5.1: This is an example of how the double birth-death model works. Branch length is not represented in the figure.

Chapter 6

Infrastructure for Running High Throughput Tests

Designing an efficient way for performing high throughput tests is necessary for success of my research. Without planning the method of running the tests there is often time and resources wasted. Another advantage to designing the method for running these tests is that after the benchmark is complete, the design can be easily turned into a pipeline for users of the software being tested. In this case both the phylogenetic reconstruction study and the benchmark of phylogenetic horizontal gene transfer detection methods can be easily hooked up together to aid in the detection of HGT. There are some important steps that need to be taken which I have applied to my HGT benchmark, the first of which is get to know the system you are performing the tests on.

6.1 Gaining Knowledge of the System

Before even trying to start running tests or gathering data, we need to understand the best way to design the test based on the capabilities and limitations of the computer system we are using. For the HGT detection benchmark all tests are being run on the Steele cluster at Purdue, which has 893 dual quad-core Intel E5410 processor compute nodes, and is part of the TeraGrid scientific research infrastructure. After working with Steele for a short amount of time, I determined

that there are five queues that are available for my use. The five queues go by the names: `standby`, `standby-8`, `steele_hold`, `tg_short`, and `tg_workq`. Table 6.1 shows the five queues. Although `steele_hold` has unlimited wall time the jobs must be approved before running, and from experience, this can take upwards of a few days. Since I am aware that as the size of my trees grow the HGT detection methods slow down considerably, I can optimize my time. I can run the smaller trees in the `standby` and `standby-8` queues, while the larger trees will have to be run on `tg_short` or `tg_workq`. Something else I discovered while using Steele is the file and memory limits put on the scratch directory where I keep all of the data collected. There is a 10,000-file limit, which seems like a lot until you are running high throughput tests. To get around this I sectioned the data based on the size of the trees, and when a dataset was not running it was tarred up.

Table 6.1: Here is an overview of the queues available to me on the Steele cluster at Purdue.

Queue	Walltime
<code>standby</code>	4:00:00
<code>standby-8</code>	8:00:00
<code>tg_short</code>	72:00:00
<code>tg_workq</code>	720:00:00
<code>steele_hold</code>	unlimited

6.2 Simulating Appropriate Datasets

The next step to running high throughput tests is to obtain the data necessary for the tests. Often this is a simulated dataset, or a well-studied and well-known dataset. I used simulated data because it provides a “true” set of transfers for artificial data, which will allow me to test the accuracy of each of the

detection methods. This gives the ability to evaluate the power of each method by comparing the true set of transfers with the detected set of transfers for that method. For the HGT tests I first decided which parameters to vary when simulating trees, such as the number of taxa, so that we have significant changes in trees, but a realistic number considering computation time. I used the double-birth-death model implemented in PyCogent [25] to simulate random gene trees and a species tree with HGT histories. I also used SeqSim in PyCogent to simulate sequences for the gene trees and species trees. To start I simulated sequences of length 1250 for the species tree, and random lengths for the gene trees since not every gene is of the same size. As a complete dataset I have many species trees, each with sequence alignments, a series of gene trees each with a sequence alignment, and a set of simulated HGT events. I wrote code to automate the generation of the simulated trees and their corresponding alignments.

6.3 Running Detection Programs on Datasets

After spending some time installing all of the software on Steele and running test datasets, I was set up to begin using the programs on my simulated data. To make this process easier and more efficient I developed program application controllers for the software that implements the detection methods to allow for easy adjustment to the different types of data input of the methods. The programs are currently being run on the simulated data. I also wrote parsers for the different formats of output from the programs so that the data is more easily readable. The reason for doing this is that trying to manually go through all of the output, which can sometimes be millions of lines, would take days or weeks, whereas by applying a standardized and automated framework for collating and processing results, the

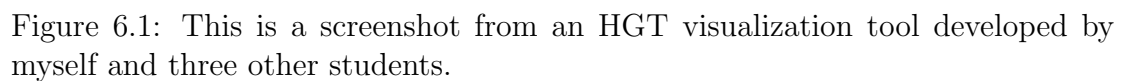
output is parsed quickly and returned in a way that is easy to understand. This also gives one common output style for all the programs so that it is easier to compare them with one another. In addition, I wrote code to completely automate the process, which will only need small revisions to adapt to a user interface for HGT detection.

6.4 Data Management and Analysis

The next step when all of my simulated data is finished running and parsed, is to have an effective plan for organizing all of the data to allow for easy analysis. Since the scale of this project is very large it would be impossible to look through all of the data by hand. For this reason, I will create a database with MySQL, where all the data will be stored in tables and parameters separated by columns. This will allow me to easily and efficiently store the parameters used to simulate the data, the method used, the percentage of the simulated transfers that were detected and the percentage of false positives (transfers detected by the method that are not actually transfers). We can then generate plots for each of the varied parameters where each method is a separate point with one axis being the number of correct transfers detected and the other axis the number of false positives given by that method. This will give us a good indication of the power each method has at detecting HGT events.

6.5 HGT Visualization

Now that I have written application controllers and parsers for these HGT



Chapter 7

A New Network Based Method for Detecting Horizontal Gene Transfer

The aim of this research was to develop a method for detecting HGT events using network statistics and a phylogenetic species tree. The program requires as input a file containing all of the gene sequences in Fasta format and labeled with the species they belong to, and a species tree containing all or a subset of the species represented in the sequence file. The output is the detected horizontal transfer events that have occurred between the species. I implemented the algorithm myself, and the some of the data collection and analysis was aided by three others.

7.1 Data Collection

Since we need to effectively test the reliability of the detection method we used simulated data. Simulated data is used because it provides a “true” set of transfers for artificial data, which will allow us to evaluate our method by comparing the true set of transfers with the detected set of transfers. We simulated random gene trees and a species tree with HGT histories using double birth-death model in PyCogent. Then we used a Markovian evolutionary model implemented

in the PyCogent SeqSim Module to simulate sequences for the gene trees and the species tree. We then labeled the sequences with an identification of the gene tree that they were in and the genome that they belong to. We then combined all of the gene sequences into one file, giving us the required input. We randomly varied the lengths of the alignments for the gene trees since it is more biologically correct to have genes of different sizes in the genome. To start we used a small dataset of 20 species each with about 10 genes (can be more or less due to gene death).

We also hoped to apply the detection method on biological data to see how the method works on a large and real dataset. We gathered the bacterial gene sequences from KEGG (Kyoto Encyclopedia of genes and genomes) using the KEGG ftp [23]. We isolated the nucleotide sequences of the complete genomes of 908 bacterial species and combined all of the genomes into one file. We removed sequences that are smaller or larger than a certain threshold so that they don't bias the results. The script used to combine the gene sequences also built a file containing all 16S rRNA based on the labels of the sequences given by KEGG. These sequences were then run through the alignment software MUSCLE [12]. After obtaining this alignment we built a phylogenetic tree with the reconstruction software FastTree.

7.2 Clustering

We used CD-Hit to perform the clustering of the sequences by similarity [28]. The workflow that we created allows the user to identify the parameters they want to use with CD-Hit, otherwise we use default parameters. We used some of the scripts in the software QIIME (Quantitative Insights Into Microbial

Ecology) for our workflow [6]. To begin we used a script called `pick_otus.py` that is able to take the combined sequence file and run CD-hit on it given the users parameters. It returns a table in the format of a number identifying the cluster followed by a list of sequences that belong to the given cluster. This output is then fed into another script called `make_otu_table.py`. This script just formats the cluster data in another way so that it can later be used in a different script.

