ALGORITHMS FOR SEARCHING AND ANALYZING SETS OF EVOLUTIONARY TREES

A Dissertation

by

GRANT REYNOLDS BRAMMER

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2014

Major Subject: Computer Science and Engineering

ABSTRACT

The evolutionary relationships between organisms are represented as phylogenetic trees. These trees have important implications for understanding biodiversity, tracking disease, and designing medicine. Since the evolutionary process that led to modern biodiversity was not directly recorded, phylogenetic trees are inferred from modern observations. Inferring accurate phylogenies is computationally difficult and many inference algorithms produce multiple phylogenetic trees of equal quality. The common method for presenting a set of trees is to summarize their common features into a single consensus tree. Consensus methods make it easy to tell which features are common to a set of trees, but how do you explore the hypotheses that are not the majority of trees? This question is best answered by a search algorithm.

We present algorithms to query a set of trees based on their internal structure. Trees can be queried based on their bipartitions, quartets, clades, subtrees, or taxa, and we present a new concept which unifies edge based relationships for search functions. To extend the power of our search functions we provide the ability to combine the results of multiple searches using set operations.

We also explore the differences between sets of trees. Clustering algorithms can detect if there are multiple distinct hypotheses within a set of trees. Decision tree depth and distinguishing bipartitions can be used to measure the similarity between sets of trees. For situations where a set of trees is made up of multiple distinct sets, we present p-support which is a measure to quantify the impact of the individual sets on a single consensus tree.

The algorithms are presented within the context of TreeHouse. This is my open source platform for querying and analyzing sets of trees. One goal of TreeHouse was to unite query and analysis algorithms under a single user interface. The seamless interaction between fast filtering and analysis algorithms allows users to the explore their data in a way not easily accomplished elsewhere. We believe that the algorithms in this document and in TreeHouse can shed new light on often unexplored territory.

DEDICATION

To Amy for everything

and

to my family and friends for all of their support over the years.

ACKNOWLEDGEMENTS

I would like to take this time to thank my advisor, Dr. Tiffani Williams, for her guidance and support over this handful of years. I don't know where I'd be without your guidance. You've provide support for me as a researcher, a writer, and a person. I would also like to thank my committee members, Dr. Nancy Amato, Dr. Jennifer Welch, and Dr. Bill Murphy, for the all of the feedback input and revisions. Not just on this document, but on the whole process of graduate school.

I've been honored to work with multiple undergraduate researchers in my time at A&M. Arthur Philpott, Matthew Cember, Jarrett Holtz, and Mark Adamo all contributed to what has become my dissertation work. They also contributed to my understanding of myself as a researcher, a leader, a mentor, and a teacher. Some of the best times I've had in graduate school stem from being a mentor to these individuals. I have Dr. Marc Smith to thank for many of those opportunities, and while he allowed me to be a mentor to his students, he was also being a mentor to me.

I'd also like to recognize my lab mates for all the help, the conversations, the friendship, and the research collaborations. Dr. Seung-Jin Sul, Dr. Suzanne Matthews, and the future Dr. Ralph Crosby have played a large role in my time here.

This department has been a second home to me, and it wouldn't be the same without the staff. Kathy Waskom, Tony Okonski, Theresa Roberts, Elena Rodriguez, Tina Broughton, Lindsay Striegler, Bekah Holle, and Adrienne Krenek supported me in my crazy adventures as the president of the department's graduate student counsel. And much to my surprise and gratitude, they continued to support me after I left that role. These folks do not get enough recognition for the work they do.

I'd also like to thank Brittany Duncan for being the best president an ex-president turn VP could ever ask for. I wish you the best of luck in your remaining time here and all your future endeavors.

Thank you all.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

A fundamental question in biology is how organisms are related to one another. For instance, one might wonder about how the big cats of genus *Panthera* are related to each other and other cats. While biology seeks to define the evolutionary relationships between all species, certain organisms receive more attention than others. The big cats are one such group. They are a well known set of species, but they are also facing serious threats in the wild due to loss of habitat [66, 65]. Theses big cats and the hypothesis about how they are related to one another will serve as the primary running example through out this document.

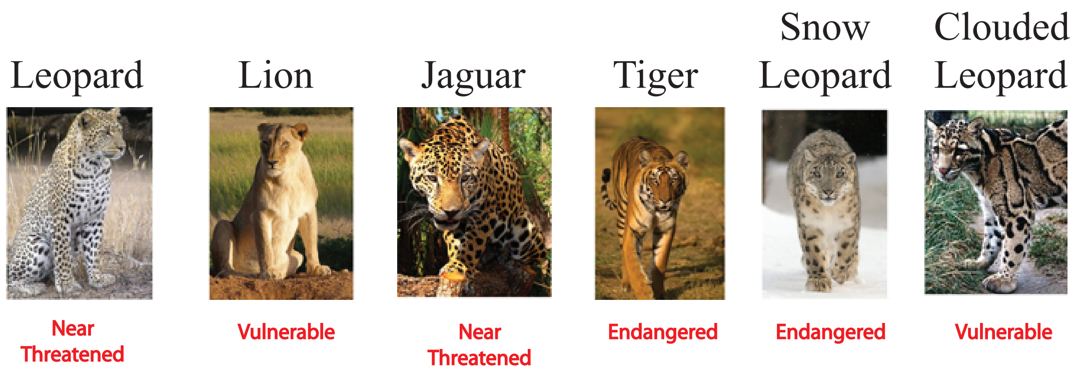


Figure 1.1: Images of the cats in our example data set. The citations for the images used in this figure: Leopard [39], Lion [20], Jaguar [97], Tiger [60], Snow Leopard [45], and Clouded Leopard [98].

The first four cats in Figure 1.1 make up genus *Panthera*, which are the roaring cats. The Snow leopard poses an interesting question because it has the similar throat structure to the roaring cats but it does not roar [28]. So, there is a question of how closely it may be related to genus *Panthera* or if it should be included in genus *Panthera*. The Clouded Leopard serves as a bridge of sorts. It is seen as the link between the big and small cats as it is a large Asian purring cat [18]. The inclusion of a species such as this provides a frame of reference for the questions about relationships between species. A hypothesis of the evolutionary relationships between species is known as a phylogeny.

1.1 What is a phylogeny?

The theory of evolution implies the decent of all living organisms on Earth from a common ancestral gene pool. Evolution is thought of as a branching process where populations are genetically altered over time and may speciate into separate groups, hybridize together, or become extinct. A tree is often used as a metaphor for evolution with evolutionary change flowing from the root to the tips. An early phylogeny from 1866, drawn as a literal tree, is shown in Figure 1.2 [27]. This tree represents one of the first attempts to draw a phylogeny which classifies all known life forms. Modern trees differ from this example in many ways. Much has changed in our understanding of the positions of different organisms and the structure of modern trees tends to be presented in a less literal fashion.

Today's living organisms are just one stage of the evolutionary process resulting from a long series of these speciation, hybridization and extinction events. A tree used to represent the evolutionary relationships between organisms is known as a phylogenetic tree or phylogeny. The leaves of these trees are living organisms and the internal nodes represent the most common ancestor of the nodes that branch from it. Phylogenies have been a part of evolutionary biology since the time of Darwin and provide biologists with a way to structure and visualize their knowledge of biodiversity [105].

The current best hypothesis of the relationships between the six cats in Figure 1.1

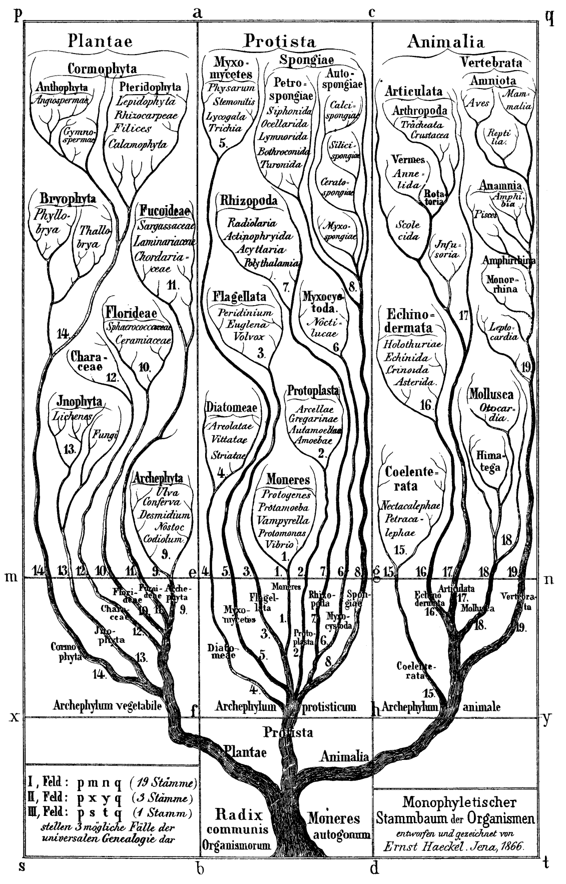

Plantae
Protista
Animalia
Radix communis Organismorum
Moneres autogonum
Monophyletischer Stammbaum der Organismen
entworfen und gezeichnet von Ernst Haeckel. Jena, 1866.
Archephylum vegetabile
Archephylum protisticum
Archephylum animale

Figure 1.2: An early phylogeny drawn by E. Haeckel which was originally published in 1866 [27]. This figure shows the phylogeny as a literal tree.

comes from the work of Davis, Li, and Murphy which was published in 2010 [16]. The phylogenetic relationships reported by the authors are shown in Figure 1.3. A phylogeny can be read similarly to a family tree in that when few branches separate two organisms they are considered to be closely related. One interesting thing to note about this phylogeny is that it places the snow leopard as the closest relative to the tiger. This puts the tiger which is a roaring cat more closely related to a non-roaring cat than the other species that make up genus *Panthera*.