7.3 Building the Network

This is done using the script `make_otu_network.py`, which is also in QIIME, and written by me. It takes as input the otu table output by `make_otu_table.py`, and it outputs a file that is easily readable by Cytoscape, a network visualization software [43]. This file is then loaded into Cytoscape, which immediately loads networks into a grid layout, this layout is not a useful way for visualizing our data. The layout we chose to look at was called BiLayout, this layout allowed us to view the cluster nodes separately from the species nodes giving a better view for interpreting the data.

7.4 Interesting Results

In our simulated dataset from the double-birth death model we have an output of transfers that have occurred, Table 7.1 shows these transfers. Figure 7.2 shows the ancestral evolutionary relationship between each of the species in the simulated data. The nodes in the tree are labeled and the transfers in Table 7.1 correspond to the nodes on this species tree. From looking at the species

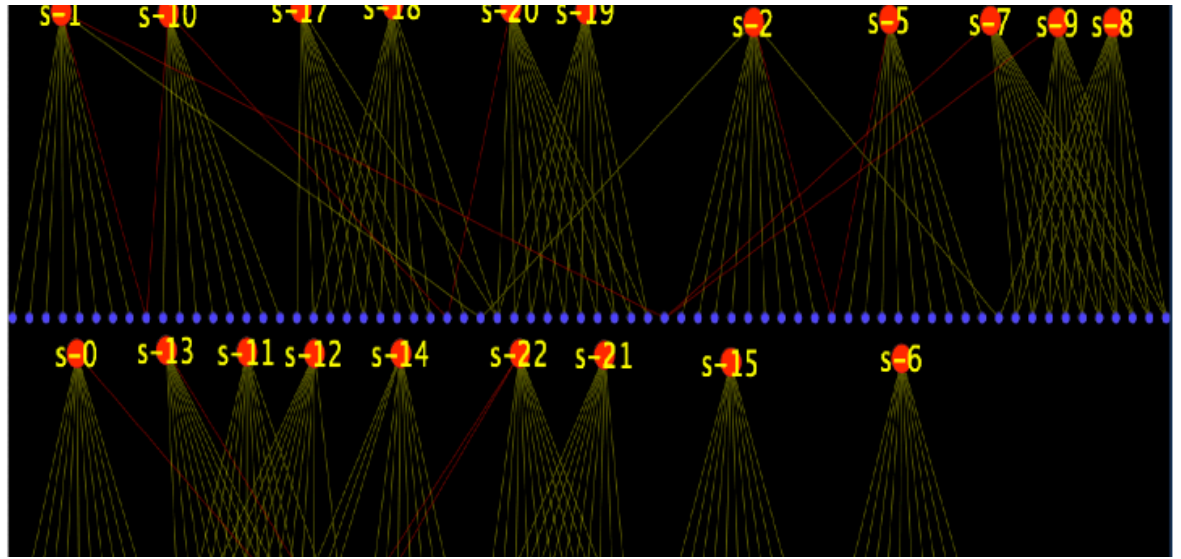


Figure 7.1: The network we obtained from clustering the gene sequences simulated along the gene trees from the double birth-death model

tree we can see the vertical relationship between species and differentiate these relationships from HGT events when we look at the network. Lets examine the network in Figure 7.1, look at the node in the network that has the label s-13, which corresponds to the tip numbered s-13 in the species tree. This has many of the same connections to clusters as the nodes labeled s-11 and s-12. If you look at the species tree in Figure 7.2, you expect to see many connections between these nodes because of the close ancestral connections between these species. If you look closer there are a couple connections with clusters that are also connected to the node labeled s-22. When you look at the species tree for these nodes, there is no close connection between species s-22 and s-13, so we call this an HGT event, now if we refer back to Table 7.1 we see that there is an HGT event between species s-22 and species s-13. Our method correctly identified this transfer. Now that we have been able to observe a relationship between the HGT events and these abnormal connections in the network, we need to think of a way to automate this process.

Table 7.1: HGT events from the double birth-death model.

Donor	Recipient	Gene Tree
41	0	8
5	2	9
1	29	7
0	22	1
10	1	2
10	20	9
14	2	8
13	22	9

7.5 Algorithm to Automate HGT Detection

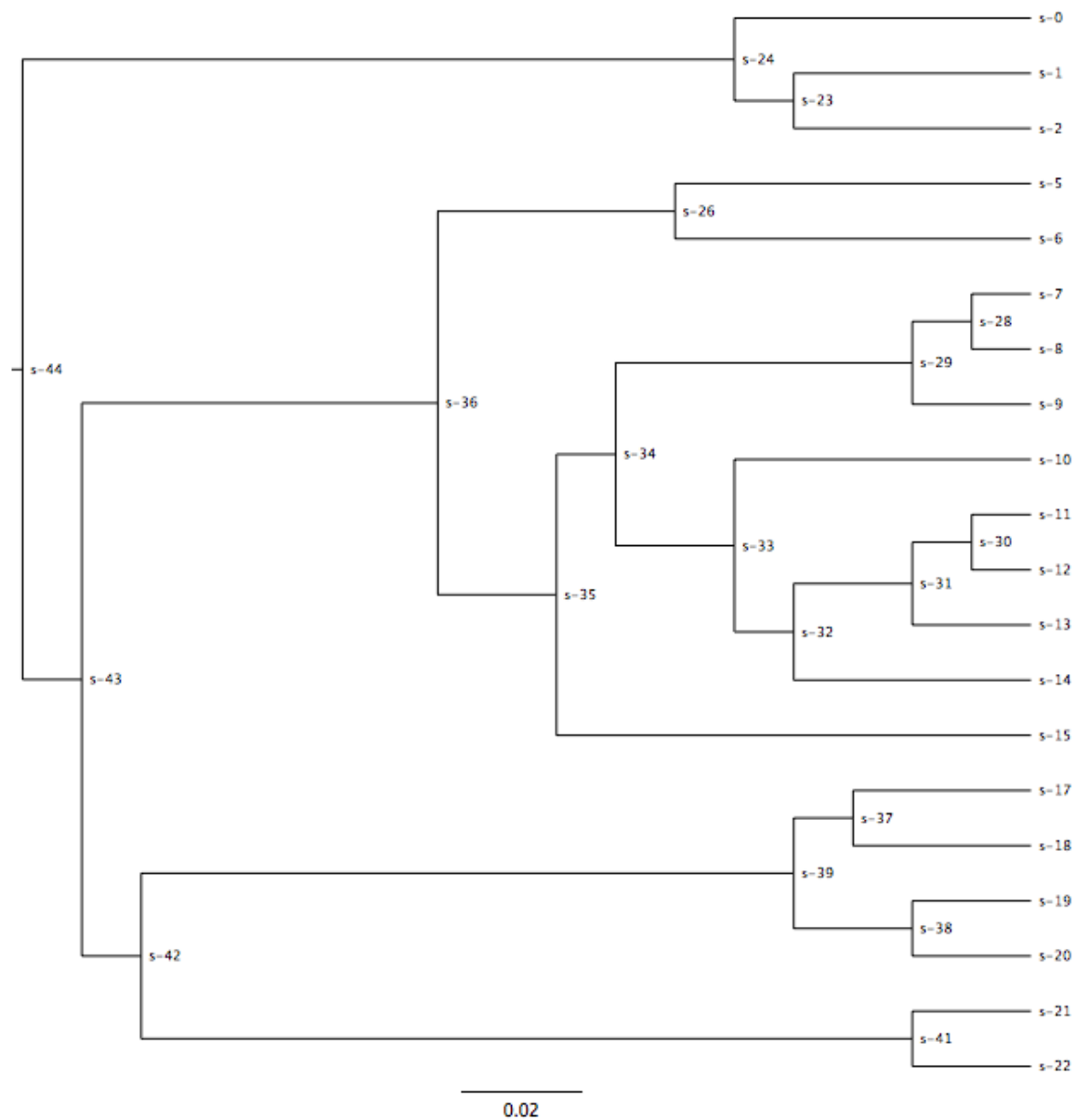


Figure 7.2: Species tree from double birth-death model

Our next step is to devise an algorithm that will take us away from needing to visualize the data in cytoscape every time. First, since clustering at different similarity levels can provide us with information relating to different levels of the species tree, it would be useful to look at multiple similarity levels. This is done by iterating through the similarity levels in CD-Hit, the user specifies the lowest similarity level and the number of levels they wish to iterate over. At each iteration CD-Hit gives a file where each line in the file is a list of the species that are in a cluster. The algorithm looks at the species tree and finds the last common ancestor of the species in this cluster. We expect that a transferred species will not be in the same subtree as the other species in the cluster because these species are clustering together due to their close ancestral relationship, while the transferred species is clustering due to the horizontal gene transfer. Figure 7.3 illustrates this step.