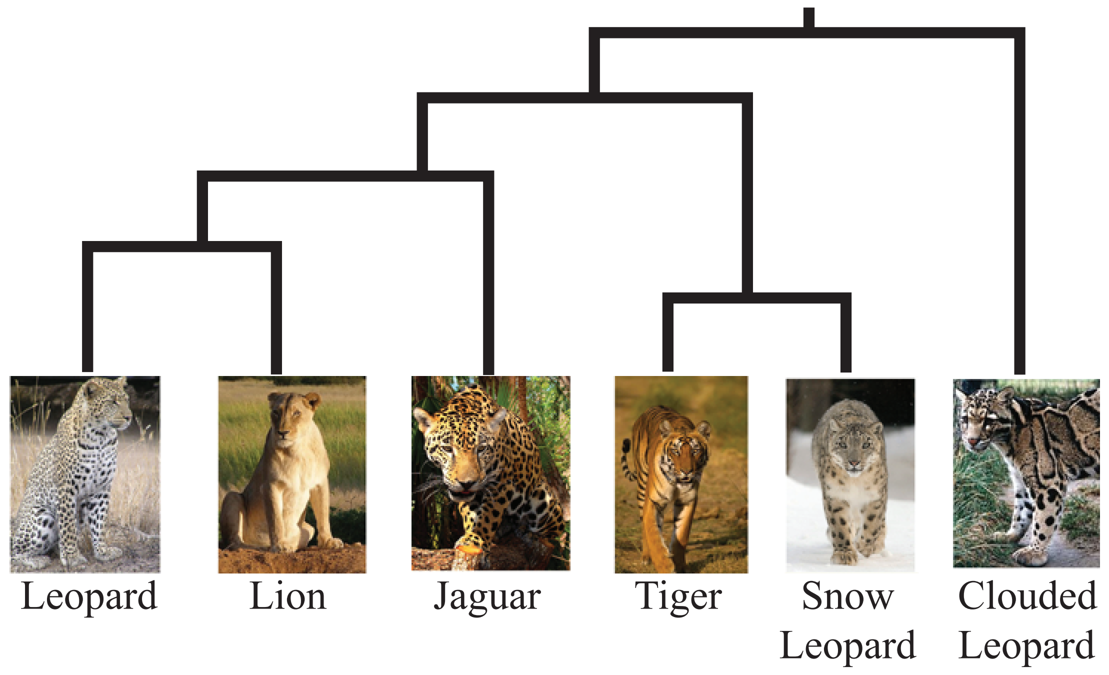


Figure 1.3: The current best hypothesis of the evolutionary relationships between the taxa in genus *Panthera* [16].

1.2 Why do we care about phylogenies?

In many ways a phylogeny is like a map that serves to help organize our knowledge of biological diversity. By placing organisms in the appropriate phylogenetic context we can obtain a better understanding of the patterns and processes of evolution. Phylogenetics has become an important component in modern taxonomy, which is a discipline concerned with the identification, classification, and naming of organisms. Classifying organisms is an important part understanding biodiversity in any given area. Informed decisions about conservation efforts require accurate classifications and estimates of biodiversity, making phylogenetics an important component in conservation efforts [62, 36]. Phylogenetics can also play an important role in other scientific efforts. Phylogenies are used in tracking the spread and evolution of diseases such as HIV [1], hepatitis C [32], and SARS [47, 49]. Phylogenetics has even played an evidentiary role in court cases related to the purposeful transmission of HIV [81, 58, 7, 69]. Phylogenetics is also used in designing chemicals such as medicine [79, 95, 11] and even weed killer [53].

The impact of phylogenetics even extends beyond biology. Stemmatological methods, which are deeply inspired by phylogenetic methods, are used to trace the origins of medieval texts [89, 86, 2]. In many ways the importance of a phylogeny comes from the context that it can provide. Placing an unknown organism into a phylogenetic context provides insight into the evolutionary history of that organism and how its history relates to those of other organisms.

1.3 Why do we care about sets of phylogenies?

Inferring phylogenetic relationships is a difficult process. Evolution can be observed in limited cases such as small scale experiments with single cell organisms or viruses. The process of evolution that led to the differentiation of the species we know today was a long and slow process that took place over millions of years of unrecorded history. Since the exact process of speciation, hybridization, and extinction that brought us modern biological diversity is unknown, phylogenetic trees must be inferred from the evidence of evolution that we can observe. This makes the process of inferring a phylogenetic tree one that seeks to match the observations about a set of organisms to a model of evolution. Finding the tree that best fits the a model of evolution for a set of organisms is an NP-Hard optimization problem. For n organisms of interest there are $(2n-3)!!$ possible rooted binary trees which could describe their relationships [22]. This means that there are almost three times as many ways to arrange 52 taxa into a binary tree than the 1×10^{80} atoms in the visible universe. Many phylogenetic inference algorithms return more than a single tree. Bayesian inference algorithms are known to often return tens of thousands of trees [35]. The common features of these trees are often summarized and presented as a single tree in a publication. The single published tree represents the evolutionary hypothesis for how the organisms of interest are related to one another. The trees that form the basis for this summary are important because they are part of the supporting evidence for published hypothesis.

Sets of trees can also be created as a part of the validation process for a phylogenetic hypothesis. One of the most common support measures for phylogenetic trees is bootstrap support [21]. This method constructs multiple phylogenetic trees with each tree being built from a subset of the observations about the organisms. The frequency of features appearing in the bootstrap support trees is used as a measure of stability of those features in relation to the observations. These trees are treated directly as support for the relationships in the proposed phylogenetic hypothesis. It is common to find the hypotheses, the single published trees, in journal publications and to some degree in online data repositories. The phylogenies that are used to support these hypotheses are harder to find and less explored. The ability to understand and explore the evidence which supports a hypothesis is an important part of any scientific research. The ability for others to understand and explore the evidence is an important part of reproducibility in science.

The work that led to the Pantheran phylogeny in Figure 1.3 included exploring all possible topological arrangements of the organisms. Multiple inference methods were used in order to corroborate the results between multiple algorithms. Once the most likely phylogeny was found, thousands of support trees were built to determine confidence values for the solution. This single tree is the product of a very thorough analysis which included the creation of thousands of trees [16]. Yet, while this tree represents the thousands of trees it is only a slice of the picture in the long history of genus *Panthera*. Other researchers have also looked at the same organisms and come to different conclusions. Figure 1.4 shows fourteen hypotheses and the years they were published. Many of these hypotheses came to very different conclusions about the relationships between the species. They used different data and different methods. There are likely many trees behind these hypothesis in a similar way that multiple trees were built to support the current hypothesis.