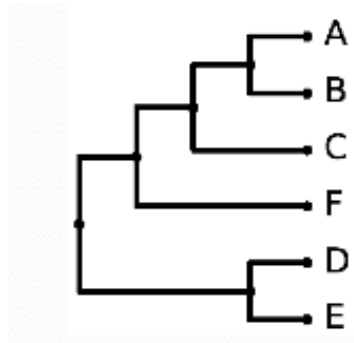


Figure 7.3: Say we have a cluster that is [A, B, E]. On this tree the last common ancestor of these tips is the root. The root has tips A,B,C,D,E, and F while the cluster only has three of these five tips, so there is a high likelihood that there is a transfer here. Further investigation is used to determine the actual transfer.

During this iteration we also determine the tip-to-tip distance for each pair

of nodes in the cluster. As the clustering similarity decreases we see larger clusters that correspond to deeper nodes in the species tree. Using this knowledge and the tip-to-tip distance, we expect that nodes that have smaller tip-to-tip distances will cluster together because they are more closely related. This idea is shown in Figure 7.4.

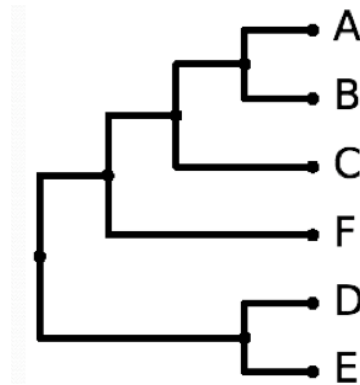


Figure 7.4: If we have the above tree as the species tree, and we have two clusters: cluster 0: [A, B] and cluster 1: [A, E], we expect A and B to cluster together, but there is a good chance that a transfer event occurred between A and E.

In order to decide the direction of transfer, meaning which species is the donor and which is the recipient, the algorithm keeps track of the transfers detected at the higher similarity. Then if a cluster at a lower similarity contains all of the nodes that clustered for the transfer detected at the higher similarity, and the species descending from the transfer node you can infer direction as seen in Figure 7.5.

Using my method on the same dataset that was put into Cytoscape we obtain the output in Figure 7.6. We see that the method detected five of the eight

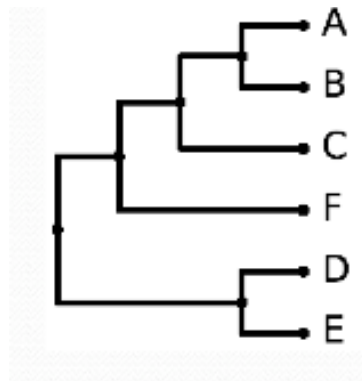
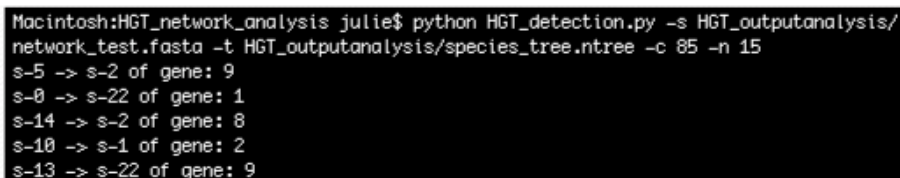


Figure 7.5: If we have the above species tree and a cluster at 98% similarity: $[A, E]$, and a cluster at 95% similarity: $[A, B, E]$. We can say that a transfer most likely occurred from A to E because A and B are clustering with E so E has a sequence similar to both A and B. If a transfer occurred the other way you wouldn't expect to see clustering like this.

transfers, with no false positives. I also ran Horizstory, Phylonet, and EEEP on this dataset. Horizstory identified six of the eight transfers and no false positives at their strict consensus, and six of the eight transfers and two false positives at their majority consensus. Horizstory also took about four times as long to run as my method. Phylonet was the fastest of the methods, but it only identified two of the transfers and returned one false positive. EEEP ran about the same time as my method, and identified two of the transfers with two false positives. Another trait that my program has that EEEP and Phylonet do not have, is the ability to handle paralogs (gene duplications). EEEP just has a segmentation fault when a tree contains paralogs; while Phylonet fails and tells the user there is a duplicate node. Horizstory seems to allow paralogs and does not seem to fail, but I do not know how it handles them. I am also interested in applying this method to more simulated datasets, and biological datasets in the future.



```
Macintosh:HGT_network_analysis julie$ python HGT_detection.py -s HGT_outputanalysis/
network_test.fasta -t HGT_outputanalysis/species_tree.ntree -c 85 -n 15
s-5 -> s-2 of gene: 9
s-8 -> s-22 of gene: 1
s-14 -> s-2 of gene: 8
s-18 -> s-1 of gene: 2
s-13 -> s-22 of gene: 9
```

Figure 7.6: This is the output from the automated HGT detection program, the method detected five of the eight transfers

Chapter 8

Compositional Methods for Detecting Horizontal Gene Transfer: Stand Alone Version of CodonExplorer

8.1 Codon Explorer

CodonExplorer is an online tool that has the capability to connect to the KEGG (Kyoto Encyclopedia of Genes and Genomes) or NCBI databases, allowing the user to choose either entire genomes or specific genes from these databases [18, 54, 52]. CodonExplorer then provides the user with the ability to analyze codon usage and sequence composition using tables of codon usage and visualizations such as histograms, heatmaps and scatter plots. Analyses like this have importance in determining mutational effects on codon usage, finding possible highly expressed genes, and identification of horizontal gene transfer (HGT). Since CodonExplorer has the ability to examine the differences that specific genes exhibit from the remainder of the genome, it can be used as a compositional method of detecting HGT.

Although this web interface exists, the database has not been updated in a year or more. This version therefore does not give a user the ability to use any dataset they wish, it only allows them to use the datasets in the current status of the database. Since this is not always a desired feature, I have developed a stand

alone version of CodonExplorer. I used many of the statistical and graphing methods available in PyCogent to put together this version. In addition, parts of the graphing code needed to be updated for compatibility with more current versions of Matplotlib. Now the user has the ability to use any one genome or multiple genomes for comparison. They can also name specific genes to compare against the whole genome. The development of this stand alone version was important for this research work and will be useful for others in the future. I made the change over to the stand alone version; the analysis was done by myself and Ryan Walters.

8.2 Rationale

For over 100 years *Enterococcus faecalis* has been a well established cause of endocarditis and more recently has been shown to be a significant source of nosocomial infections [34, 16]. Although normally part of the intestinal flora, increasing medical concerns over *Enterococcus faecalis* infections stems from their innate and acquired resistance to various antimicrobial agents [33, 46]. Moreover, evidence supports the notion that Enterococci are able to transfer genes providing antimicrobial resistance between both themselves and bacteria going through the colon [42]. *Enterococcus faecalis* have the ability to acquire resistance to vancomycin, an antibiotic often considered the last line of defense, making these pathogens extremely difficult to treat clinically [7]. *Enterococcus faecalis* ATCC 29200 and other clinically isolated strains have been chosen for genomic sequencing largely to provide insight as to how Enterococci acquire resistance to antimicrobial agents [37].

Our overall goal was to use CodonExplorer to examine the genomic sequence of *Enterococcus faecalis* ATCC 29200 and look for atypical regions. Using the knowledge that codon usage is non-random, tends to be consistent across a genome, and varies considerable among organisms we can infer the reasons behind the areas of the genome that differ [17]. For example, looking at GC content or codon usage to find genes that exhibit a difference from the bulk of the genome [36]. Finding these regions of differing composition can provide us with insight about which genes in *Enterococcus faecalis* ATCC 29200 may have been horizontally transferred or are highly expressed [24, 20].

To begin our analysis we used code in PyCogent to generate Monte Carlo histograms comparing every gene in the genome to the genome as a whole [25]. This is to determine which genes show significantly different Codon Adaptation Index (CAI) values than the majority of the genome [44]. CAI values are determined using a representative set of genes known to be highly expressed and gives a value that predicts the expression level of a gene. The CAI values are calculated using the formula below; we used a function in PyCogent that implements this equation and used genes that encode ribosomal proteins as the representative set of highly expressed genes. Highly expressed genes typically have high CAI values.

$$CAI = \frac{(\prod_{k=1}^L RSCU_k)^{\frac{1}{L}}}{(\prod_{k=1}^L RSCU_{kmax})^{\frac{1}{L}}}$$

In this equation RSCU is relative synonymous codon usage, this table is built using the reference set of highly expressed genes in the organism. RSCU_k is the RSCU value for the kth codon and RSCU_{kmax} is the max RSCU value.

A one-tailed t-test is used to compare the Monte Carlo histograms and provide us with a probability that the difference between the histograms is more than what we would expect to see by mere chance. We now identify all of the genes with a p-value less than .05 on the CAI Monte Carlo histograms and generate scatter plots, heat maps, and codon usage tables for each of them. The scatter plots and heatmaps are plots of CAI vs. P3 where P3 is the GC content at the third codon position. One change we added to our code that varied from the CodonExplorer functionality was to include a distinct blue dot on the scatter plot and heatmaps that corresponds to the given gene. This is to let us visualize the difference of that specific gene from the rest of the genes in the genome that are on the plot. The scatter plot is useful to see exactly where there are points on the plot, but in areas of many overlapping points it is easier to visualize the areas of concentration by using a heat map. From further analysis of these plots and tables we identify possible horizontal gene transfer events or highly expressed genes.

8.3 Challenges

When beginning analysis using CodonExplorer we attempted to use the web interface (<http://bmf.colorado.edu/codonexplorer/>) to download our genome from the NCBI database. We tried to search for our genome, but the search returned an error, leading us to believe that the database underlying the interface has not been maintained. Another setback of this tool is that it can only be used through this web interface; there currently is no standalone version for CodonExplorer. We found out that the majority of the functions that are performed by the webapp are actually current PyCogent Modules. So we used the PyCogent modules to do exactly what CodonExplorer does. Most of the functionality that we needed

is in `cogent.draw.codon_usage` and `cogent.core.usage`. We also discovered that we needed to use some of the modules in `cogent.maths.stats` to calculate the CAI values and the one-tailed t-test. Some modifications had to be made to allow for compatibility with Matplotlib, which is the software used by PyCogent to generate plots. For the Monte Carlo histograms we obtained the code used by CodonExplorer and created a standalone version that is disconnected from any databases. The user just passes in the GenBank file or the genome accession number so that the GenBank file can be downloaded from NCBI[4]. Since the GenBank file for this genome is actually empty we built the GenBank file using IMG/M and then downloaded it[32]. We were then able to use the GenBank parser in PyCogent to load the genes into CodonUsage objects, which were used to calculate the values needed for the various plots and tables that we generated.

8.4 Results

A codon usage table of the entire genome was generated (8.1 A) and revealed a low average CG content. In particular, the third position of the codon had a CG content of 30.44%. Of the approximately 3000 genes analyzed, 71 genes were identified with a p-value less than .05 on the CAI Monte Carlo histograms. 20 of the genes identified were tRNAs and ribosomal RNAs. It is not surprising that these structural RNAs were identified given their higher CG content required for structural stability. Ribosomal proteins (`rpmL`, `rplU`, `rpmF-3`, `rplK`, `rplA`, `rplJ`, `rplL`, `rpsB`) and translational elongation factors (`efp`, `tsf`, `tig`) were identified as being highly expressed (Figure 8.2). The Monte Carlo histogram and codon usage table are shown for 50S ribosomal protein L21 (`RplU`) as representative gene from this set. An additional subset of genes were identified as being involved in

the glycolysis pathway. These genes include endo-1,4-beta-glucanase (CelA-4), glyceraldehyde-3-phosphate dehydrogenase (GAP-2), and polyphosphate glucokinase (pgk) (Figure 8.3). Interestingly, a gene belonging to the merR family of transcriptional regulators was also identified (Figure 8.4). This gene shows dramatically different codon usage, primarily in the third position.

Although sequence composition and codon usage provide insight into potential HGT events, there are some disadvantages to using this computational approach. Primarily, as shown in the results section, ribosomal proteins are often identified as having been potentially horizontally transferred. This however is an artifact due to the sequence composition of this class of gene [54]. Additionally, ancient genes that may have been horizontally transferred are unlikely to be detected as their sequence will eventually reflect the sequence composition of the host [26]. It is also possible that species undergoing the transfer share enough similarity in codon usage and sequence composition that the HGT cannot be computationally detected. However, we were able to identify a couple of genes whose composition is abnormal relative to that of the genome and whose functions provide information on the lifestyle of *Enterococcus Faecalis*. For example, the gene Cel4-A is endo-1,4-beta-glucanase. These enzymes are able to break down 1,4-beta-glucans, which are structural components of plant cells walls, primarily in barley and oats. Since *Enterococcus Faecalis* is found in the intestine this could serve as a mechanism for nutrient availability.

We also identified a merR family transcriptional regulator which is intriguing since the merR family has been implicated in multidrug as well as mercury

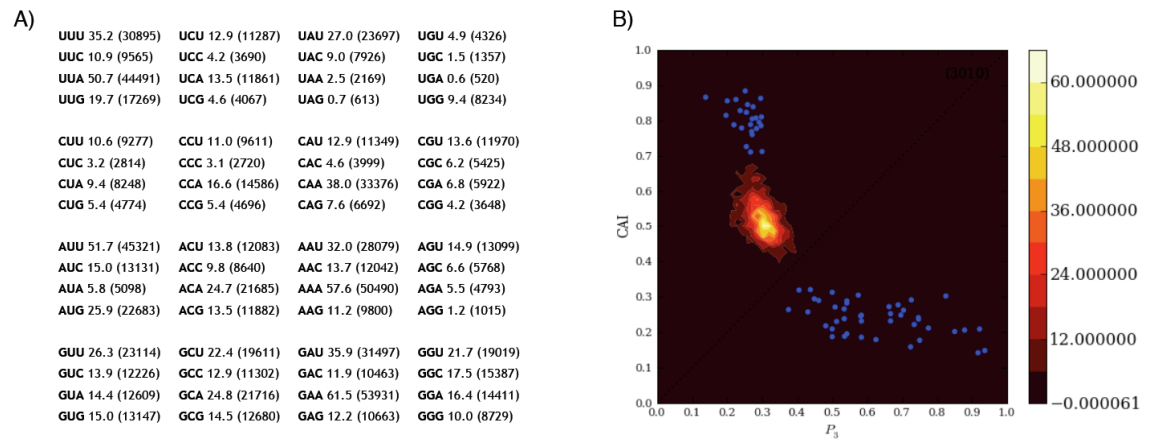


Figure 8.1: A) Codon usage table of 877,285 codons for 3010 genes in *E. Faecalis*. Fields: [codon triplet] [Frequency: per thousand] ([number]). Coding GC content: 38.09%; 1st letter GC: 48.98%; 2nd letter GC: 34.85%; 3rd letter GC: 30.44%. B) Heat map plot of CAI vs P3 for all genes identified with a p-value less than .05 on the CAI Monte Carlo histograms.