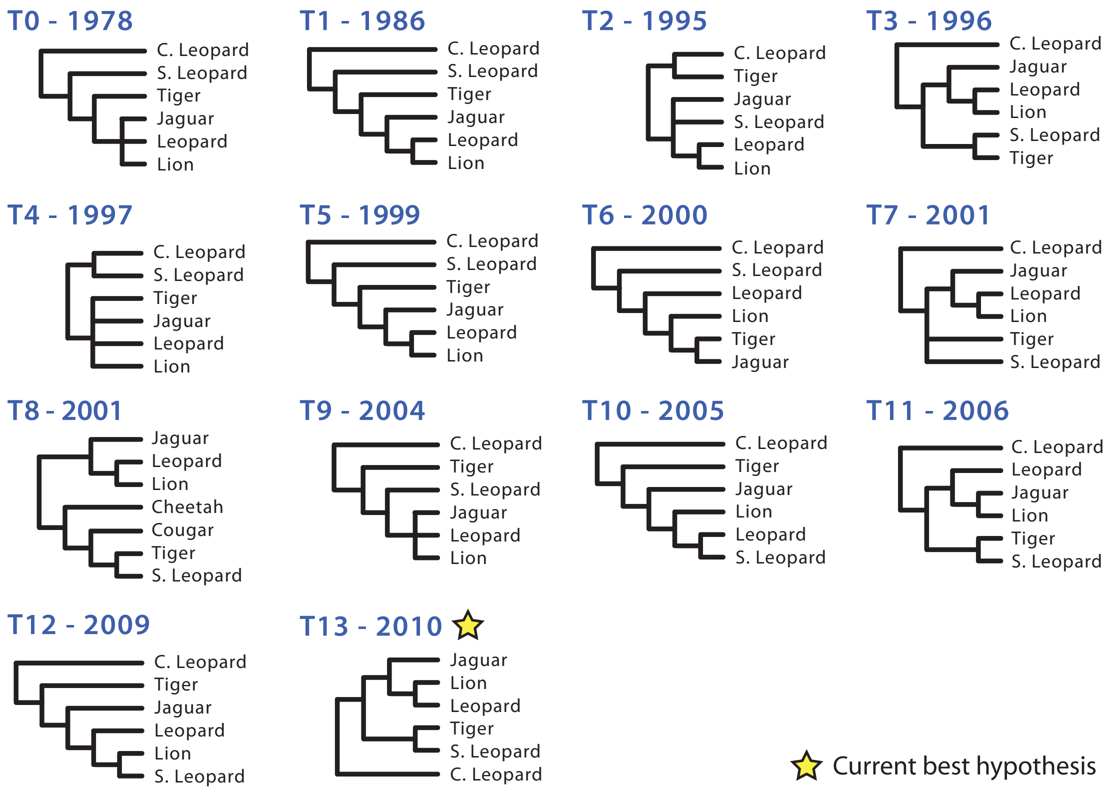


Figure 1.4: Multiple hypothesis of genus *Panthera* over the last 30 years with publication dates shown. T_0 and T_1 are based on morphological data while the rest of the trees are based on biochemical or molecular data. Citations for the trees are as follows: T_0 [30], T_1 [29, 80], T_2 [38], T_3 [41], T_4 [40], T_5 [6], T_6 [54], T_7 [37], T_8 [5], T_9 [73], T_10 [106], T_11 [42], T_12 [104], and T_13 [16].

So, while there may be a set of phylogenies from a single analysis, sets of trees may also be built from the phylogenies resulting from multiple studies. These trees may have been constructed by different algorithms and may contain different taxa from one another. Trees might come from multiple analyses conducted in a single lab or may represent the aggregated work from multiple labs. These trees may represent multiple distinct hypotheses as to the true evolutionary history of the organisms they contain. Trees like this are important to understand both for the phylogenetic information they contain, but they are also important because they represent our changing understanding of the evolutionary history for the set of organisms.

1.4 Research objectives and contributions

The overarching goal of this research is to help biologists build more accurate phylogenetic trees. The accuracy of a phylogenetic analysis is directly tied to the data and algorithms used to infer the phylogeny. Many published phylogenies are compiled from, or supported by, sets of trees. Understanding those trees and what they say about the analysis is an important component to having confidence in the final result. Some sets of trees may completely support a hypothesis while others may have inconsistencies or unexpected patterns. Sets of trees may have multiple distinct hypotheses. These hypotheses can go unnoticed when they are left unexplored and presented as a single consensus hypothesis.

Algorithms for analyzing sets of trees have been limited and lacking. Fundamental questions, such as, "Do any of the trees in this set have a certain feature of interest?" have been difficult to answer. Consensus algorithms allow for the quick identification of the most prevalent bipartitions in a set of trees. However, consensus algorithms are not useful in many cases such as when the bipartition of interest does not appear in the majority of trees or when the feature of interest is not a bipartition. In these cases search algorithms are required. To address the need for search and analysis algorithms for sets of trees we have introduced TreeHouse. TreeHouse is an open source project which aims to provide biologists with the algorithms needed to investigate large sets of trees. TreeHouse allows its users to search, filter, analyze and explore sets of trees with in a single framework. The prevailing idea of TreeHouse is to provide provide users with fast tools which can be quickly combined. When tools like these are at the user's fingertips, interaction and engagement with the data is encouraged.

The rest of this document is laid out in the following sections. Background for phylogenetics is presented in Section 2. Different types of phylogenetic trees and their features are discussed. Also, different ways to define the relationships between taxa such as bipartitions, quartets, subtrees, and clades are defined.

Work related to TreeHouse is presented in Section 3. TreeHouse is not the first platform to provide a solution for querying sets of trees. However, much of the previous work in this area has been focused on a different search question. Many projects have focused on returning the set of trees which contain the taxa which are of interest to the user. Fewer projects have provided a solution which allows the user to query a set of trees based on their structure. The few projects that have addressed this question have done so with narrow scope in terms of ways of defining tree structure.

Section 4 of this document describes the major components of the TreeHouse system. The grammar, lexer and parser for the system are described. These pieces define the interpreted language which allows users to interact with the system via the command line or by running batch scripts. Also described in this Section is the method for storing trees within the TreeHouse system.

Search algorithms are described in Section 5. These algorithms are intended to provide biologists with the answer to the question, ”which, if any, of the trees in the set have a feature of interest?” These algorithms allow sets of trees to be queried based on which bipartitions, quartets, subtrees, or clades they contain. The different ways that a user can define the relationships between taxa in a phylogenetic tree are described in Section 2. This Section also shows how K-tets can be used in queries. The concept of a K-tet is novel to my work, and is first introduced in Section 2. This concept bridges the gap between bipartitions and quartets. Also described is how multiple searches can be combined by using set operations. Combining searches in this way allows for the creation of arbitrarily complex queries. Experiments are presented which show the correlation between complexity and the size of the result set. The experiments show that searches with few taxa can be powerful tools to filter sets of trees. Searches are also shown to be quick on multiple real data sets. This Section also describes the major data structures which provide the foundation for many of the algorithms in TreeHouse. The way that trees are parsed and stored in memory has been designed to make searching for trees based on their structure a fast process.

Section 6 introduces analysis algorithms for single trees and sets of trees. These methods can be used to gather data about a set of trees or to summarize the results of a search. The differences in sets of trees also matter. When trees come from a single analysis, the differences could tell us whether the trees support a single hypothesis for the evolutionary history of a set of organisms or multiple distinct hypotheses. The differences in gene trees can tell us about how different genes evolved within a set of species. The differences in trees inferred from different algorithms or data sets can tell us about how different inference algorithms perform. These differences have been hard to get at. Exploration with search algorithms can help illuminate some of these differences, but when the questions are about larger trends in the data, other algorithms are needed. This Section contains information on consensus and distance methods as well as methods specifically designed for taxa heterogeneous data sets which are data sets where trees may have different taxa from one another. Since many analysis algorithms are not meaningful or well defined for trees with differing taxa, we introduce the concept of taxa masking. Taxa masking allows users to homogenize a set of trees by ignoring any taxa that are not contained in all of the trees in the group and collapsing any extra branches. This method exposes the common structure among the trees and allows the user to apply algorithms, such as consensus algorithms, to sets of trees where they would otherwise be useless.

Sections 7 and 8 build upon the analysis functions in Section 6. These Sections use a combination of concepts from the previous Sections to create more advanced analysis tools. Section 7 covers the use of distinguishing bipartitions and decision trees. These methods provide measures for how similar or different sets of trees are from one another. Section 8 covers clustering methods which can be used to detect distinct subsets within a set of trees. Algorithms for measuring clustering quality are also presented. Clustering algorithms can be useful to detect if there are multiple distinct hypothesis within a set of trees. Experiments are presented which show that multiple runs of a Bayesian inference algorithm, which were previously thought to have converged to a single result, actually represent multiple distinct hypotheses of the true tree. Also presented is p-support which is a support measure which relates the stability of a relationship in a phylogeny to the multiple hypotheses represented in the tree set. This Section also relates the idea of multiple distinct hypotheses within a set of trees to the concept of convergence. Knowing whether or not an analysis reached convergence is important in validating the results of Bayesian analyses.

Two more Sections round out this document Section 9 discusses TreeHouse as platform for reproducible science, and finally, conclusions and a brief discussion of future work is presented in Section 10.

2. BACKGROUND

2.1 Phylogeny as a biological hypothesis

A phylogeny is a way to express how organism are genealogically related. Since the evolutionary events that have led to the diversity of modern species were not directly observed, the relationships between modern species must be inferred. Phylogenies represent hypotheses of the flow of evolution and how organisms are related to one another. A major goal of biology is to reconstruct the tree of life which is the phylogeny depicting the relationships between all of the estimated 5 to 100 million living species.

2.1.1 The structure of a phylogeny

Figure 2.1 shows a phylogenetic tree representing a proposed phylogeny for genus *Panthera* [16]. The major parts of the phylogeny are annotated.

- Leaf Node: The subjects of the phylogenetic study are placed at the leaf nodes of the tree. These often represent taxa such as individuals or populations of organisms. Taxa can also represent larger groupings such as families, genera or orders. Genes or bacterial strains can also be the subjects of phylogenetic studies. While the taxa appearing on at the leaf nodes tend to be currently living, extinct species are sometimes used.

- Internal Node: The internal nodes represent the inferred common ancestor to all of the descendant nodes.

- Clade: A group of taxa including their most recent common ancestor and all of its descendant is referred to as a clade. The internal node marked in the

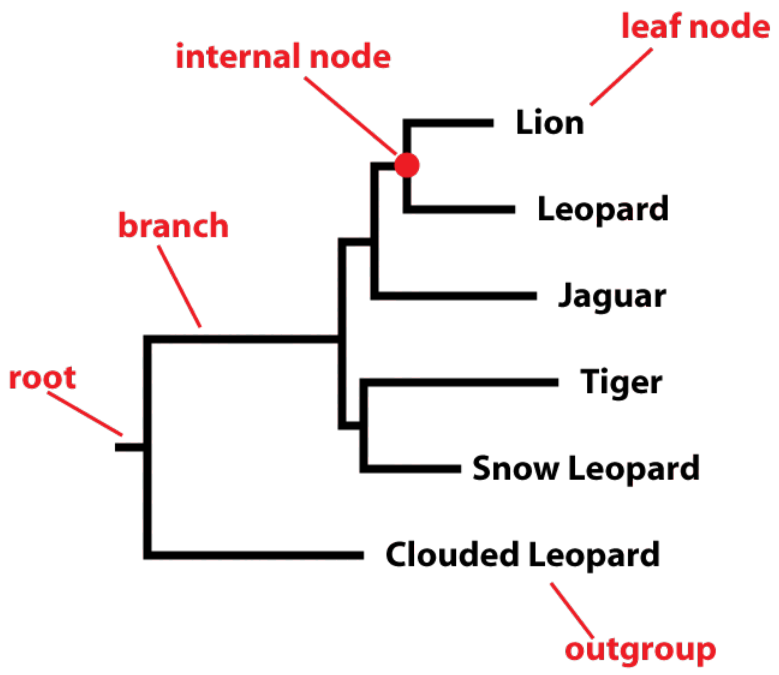


Figure 2.1: The parts of a phylogenetic tree labeled on an example phylogeny for genus *Panthera*.

annotated phylogeny defines a clade which includes lion, leopard, and their most recent common ancestor.

- Root: The root of the tree represents the common ancestor to all of the taxa in the tree.
- Branch: The branches in the phylogeny define the relationships between the taxa.
- Branch length: The length of the branches in a tree can represent the amount of evolutionary change that has occurred over the branch. It can also represent an estimate of the amount to time it took for those changes to occur. A distance scale may be shown with the tree to provide a unit of measurement for how the branch lengths should be interpreted.

- Topology: The topology of the tree is defined by the branching pattern and placement of the taxa.

- Tree shape: The shape of a phylogenetic tree is defined by the branching pattern of the tree without regarding the placement of the taxa.

- Outgroup: To provide a root for a phylogenetic tree, an outgroup is added to the study. This is a taxon or set of taxa which are thought to be outside of the clade that makes up the studied organisms. The subjects of the study can be referred to the ingroup. The tree is rooted where the outgroup meets the ingroup. In the example tree the outgroup is the clouded leopard.

2.1.2 Newick format

The Newick format [23] is the most common format for representing phylogenetic trees as text. It is also the most common format for storing phylogenetic trees on disk. In this format, the topology of the evolutionary tree is represented using a notation based on balanced parentheses. Consider the evolutionary tree in Figure 2.1. A Newick representation of the topology of this tree is "((((Lion, Leopard), Jaguar),(Tiger, S. Leopard)), C. Leopard);." Each set of parenthesis represents a node with the enclosed taxa as children of that node. The ';' character denotes the end of the Newick string. The Newick representation of a tree is not unique. For example, another valid Newick string for the same tree is "(((Tiger, S. Leopard),((Leopard, Lion), Jaguar)), C. Leopard);." In fact, for a tree with n taxa there are 2^{n-1} equivalent Newick strings which represent the tree.

2.1.3 Types of phylogenies

The example tree in Figure 2.1 is a rooted trees with branch lengths. However, a phylogeny can be drawn as either a rooted or unrooted tree, and the lengths of its branches may or may not carry information. In Figure 2.2 we show four possible drawings of the same tree. $Tree$ 1 shows the tree drawn as a rooted cladogram. In this representation the branch lengths are not representative of any data and are often just scaled to be pleasing to the eye. $Tree$ 2 is also rooted but with the branches scaled to represent the amount of evolutionary change down each branch. $Tree$ 3 and $Tree$ 4 are unrooted representations of $Tree$ 1 and $Tree$ 2 respectively. As only the relative positions of taxa and branches matter to the meaning of the tree there are multiple ways to draw the each of the four trees shown. For a tree with n taxa, the number of equivalent visual arrangements of a drawn phylogeny is similar to the 2^{n-1} equivalent Newick representations of the tree.

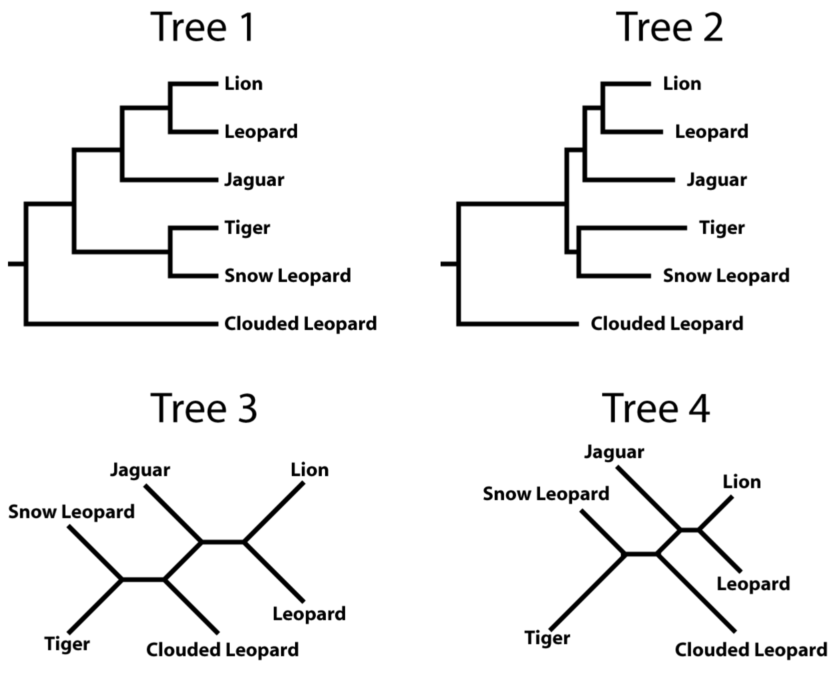


Figure 2.2: 4 ways to draw one phylogenetic tree.

The branches in a tree may contain information beyond the tree's topology.

Branches may carry a range of different values. For instance a tree's branches can be annotated by the support values of that branch, the amount of evolutionary change that occurred down the that branch, or the estimate of the amount of divergence time based on the fossil record. Other trees are depicted with no lengths or support on the branches. When the branch lengths on a tree are based on the amount of evolutionary change the tree is referred to as a phylogram. When the branches are relative to the amount of time since divergence the tree is referred to as a chronogram. When a tree is only representative of the branching pattern of the taxa involved it is referred to as a cladogram. Trees in this document are referred to as either weighted or unweighted trees. Trees that have information attached to their branches are referred to as weighted trees.