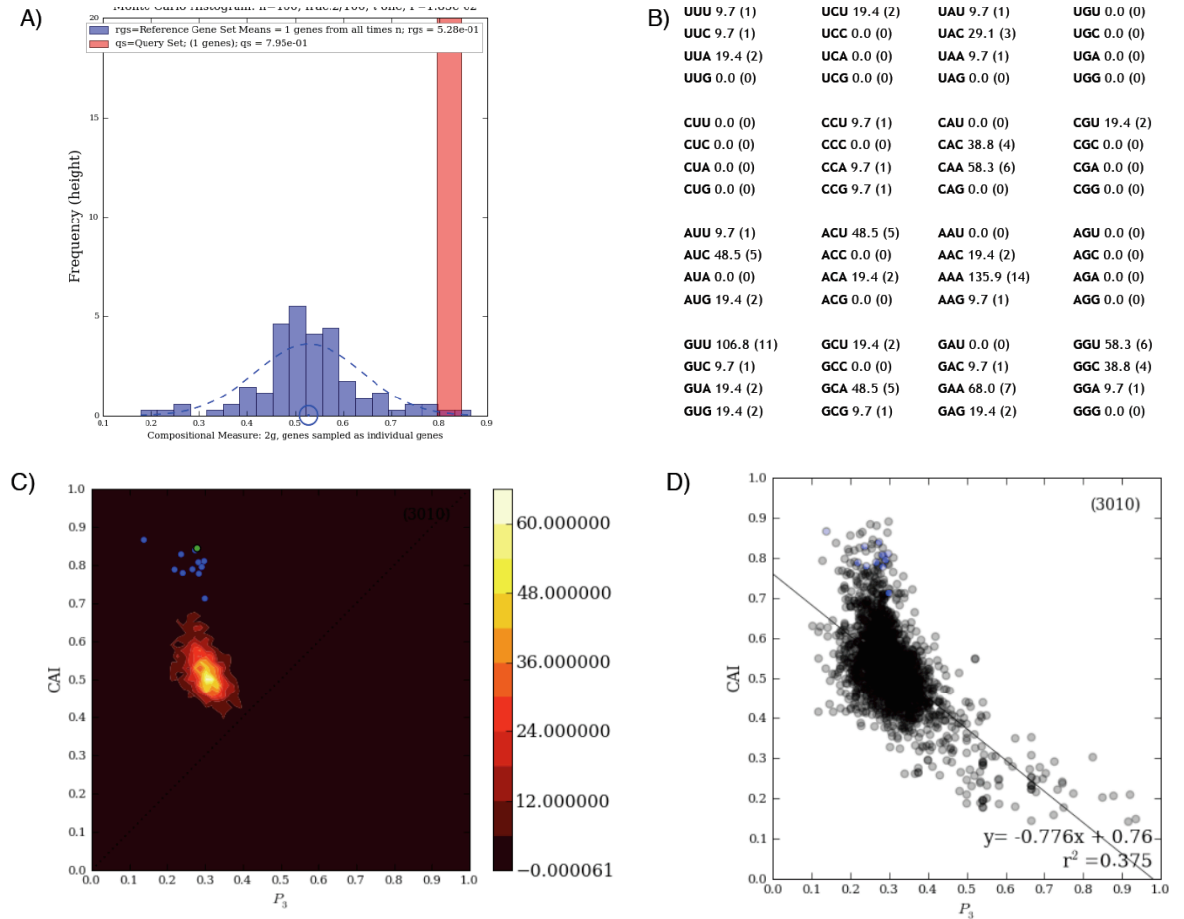


Figure 8.2: Monte Carlo histogram plot (B) and codon usage table (A) for 50S ribosomal protein L21. Coding GC content: 39.81%; 1st letter GC: 58.25%; 2nd letter GC: 32.04%; 3rd letter GC: 29.13%. (C) Heat map plot of CAI vs P_3 . 50S ribosomal protein L21 is shown in green, addition ribosomal proteins and translational elongation factors shown in blue. (D) Scatter plot of CAI vs P_3 , ribosomal and translational elongation factors are shown in blue. Ribosomal proteins and elongation factors include: rpmL rplU, rpmF-3, rplK, rplA, rplJ, rplL, rpsB, efp, tsf, and tig.