2.1.4 Resolution

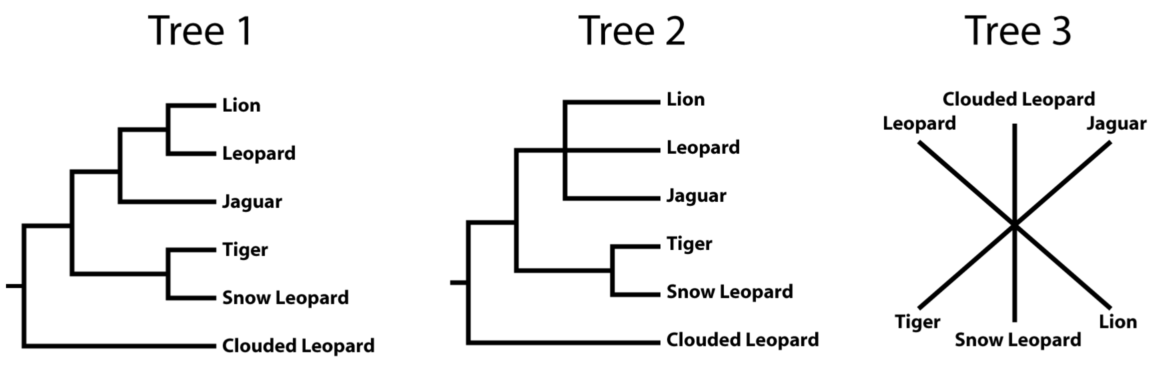


Figure 2.3: Three trees which have the same taxa but have different degrees of resolution.

When the relationships between all the taxa in a tree are resolved, the tree is bifurcating. This means that every internal node has two children. When a relationship is unresolved, it is represented as a polytomy or a node with more than two children. We refer to the percentage of resolved relationships in a tree as that tree's resolution rate. Figure 2.3 shows three trees each containing the same taxa but at different levels of resolution. In this set $Tree$ 1 is fully resolved, $Tree$ 2 is partially resolved, and $Tree$ 3 is completely unresolved. A tree like tree like $Tree$ 3 has a 0% resolution rate has contains no information about the relationships between the taxa. This topology is known to as a star topology.

2.2 Phylogeny as a data structure

When considered as a data structure, phylogenies are interesting due to the fact that the information they contain is stored in the relative position of the labeled leaf nodes. As such, a phylogeny only contains information about how the taxa contained in the tree are related to one another. While it is the relative position of taxa within the topology that is of interest, phylogenies do not enforce order on their leaf nodes. Multiple configurations can represent the same information, such as with Newick strings. It is useful to have a system where each tree has a single unique representation for storing and comparing phylogenies. When dealing with phylogenies where all taxa are uniquely labeled, meaning that a single taxon can not appear in multiple places with in a single tree, we can uniquely represent the phylogeny as the set of relationships between taxa as defined by the tree's clades. MUL-trees may have multiple leaf nodes with the same label. These trees can not be uniquely identified by their set of clades and are not handled by TreeHouse.

2.2.1 Clades

A phylogenetic tree can be uniquely defined by its set of clades. Figure 2.4 shows the four clades and the clade which they define. A clade can be computed at each internal node of a phylogeny. The clade is represented by all of the taxa that descend from the node. Clades provide a convenient representation for phylogenetic trees.

Given a set of clades only one tree topology can be built making the reassembly of a phylogeny possible. A phylogeny can contain at most $n - 2$ clades where n is the number of taxa in the tree. Clades are particularly useful when dealing with sets of trees where different trees may contain different taxa. Since a clade is only defined by the taxa beneath a node, two trees do not have to contain the exact same sets of taxa to have a clade in common. This is different than a bipartition which is defined by the placement of all of the taxa in the tree. Also, since a clade is defined by the taxa below a node in the phylogeny the concept only makes sense for rooted trees. This is not a problem in TreeHouse since the standard format for storing trees, Newick format, is a rooted representation.

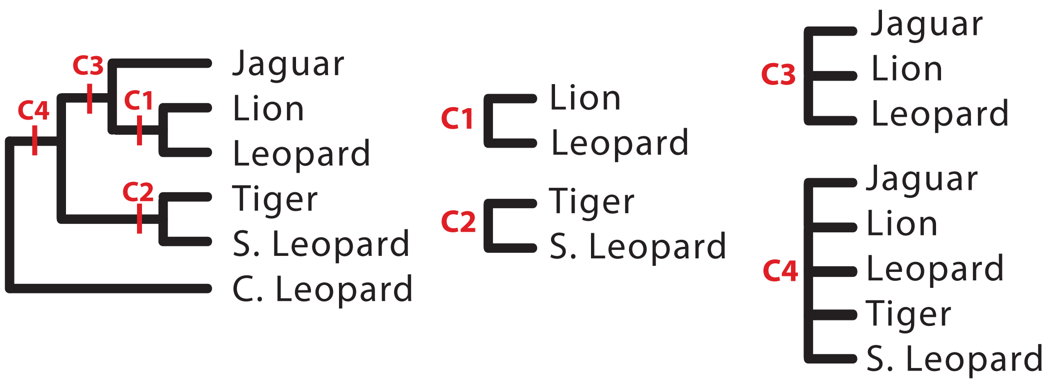


Figure 2.4: A phylogeny and the four clades which define its structure.

2.2.2 Bipartitions

A phylogenetic tree can also be uniquely defined by its set of bipartitions (or edges). When removed, a bipartition splits the taxa into two sets. In Figure 2.5, consider bipartition b_1 in *Tree* 1. It partitions the set of taxa into two groups with taxa A and B on one side and taxa C, D, and E on the other side. We represent

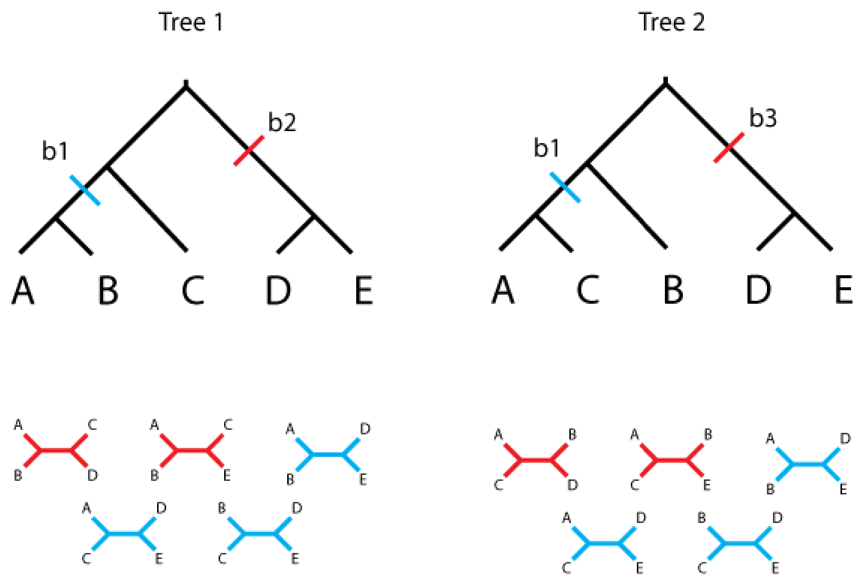


Figure 2.5: Two examples of evolutionary trees depicting the relationships between 5 taxa. Each bipartition is labeled and the quartets making up each tree are shown. Bipartitions and quartets shown in blue are shared between the two trees. Those shown in red are not shared between the two trees.

this bipartition as $AB|CDE$. The taxa in the bipartition are presented using alpha ordering with the side which has the taxa with the smallest alpha numeric value presented first. This is a convention used to prevent confusion. However, the order of the taxa on a side, or the order in which the sides are presented, does not effect the relationship represented by the bipartition. The relationship $AB|CDE$ is also found in $Tree$ 2 and is marked in blue in both trees. The bipartitions b_2 and b_3 are marked in red and are unique to their respective trees. $Tree$ 1 is uniquely defined by the bipartitions b_1 and b_2.

A bipartition generally refers to an internal edge and not those directly linking a leaf node to the tree. While those edges can also be used to partition a tree, they are most often referred to as trivial bipartitions. This is because even the most uninformative star topology, such as the topology of $Tree$ 3 in Figure 2.3, contains n trivial bipartitions. This is not to say these trivial bipartition never hold useful information. In a weighted tree the branch lengths of the trivial bipartitions are important. This means that any algorithm dealing with weighted trees must also handle the trivial bipartitions. A fully bifurcating tree has $n - 3$ non-trivial bipartition which uniquely define the structure of a tree, in addition to its n trivial bipartitions.

2.2.3 Quartets

A phylogenetic tree can also be uniquely defined by its set of quartets [19]. A quartet can be represented as an unrooted 4 taxa tree which defines the relationship between those 4 taxa relative to an edge. If we were to prune all but 4 taxa from a tree, then collapse any uninformative branches, we would be left with a single quartet. A quartet is the smallest tree which contains information about the relatedness of species. For this reason, quartets have garnered interest in the community as they can be thought of as the most basic relational building blocks of a tree.