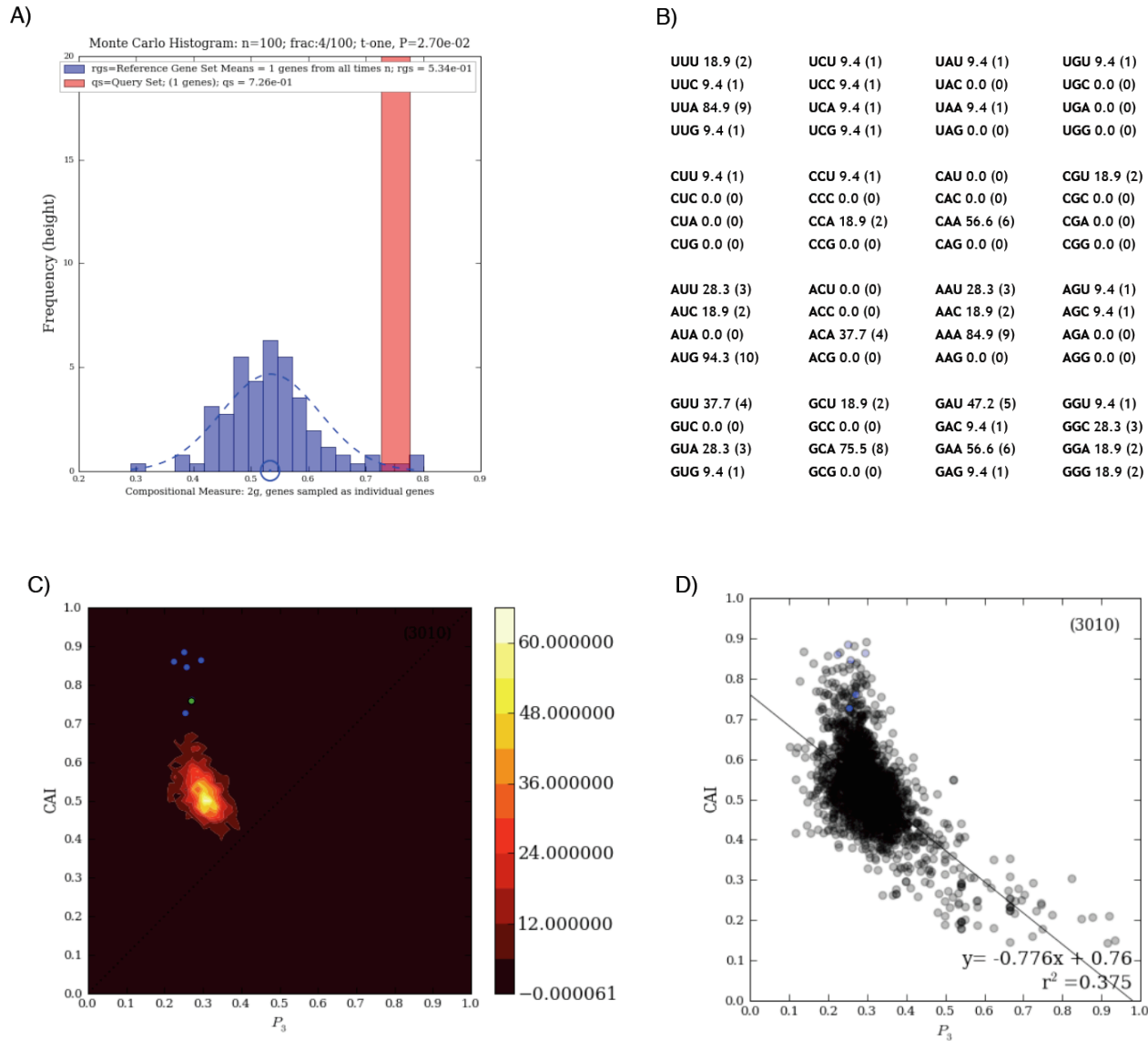


Figure 8.3: Monte Carlo histogram plot (A) and codon usage table (B) for CelaA-4, an endo-1,4-beta-glucanase. Coding GC content: 35.22%; 1st letter GC: 48.11%; 2nd letter GC: 32.08%; 3rd letter GC: 25.47%. (C) Heat map plot of CAI vs P_3 . CelaA-4 is shown in green, 5 other genes involved in glycolysis are shown in blue. (D) Scatter plot of CAI vs P_3 .

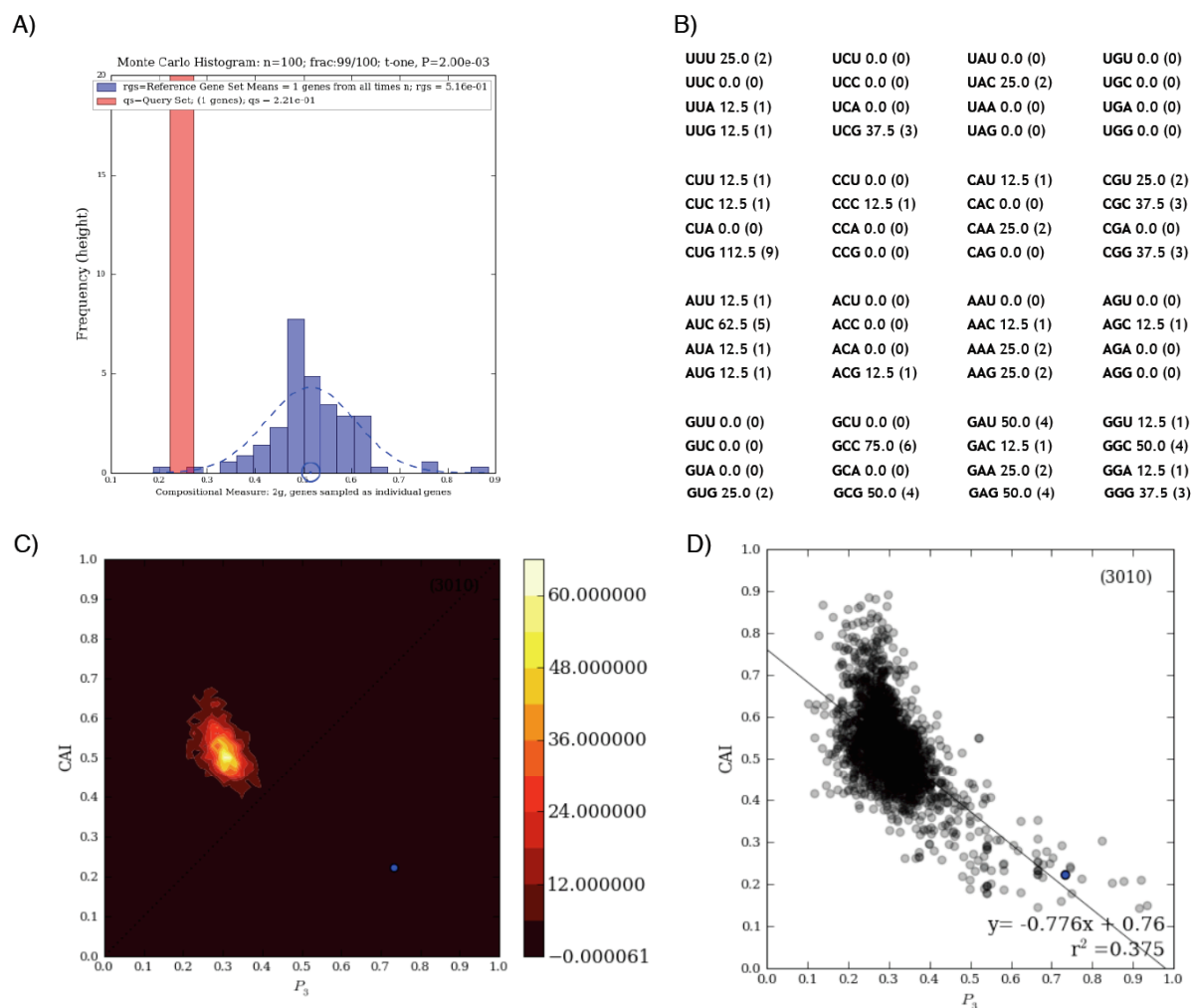


Figure 8.4: Monte Carlo histogram plot (A) and codon usage table (B) for a merR family of transcriptional regulators. Coding GC content: 61.60%; 1st letter GC: 69.62%; 2nd letter GC: 41.77%; 3rd letter GC: 73.42%. (C) Heat map plot of CAI vs P_3 . A merR family of transcriptional regulators shown in blue. (D) Scatter plot of CAI vs P_3 .

resistance [5]. These helix-turn-helix DNA binding proteins respond to various environmental stresses which can range from antibiotics to oxidative stress. Some of the merR genes are also found on transposable elements which could be the underlying reason why this gene shows dramatically different codon usage and CG content.

Chapter 9

Conclusions and Future Directions

In this thesis, I have presented many analyses and algorithms dealing with phylogenetics and horizontal gene transfer. The phylogenetic reconstruction project was a large step in understanding the power of the available tree inference methods. Most importantly the comparison of the methods identified that the software FastTree performed better or equivalently to the other methods. FastTree also has significantly less memory and time requirements than the other two methods. This finding led to building the greengenes and Guerrero Negro phylogenetic tree with FastTree and then the development of an algorithm for decorating this tree with a seven level taxonomy.

It has become increasingly important to gain an understanding of HGT because of its impact on evolution, ecology, and medicine. The results I provided in this thesis suggest a number of avenues for attacking this problem and a path forward for benchmarking and integration efforts. I have shown the importance of using simulated data to effectively benchmark HGT detection methods as well as phylogenetic reconstruction methods. I took part in the implementation of the double birth-death model, which simulates data with known HGT histories. This model allowed me to simulate datasets with varying parameters for the phylogenetic HGT detection comparison, and can be useful in the further analysis

of available compositional HGT detection methods. This general comparison of HGT detection methods will be very important for understanding drug resistance, pathogenicity, and ecological diversity.

Through the research I did for this thesis I gained an understanding of the significant impact computational methods have on answering biological questions. In addition I am walking away with the knowledge that even tasks that seem trivial should be well planned out. For example, I thought it would be simple to use Steele for high throughput tests, but then I discovered the file limit, which forced me to plan out the most efficient way to store files. Another problem I ran into was the naming of the files for the tree reconstruction benchmark. It was important to be able to map the simulated trees with the simulated alignments as well as mapping those to the inferred trees. The file names also needed to contain information about the many parameters that were varied, and stay consistent between all of the datasets. It did not seem like a difficult problem until a little while into the analysis, when I had to run those simulations again simply because of the file names.

In this work I have also presented a new HGT detection method that shows promising results. To further this research, I can compare it with the other phylogenetic and compositional detection methods. Since I have simulated datasets for the phylogenetic methods, these same datasets can be used to test my method and benchmark it against the other methods. My development of a stand-alone version of CodonExplorer gives users more power to analyze their datasets. They no longer need to use only the data found in the database that is hooked up to the web interface and has not been maintained. This implementation also adds to the online version by letting the user specify the titles of their graphs. It also gives

them the ability to visualize certain genes that they specify in a separate color from the rest of the dataset. Overall, the research explained in this thesis has answered many questions involving phylogenetics and horizontal gene transfer, and in doing so has also opened up many additional questions for further study.

Bibliography

- [1] L. Addario-Berry, M. Hallett, and J. Lagergren. Towards identifying lateral gene transfer events. Pac Symp Biocomput, pages 279–90, 2003.
- [2] R. G. Beiko and N. Hamilton. Phylogenetic identification of lateral genetic transfer events. BMC Evol Biol, 6:15, 2006.
- [3] R. G. Beiko, T. J. Harlow, and M. A. Ragan. Highways of gene sharing in prokaryotes. Proc Natl Acad Sci U S A, 102(40):14332–7, 2005.
- [4] D. A. Benson, I. Karsch-Mizrachi, D. J. Lipman, J. Ostell, and E. W. Sayers. Genbank. Nucleic Acids Res, 37(Database issue):D26–31, 2009.
- [5] N. L. Brown, J. V. Stoyanov, S. P. Kidd, and J. L. Hobman. The merr family of transcriptional regulators. FEMS Microbiol Rev, 27(2-3):145–63, 2003.
- [6] J. G. Caporaso, J. Kuczynski, J. Stombaugh, K. Bittinger, F. D. Bushman, E. K. Costello, N. Fierer, A. G. Pena, J. K. Goodrich, J. I. Gordon, G. A. Huttley, S. T. Kelley, D. Knights, J. E. Koenig, R. E. Ley, C. A. Lozupone, D. McDonald, B. D. Muegge, M. Pirrung, J. Reeder, J. R. Sevinsky, P. J. Turnbaugh, W. A. Walters, J. Widmann, T. Yatsunenko, J. Zaneveld, and R. Knight. Qiime allows analysis of high-throughput community sequencing data. Nat Methods. Journal article Nature methods Nat Methods. 2010 Apr 11.
- [7] Y. Cetinkaya, P. Falk, and C. G. Mayhall. Vancomycin-resistant enterococci. Clin Microbiol Rev, 13(4):686–707, 2000.
- [8] Riverbank Computing. <http://www.riverbankcomputing.co.uk/software/pyqt/>, Retrieved: April 10, 2010.
- [9] T. Z. DeSantis, P. Hugenholtz, N. Larsen, M. Rojas, E. L. Brodie, K. Keller, T. Huber, D. Dalevi, P. Hu, and G. L. Andersen. Greengenes, a chimera-checked 16s rRNA gene database and workbench compatible with arb. Appl Environ Microbiol, 72(7):5069–72, 2006.
- [10] Z. Du, F. Lin, and U. W. Roshan. Reconstruction of large phylogenetic trees: a parallel approach. Comput Biol Chem, 29(4):273–80, 2005.

- [11] R. V. Eck and M. O. Dayhoff. Evolution of the structure of ferredoxin based on living relics of primitive amino acid sequences. Science, 152(3720):363–366, 1966.
- [12] R. C. Edgar. Muscle: multiple sequence alignment with high accuracy and high throughput. Nucleic Acids Res, 32(5):1792–7, 2004.
- [13] Eirik Eng. Qt gui toolkit: Porting graphics to multiple platforms using a gui toolkit. Linux J., page 2.
- [14] J. Felsenstein. Evolutionary trees from dna sequences: a maximum likelihood approach. J Mol Evol, 17(6):368–76, 1981.
- [15] F. Ge, L. S. Wang, and J. Kim. The cobweb of life revealed by genome-scale estimates of horizontal gene transfer. PLoS Biol, 3(10):e316, 2005.
- [16] R. Gomez-Gil, M. P. Romero-Gomez, A. Garcia-Arias, M. G. Ubeda, M. S. Busselo, R. Cisterna, A. Gutierrez-Altes, and J. Mingorance. Nosocomial outbreak of linezolid-resistant enterococcus faecalis infection in a tertiary care hospital. Diagn Microbiol Infect Dis, 65(2):175–9, 2009.
- [17] R. Grantham, C. Gautier, M. Gouy, R. Mercier, and A. Pave. Codon catalog usage and the genome hypothesis. Nucleic Acids Res, 8(1):r49–r62, 1980.
- [18] M. Hamady, S. A. Wilson, J. Zaneveld, N. Sueoka, and R. Knight. Codonexplorer: an online tool for analyzing codon usage and sequence composition, scaling from genes to genomes. Bioinformatics, 25(10):1331–2, 2009.
- [19] J. C. Hotopp, M. E. Clark, D. C. Oliveira, J. M. Foster, P. Fischer, M. C. Torres, J. D. Giebel, N. Kumar, N. Ishmael, S. Wang, J. Ingram, R. V. Nene, J. Shepard, J. Tomkins, S. Richards, D. J. Spiro, E. Ghedin, B. E. Slatko, H. Tettelin, and J. H. Werren. Widespread lateral gene transfer from intracellular bacteria to multicellular eukaryotes. Science, 317(5845):1753–6, 2007.
- [20] T. Ikemura and H. Ozeki. Codon usage and transfer rna contents: organism-specific codon-choice patterns in reference to the isoacceptor contents. Cold Spring Harb Symp Quant Biol, 47 Pt 2:1087–97, 1983.
- [21] G. Jin, L. Nakhleh, S. Snir, and T. Tuller. Maximum likelihood of phylogenetic networks. Bioinformatics, 22(21):2604–11, 2006.
- [22] G. Jin, L. Nakhleh, S. Snir, and T. Tuller. Efficient parsimony-based methods for phylogenetic network reconstruction. Bioinformatics, 23(2):e123–8, 2007.
- [23] M. Kanehisa and S. Goto. Kegg: kyoto encyclopedia of genes and genomes. Nucleic Acids Res, 28(1):27–30, 2000.