Figure 2.5 shows the five quartets that make up each tree. We can denote a quartet in a similar manner to bipartitions. For instance, the relationship between the taxa A, B, C, and D is defined as $AB|CD$ in $Tree$ 1 and as $AC|BD$ in $Tree$ 2. For any set of four taxa, A, B, C, and D, there are four possible arrangements. The quartet will either be $AB|CD$, $AC|BD$, $AD|BC$ or $ABCD$, with the last arrangement being the uninformative star topology. While there are $n - 3$ bipartitions in a binary tree, there are $\binom{n}{4}$ quartets. Each tree will have one of the possible arrangements for each of the $\binom{n}{4}$ quartets. In a binary tree, all the quartets will have one of the three informative arrangements as the uninformative arrangement is only possible in multifurcating trees. This means that a fully bifurcating tree contains 1/3 of all possible quartet arrangements for its set of taxa.

Quartets are computationally more challenging than bipartitions, but they have interesting properties. For instance, if we are interested how similar two trees are, we could count the number of bipartitions that they have in common. We would get a number between 0 and $n-3$ for bifurcating trees. If we were to instead count the number of quartets that two trees have in common, our cap would be the $\binom{n}{4}$ total quartets. However, what is more interesting, is that floor for the value is not 0, at least not for trees with more than 6 taxa.

Any two trees with at least 6 taxa in common will share at least one quartet, and trees with more taxa in common will share more quartets. This can be seen by observing the relationship between bipartitions and quartets. We can show that any two bipartitions which split the same group of taxa into two groups of three or more will have at least one quartet in common. Since every tree with 6 taxa will have a bipartition that splits the taxa into two groups of three, all trees with 6 or more taxa will have at least one quartet in common. We will demonstrate this effect with the bipartition $ABC|DEF$.

We can think of all of the other bipartitions which split those 6 taxa into two groups of three as falling into a handful of classes. They could be described as swapping one taxa from each side over the edge, swapping two taxa, or swapping all the taxa. Since the order of taxa in a bipartition does not matter, nor the order in which the two sides are presented, swapping all of the taxa is equivalent to not swapping any taxa at all. This leaves us with the cases where one or two taxa are swapped. In our example, we can swap A for D to get $AEF|DBC$. Again the symmetric nature of the bipartition effects the results, and swapping A for D is equivalent to swapping B and C for E and F. In terms of quartets, the most different a bipartition can be from another bipartition is when half of the taxa are swapped over the edge. However, in cases where there are at least three taxa on each side of the edge, it is impossible not to preserve at least one quartet, no matter the number of swapping operations performed. For instance, in our example both $ABC|DEF$ and $AEF|DBC$ contain the quartet $BC|EF$. This effect has a major impact of the minimum value for how many quartets two trees must have in common. All trees with more than 6 taxa in common have at least one shared quartet, and those with more taxa in common will have more shared quartets.

2.2.4 Unified edge based relationships

Bipartitions and quartets are related. Since every bipartition defines an edge in a tree, we can think of that bipartition as defining some number of quartets. A bipartition of size $L|R$ where L is the number of taxa on one side of the edge and R is the number of taxa on the other defines $\binom{L}{2} \times \binom{R}{2}$ quartets. In this way, we can think of a bipartition as a compressed format for representing quartets. However, while each bipartition may represent multiple quartets, there is no guarantee that a quartet will not be represented in more than one of a tree's bipartitions.

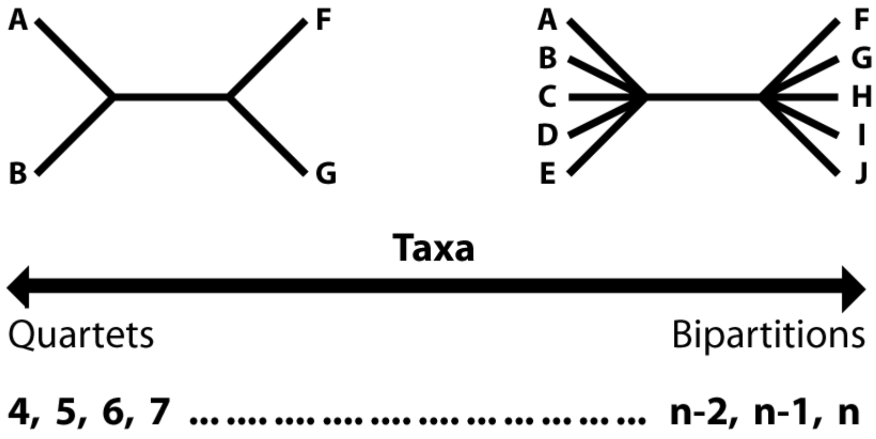


Figure 2.6: Bipartitions and quartets are relationships which define how taxa are placed in a tree relative to an edge. Bipartitions define the positions of all taxa, and quartets define the position of 4 taxa in a tree.

If we think of bipartitions as a collection of quartets over a single edge in a tree, we can draw a bipartition as a subtree much in the same way we graphically represent a quartet. A drawing is shown in Figure 2.6. From this figure, it is easy to imagine that quartets and bipartitions exist at the opposite ends of a spectrum of edge based relationships. A quartet defines the relationship between the minimum number of informative taxa relative to an edge, and a bipartition defines the relationship between all taxa in the tree relative to an edge.

We can fill in the spectrum by considering the relation between $5, 6, 7 ... n-1$ taxa over an edge. These K-tets, or partial bipartitions, may provide us with the tools to look at different aspects of tree structure in new and interesting ways. TreeHouse provides the ability to query sets of trees using any of these edge based relationships.

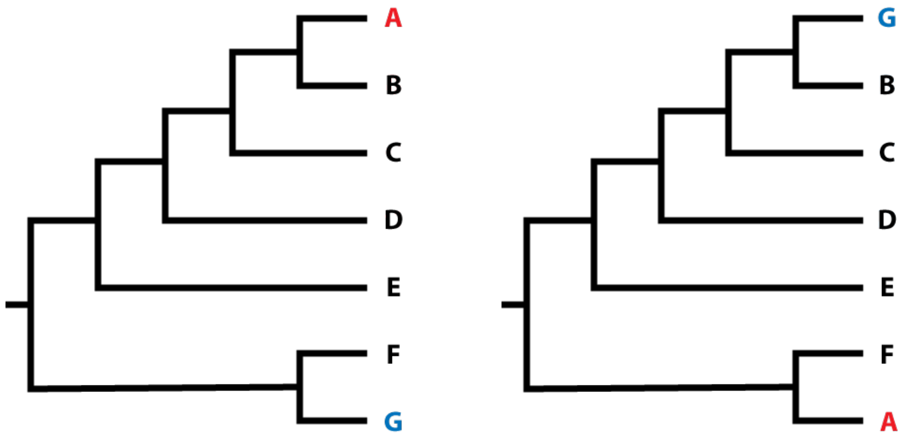


Figure 2.7: Two trees where the second tree is a clone of the first but with Taxon A and Taxon G swapped.

Bipartitions provide a good representation for storing and searching trees. However, when the task is to compare two or more trees, bipartition based measures are often criticized. Figure 2.7 shows two trees. The second tree is a clone of the first with the positions of taxa A and G swapped. These two trees have no bipartitions in common, though much of their structure is similar. The similarity is not reflected when comparing bipartitions because bipartitions define the relationship between all taxa over an edge. The swapped taxa in our example come for opposite ends of the tree, and they are swapped over all the edges in the tree.

Quartets are more robust to detecting the similarity in these types of situations. The trees in Figure 2.7 have five of the 35 possible quartets in common. Computing the similarity of the trees based on quartets allows for the similar structure between groups of four taxa to be measured. That robustness comes at a computational cost as compared to bipartitions. When measuring quartet similarity on two trees with seven taxa we must check 35 relationships, yet there are only four bipartitions to consider.

Relationship	Common	Total
Quartet	5	35
5-tet	2	42
6-tet	0	21
Bipartition	0	4

Table 2.1: The edge based relationships in common between the two trees in Figure 2.7 and the total number of edge based relationships which could be in common for two trees with seven taxa.

K-tets span the gap between bipartitions and quartets. They allow the user to detect different levels of structural similarity as the computational difficulty is varied. This gradation is shown in Table 2.1. The trade off between the sensitivity of a relationship and its complexity is not linear. We can see from the table that the relationship that requires the greatest number of comparisons for a tree with seven taxa is actually the 5-tet. This makes sense when we consider Equation 2.1 which defines r as the number of relationships of size k for a tree with n taxa. The relationship with the maximum complexity, in terms of edges to consider, is related to the number of taxa in the trees. For instance, for 10 taxa trees the K-tet with the most edges to consider is the 6-tet.

$$r = \binom{n}{k} \times (k - 3) \tag{2.1}$$

It is also worth noting that while comparing the trees in Figure 2.7 in terms of 6-tets is more robust than comparing the trees in terms of bipartitions, neither relationship can detect any similarity between the two trees. Since two taxa are out of position in the trees, similarity will only be detected in relationships computed on groups of five or fewer taxa.