- [24] S. Karlin, J. Mrazek, and A. M. Campbell. Codon usages in different gene classes of the escherichia coli genome. Mol Microbiol, 29(6):1341–55, 1998.
- [25] R. Knight, P. Maxwell, A. Birmingham, J. Carnes, J. G. Caporaso, B. C. Easton, M. Eaton, M. Hamady, H. Lindsay, Z. Liu, C. Lozupone, D. McDonald, M. Robeson, R. Sammut, S. Smit, M. J. Wakefield, J. Widmann, S. Wikman, S. Wilson, H. Ying, and G. A. Huttley. Pycogent: a toolkit for making sense from sequence. Genome Biol, 8(8):R171, 2007.
- [26] J. G. Lawrence and H. Ochman. Amelioration of bacterial genomes: rates of change and exchange. J Mol Evol, 44(4):383–97, 1997.
- [27] R. E. Ley, J. K. Harris, J. Wilcox, J. R. Spear, S. R. Miller, B. M. Bebout, J. A. Maresca, D. A. Bryant, M. L. Sogin, and N. R. Pace. Unexpected diversity and complexity of the guerrero negro hypersaline microbial mat. Appl Environ Microbiol, 72(5):3685–95, 2006.
- [28] W. Li and A. Godzik. Cd-hit: a fast program for clustering and comparing large sets of protein or nucleotide sequences. Bioinformatics, 22(13):1658–9, 2006.
- [29] W. Ludwig, O. Strunk, R. Westram, L. Richter, H. Meier, Yadhukumar, A. Buchner, T. Lai, S. Steppi, G. Jobb, W. Forster, I. Brettske, S. Gerber, A. W. Ginhardt, O. Gross, S. Grumann, S. Hermann, R. Jost, A. Konig, T. Liss, R. Lussmann, M. May, B. Nonhoff, B. Reichel, R. Strehlow, A. Stamatakis, N. Stuckmann, A. Vilbig, M. Lenke, T. Ludwig, A. Bode, and K. H. Schleifer. Arb: a software environment for sequence data. Nucleic Acids Res, 32(4):1363–71, 2004.
- [30] D. MacLeod, R. L. Charlebois, F. Doolittle, and E. Baptiste. Deduction of probable events of lateral gene transfer through comparison of phylogenetic trees by recursive consolidation and rearrangement. BMC Evol Biol, 5(1):27, 2005.
- [31] M. Margulies, M. Egholm, W. E. Altman, S. Attiya, J. S. Bader, L. A. Bemben, J. Berka, M. S. Braverman, Y. J. Chen, Z. Chen, S. B. Dewell, L. Du, J. M. Fierro, X. V. Gomes, B. C. Godwin, W. He, S. Helgesen, C. H. Ho, G. P. Irzyk, S. C. Jando, M. L. Alenquer, T. P. Jarvie, K. B. Jirage, J. B. Kim, J. R. Knight, J. R. Lanza, J. H. Leamon, S. M. Lefkowitz, M. Lei, J. Li, K. L. Lohman, H. Lu, V. B. Makhijani, K. E. McDade, M. P. McKenna, E. W. Myers, E. Nickerson, J. R. Nobile, R. Plant, B. P. Puc, M. T. Ronan, G. T. Roth, G. J. Sarkis, J. F. Simons, J. W. Simpson, M. Srinivasan, K. R. Tartaro, A. Tomasz, K. A. Vogt, G. A. Volkmer, S. H. Wang, Y. Wang, M. P. Weiner, P. Yu, R. F. Begley, and J. M. Rothberg. Genome sequencing in microfabricated high-density picolitre reactors. Nature, 437(7057):376–80, 2005.

- [32] V. M. Markowitz, N. N. Ivanova, E. Szeto, K. Palaniappan, K. Chu, D. Dalevi, I. M. Chen, Y. Grechkin, I. Dubchak, I. Anderson, A. Lykidis, K. Mavromatis, P. Hugenholtz, and N. C. Kyrpides. *Img/m: a data management and analysis system for metagenomes*. Nucleic Acids Res, 36(Database issue):D534–8, 2008.
- [33] B. E. Murray. The life and times of the enterococcus. Clin Microbiol Rev, 3(1):46–65, 1990.
- [34] B. E. Murray. Vancomycin-resistant enterococcal infections. N Engl J Med, 342(10):710–21, 2000.
- [35] E. P. Nawrocki, D. L. Kolbe, and S. R. Eddy. Infernal 1.0: inference of rna alignments. Bioinformatics, 25(10):1335–7, 2009.
- [36] H. Ochman, J. G. Lawrence, and E. A. Groisman. Lateral gene transfer and the nature of bacterial innovation. Nature, 405(6784):299–304, 2000.
- [37] I. T. Paulsen, L. Banerjee, G. S. Myers, K. E. Nelson, R. Seshadri, T. D. Read, D. E. Fouts, J. A. Eisen, S. R. Gill, J. F. Heidelberg, H. Tettelin, R. J. Dodson, L. Umayam, L. Brinkac, M. Beanan, S. Daugherty, R. T. DeBoy, S. Durkin, J. Kolonay, R. Madupu, W. Nelson, J. Vamathevan, B. Tran, J. Upton, T. Hansen, J. Shetty, H. Khouri, T. Utterback, D. Radune, K. A. Ketchum, B. A. Dougherty, and C. M. Fraser. Role of mobile dna in the evolution of vancomycin-resistant enterococcus faecalis. Science, 299(5615):2071–4, 2003.
- [38] M. N. Price, P. S. Dehal, and A. P. Arkin. Fasttree: computing large minimum evolution trees with profiles instead of a distance matrix. Mol Biol Evol, 26(7):1641–50, 2009.
- [39] M. A. Ragan. On surrogate methods for detecting lateral gene transfer. FEMS Microbiol Lett, 201(2):187–91, 2001.
- [40] M. A. Ragan, T. J. Harlow, and R. G. Beiko. Do different surrogate methods detect lateral genetic transfer events of different relative ages? Trends Microbiol, 14(1):4–8, 2006.
- [41] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. Mol Biol Evol, 4(4):406–25, 1987.
- [42] A. A. Salyers, A. Gupta, and Y. Wang. Human intestinal bacteria as reservoirs for antibiotic resistance genes. Trends Microbiol, 12(9):412–6, 2004.
- [43] P. Shannon, A. Markiel, O. Ozier, N. S. Baliga, J. T. Wang, D. Ramage, N. Amin, B. Schwikowski, and T. Ideker. Cytoscape: a software environment for integrated models of biomolecular interaction networks. Genome Res, 13(11):2498–504, 2003.

- [44] P. M. Sharp and W. H. Li. The codon adaptation index—a measure of directional synonymous codon usage bias, and its potential applications. Nucleic Acids Res, 15(3):1281–95, 1987.
- [45] L. Sheneman, J. Evans, and J. A. Foster. Clearcut: a fast implementation of relaxed neighbor joining. Bioinformatics, 22(22):2823–4, 2006.
- [46] K. V. Singh, G. M. Weinstock, and B. E. Murray. An enterococcus faecalis abc homologue (lsa) is required for the resistance of this species to clindamycin and quinupristin-dalfopristin. Antimicrob Agents Chemother, 46(6):1845–50, 2002.
- [47] A. Stamatakis. Raxml-vi-hpc: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. Bioinformatics, 22(21):2688–90, 2006.
- [48] M. Steel and L. A. Szekely. An improved bound on the maximum agreement subtree problem. Appl Math Lett, 22(11):1778–1780, 2009.
- [49] C. Than, D. Ruths, H. Innan, and L. Nakhleh. Confounding factors in hgt detection: statistical error, coalescent effects, and multiple solutions. J Comput Biol, 14(4):517–35, 2007.
- [50] R. F. Weaver. Molecular Biology. McGraw-Hill, 2008.
- [51] R. A. Welch, V. Burland, 3rd Plunkett, G., P. Redford, P. Roesch, D. Rasko, E. L. Buckles, S. R. Liou, A. Boutin, J. Hackett, D. Stroud, G. F. Mayhew, D. J. Rose, S. Zhou, D. C. Schwartz, N. T. Perna, H. L. Mobley, M. S. Donnenberg, and F. R. Blattner. Extensive mosaic structure revealed by the complete genome sequence of uropathogenic escherichia coli. Proc Natl Acad Sci U S A, 99(26):17020–4, 2002.
- [52] S. Wilson. Alignment and Detection of Syntenic Regions of Genes to Identify Horizontally Transferred Islands in Pathogenic Bacteria. PhD thesis, University of Colorado, 2007.
- [53] Y. Wu. A practical method for exact computation of subtree prune and regraft distance. Bioinformatics, 25(2):190–6, 2009.
- [54] J. Zaneveld, M. Hamady, N. Sueoka, and R. Knight. Codonexplorer: an interactive online database for the analysis of codon usage and sequence composition. Methods Mol Biol, 537:207–32, 2009.