2.3 Inferring phylogenies

Creating accurate phylogenetic trees is important but also difficult. This task is often formulated as an NP-hard optimization problem on a solution space with $(2n-3)!!$ possible rooted trees, where n is the number of taxa considered in the analysis. Due to the inherent difficultly of producing an accurate phylogeny, biologists have used heuristic search algorithms to navigate the solution space of possible trees, score them, and return the best results. There are three major methods for evaluating phylogenies.

2.3.1 Maximum parsimony

The simplest method to score tree is maximum parsimony. In this method, the preferred tree is the one that requires the least evolutionary change to explain the sequence alignment. The input data to a maximum parsimony analysis is a set of characters for each taxon. Each character can have one of many states. These characters are often genetic sequences, however a character could be any heritable variation, such as the presence of hair or a tail. Each character can have one of many states, for instance, in sequence data a character could be an A, T, C, or G. Computing the parsimony score of a tree involves inferring the character states of the internal nodes such that they minimize the number of changes of each state in the tree. The parsimony score is the minimum total number of state changes required to explain the relationships between the taxa in the tree. Lower scores are more parsimonious and considered better trees by the phylogenetic inference algorithms. The common software packages [24, 64, 94] for solving the maximum parsimony problem are based on hill-climbing heuristics. These methods use a starting tree which is either random, user provided, or computed via a distance method. The algorithms then rearrange branches to reach neighboring trees. Each neighboring tree is scored. If a rearrangement yields a better scoring tree, it becomes the starting point for another set of rearrangements. The process continues until no better tree can be found. Parsimony analysis scores trees based on the number of character state changes and can often have multiple topologies with the same score. In these cases multiple trees are returned.

2.3.2 Maximum likelihood

Like maximum parsimony, maximum likelihood is an optimality criterion. However, maximum likelihood is a parametric statistical method and employs a model of evolution as part of its scoring function. This method is thought to be more powerful so long as the model of evolution used is a reasonable approximation of how the data was actually created. Since it is difficult to calculate the probability that a given tree is the best tree, maximum likelihood instead computes the probability of the sequence data given a tree and a model of evolution. This is useful since the probability of the sequence data given a tree is the likelihood of the tree given the data. Therefore, the tree with the highest probability of producing the sequence data based on the model of evolution is considered to be the most likely tree. Heuristic algorithms that compute maximum likelihood function much the same way as those which compute maximum parsimony. Both classes of algorithms navigate tree space by rearranging branches on the current tree and scoring its neighbors. However, computing the likelihood of a tree is a much more computationally intensive process, and they require longer run times than parsimony analyses to score the same number of trees.

2.3.3 Bayesian inference

Bayesian inference refers to the use of a prior probability distribution to determine the posterior probability of a particular hypothesis, given the observed evidence. Bayesian inference is similar to the other described methods in how tree space is navigated. It is also deeply tied to maximum likelihood in that both methods require computing the maximum likelihood of the trees visited by the inference algorithm. While a maximum likelihood analysis seeks to return a single most likely tree, a Bayesian inference analysis returns a posterior distribution of trees. The posterior probability distribution is often summarized as a single tree, and the frequencies in which the relationship appeared in the posterior distribution is represented as support values on the trees branches.

2.4 Sets of phylogenies

Maximum likelihood based methods often return a single best tree, but parsimony and Bayesian methods often return sets of trees. Since the scoring function in a parsimony analysis is defined as integer values, it is not uncommon for multiple trees to share the same best score. The goal of Bayesian analysis is to sample the parameter space and return a set of trees. Tens to hundreds of thousands of trees can result from a single Bayesian analysis. Even in the case of maximum likelihood analyses, which return a single most likely tree, support values are often produced by running the analysis multiple times while sampling the input sequence alignment for each run. So, while biologists would like a single hypothesis of the evolutionary relationships between the taxa of interest, they are often presented with sets of trees. These sets of trees are often reconciled and a single tree is published.

Published trees can be deposited into on-line databases such as TreeBASE [61] or Dryad [101]. Theses repositories allow other researchers to access previously published work. The tree sets that result from phylogenetic analysis are fairly different than the tree sets held by these online repositories. All of the trees resulting from an analysis tend to be taxa homogeneous with each tree covering the same set of taxa. This is very different from the trees in TreeBASE which are taxa heterogeneous and two trees from TreeBASE might not have any taxa in common.

Taxa homogeneous and heterogeneous trees pose some similar questions. Researchers may want to know if trees in the set contain a certain relationship, they may want to quantify the difference or similarity between trees, and they may want to summarize trees as a single structure. While there are often similar problems the methods for solving those problems can be very different. For instance researchers wishing to summarize a set of taxa homogeneous trees can use a consensus method where as a super tree method may be more appropriate for a taxa heterogeneous tree set.

2.4.1 Storing sets of phylogenies

TreeZip [55] provides an efficient compression algorithm for storing sets of phylogenetic trees. The method used by TreeZip decomposes and stores trees as a set of bipartitions providing a single representation for a set of trees. This is a significant advance over the Newick format. TreeZip also leverages shared bipartition among trees to reduce the size of the file. The more shared bipartitions in the set of trees, the more benefit from compression. This makes TreeZip an outstanding platform for storing the trees resulting from phylogenetic analyses, as trees produced in this way often have a high degree of bipartition similarity.

With storage comes the need for retrieval. While it is possible to compress and decompress whole sets of trees, the methods we are proposing allow the user to conduct searches on the compressed tree set which return only the trees that match the search criteria. TreeHouse acts as a front end for the TreeZip file allowing the user to search, filter, and analyze their trees from a single user interface.

3. RELATED WORK

3.1 TreeHouse as a platform for querying trees

Current research on searching sets of trees has focused on taxonomically diverse datasets such as the trees stored in TreeBASE[61] and an overview of the field is presented in Table 3.1. The fundamental query over a set such as this is to find the trees which contain certain taxa. This is not a trivial task given that the same organism may appear in the database multiple times under different names [71]. Software packages such as TBMap [70], PhyloFinder [12], and PhyloExplorer [77] have been built to help address this need. Search methods that consider tree structure have also been explored. The previous work on structural searching all follows a similar pattern. Each algorithm computes some measure of similarity between the query tree and the database trees and returns the trees closest to the query subtree based on this measure.

ATreeGrep[82] measures the similarity between the query and database trees and returns the trees within a certain distance threshold. In this method each tree is decomposed into paths such that there exists a path between each node and each other node. Paths are then compared between the query tree and the data base trees. ATreeGrep stores trees as a suffix array of their paths. This can be thought of as a alphabetically sorted array with pointers to each letter. This allows searches for specific paths to be done with a binary search for quicker comparisons. SearchTree[17] builds off of the suffix array and path based distance measure of ATreeGrep but allows users to input the query relationship as a visual dendrogram.

A similar method for searching is used in TreeRank[103]. TreeRank scores the similarity between an input tree and each tree in the database. Similarity between

			Search Functions			
Project	Year	Taxa Name Resolution	Taxa	Subtree	Bipart	Quartet
ATreeGrep[82]	2002			✓		
TreeRank[103]	2003			✓		
TBMap[70]	2007	✓				
PhyloFinder[12]	2008	✓	✓	✓		
PhyQL[33]	2008		✓	✓		
PhyloExplorer[77]	2009	✓	✓			
TreeHouse	2014		✓	✓	✓	✓

Table 3.1: Previous work on querying sets of trees.

two trees is based on the number of topological relationships that the two trees have in common. When a search is performed with TreeRank the query tree is scored against each tree in the database and the best scoring trees are returned in the order of their similarity. Scoring in TreeRank is based around an up down matrix representation of each tree. The matrix represents the number of moves up and down a tree to get from each leaf node to each other leaf node. The distance between two trees is computed based on the difference in their up down matrices. This work has been extended to support weighted branches[102]. ATreeGrep and TreeRank have been implemented as part of TreeBASE.

The search algorithm in PhyQL [33] measures tree similarity based on least common ancestors, and PhyloFinder uses two similarity measures, one based on the Robinson-Foulds (RF) distance [78] and the other based on least common ancestors. In all cases trees that are judged to be close to the input tree are returned. Each of these methods use the subtree model as the basis for inputing query relationships. While we've not been able to test every previous software package due to software being unavailable, those that we've tested are limited in query size. TreeHouse does not limit the size of queries.

Nakhleh et al [63] proposed a database representation of phylogenetic trees which would include the edge relationships between nodes. This idea was implemented as an Oracle database. In these experiments the database queries were issued to find the least common ancestor of two nodes, the minimum spanning clade for a set of nodes, and to find the path length between two nodes. One detractor from their work is the large memory requirement of their database implementation. The way they stored the edge relationships of the trees was significantly larger than just storing the trees in Newick format. In fact, their database representation was up to 7 times larger than storing the trees in a plain Newick file. One benefit of TreeHouse is that it uses the TreeZip[56] representation for storing trees which is often much smaller than storing the same trees in Newick format.

3.2 TreeHouse as a platform for analysis

Most of the work in searching sets of trees has been done from an online database prospective. TreeHouse is not intended to be a database search solution for phylogenetic trees. For TreeHouse, search is intended to be one of the steps in the analysis of the set of trees. Algorithms which allow users to determine the similarity between trees, summarize sets of trees, and detect anomalies in their data sets are key features. In this way, TreeHouse may be most aptly compared with projects like BioPython [13], BioPerl [87], BioRuby [25] and the phylogenetic extensions to R [96, 44, 8]. Each of these projects provide a platform for analysis where data can be manipulated in a variety of ways. These projects, and TreeHouse, allow the user to experiment with and explore their data within a unified framework. There is a community component to the development of these projects. As the code is used and tested it is iterated upon by the community. TreeHouse is still a newer project but contains code contributes from eight individuals including including five different undergraduate researchers, one of whom opted to work with the TreeHouse code base as the foundation for his senior thesis.

BioPython, BioPerl, BioRuby and the phylogenetic packages for R were built as extensions to existing languages. This is a fundamentally different design approach than what was taken with TreeHouse. TreeHouse was built from the ground up with a distinct model of phylogenetic representation in mind. This choice somewhat limits the scope of TreeHouse compared to BioPython. TreeHouse does not handle sequences, alignments, model testing, or tree inference. Instead TreeHouse focuses on problems relating to the structure of trees. This decision allows TreeHouse to use specialized data structures and algorithms optimized for this type of work.

The concepts at the root of TreeHouse build on the work of Seung-Jin Sul and Suzanne Matthews with HashCS [93], HashRF [92], HashRF(p, q) [90], MrsRF [56], and TreeZip [55]. The way that these algorithms parse phylogenetic trees into sets of bipartitions fundamental informs the work in TreeHouse. TreeHouse utilizes the TreeZip format as the input format to load trees into the system. While TreeHouse implements some distance and consensus measures similar to those in Sul and Matthew's work the algorithms in TreeHouse are not meant to compete with those in terms of speed. Distance and consensus algorithms are in TreeHouse to enable further analysis and exploration of phylogenetic data. When speed is the primary consideration, TreeHouse provides mechanisms to call faster algorithms and implementations such as those in Phlash and QuickQuartet.

Due to the broad scope of the analysis algorithms within TreeHouse it would be cumbersome to present all of the related work for each algorithm in this section. Instead, the related work for individual analysis algorithms will be presented as those algorithms are presented.

4. TREEHOUSE: A COMPUTATIONAL PLATFORM

TreeHouse is a computational platform which allows users to interact with sets of phylogenetic trees. The modes of interaction can be grouped into two major categories. The first are mechanisms for querying a set of trees. This is shown at the center of the set of algorithms in Figure 4.1. Query algorithms are useful in their own right as methods for exploring a set of trees, but the results of a query can serve as the input for further analysis. In this way, the query algorithms in TreeHouse can be seen as filters for the data sets. Set operations allow multiple queries to be combined allowing further refinement in selecting trees for further analysis.

Once a set of trees is selected through querying, the trees can be analyzed. TreeHouse supports many methods for analysis including some novel methods. The most common thing to do with a set of trees is to run them through a consensus algorithm. These algorithms take the relationships common to a set of trees and combine them into a single tree. TreeHouse has a few different algorithms for this computation. Users can also compute the similarity or difference between trees using one of TreeHouse's distance measures. Other algorithms are unique to my work. For instance, TreeHouse allows the user to compute the information gain of each relationship for sets of trees. This involves computing the how common a relationship is within and between sets of trees and provides a value for how distinct a relationship is to a set of trees. Each of these analyses can provide the user with information about their data and stand on their own as analysis functions. But, the beauty of TreeHouse lies in combining functions in a way that was not previously possible. These algorithms have become the building blocks for multiple step analyses. Distance functions have been used to implement clustering algorithms and measures of cluster quality. For

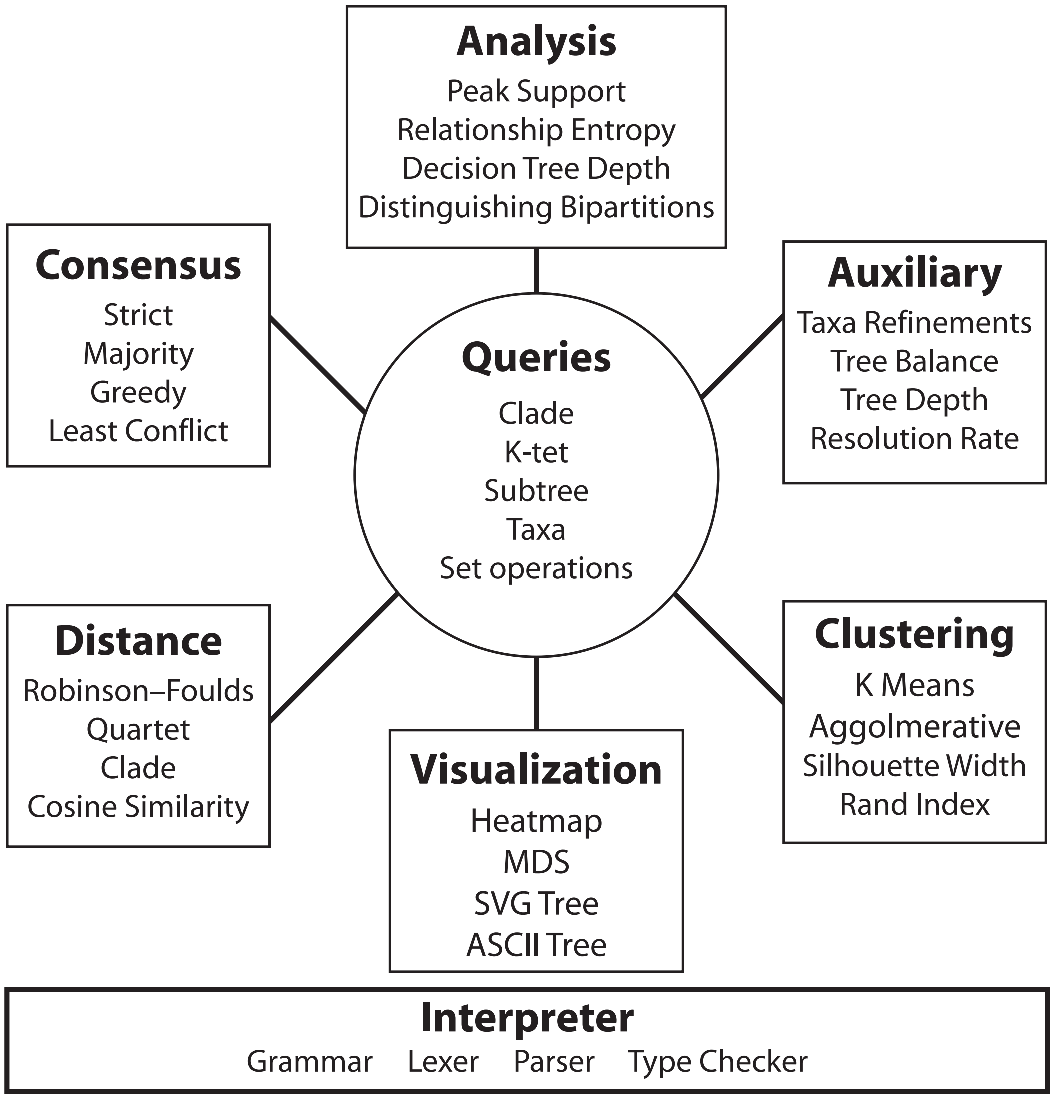


Figure 4.1: A graphical representation of the algorithms within TreeHouse.

instance, the novel support measure p-support is implemented from these building blocks.

This document does not intend to provide an in depth explanation of every algorithm implemented in TreeHouse. TreeHouse currently has over 100 functions. Some, such as consensus, are common to the field. Others such as distinguishing bipartitions, p-support, and the query functions are unique to my work. More time will be spent on these novel functions. Yet, I hope to provide enough breadth that the power of TreeHouse comes through.

One goal of TreeHouse was to unite query and analysis algorithms under a single user interface. The seamless interaction between fast filtering and analysis algorithms allows users to the explore their data in a way not easily accomplished elsewhere. Once the initial algorithms were developed we saw the need to provide an interactive environment as robust as the algorithms it provides access to. Having built the underlying search algorithms in C++ which does not provide an interpreter, we chose to build one. This decision led to the creation of a system which provides users with multiple ways to interact with sets of trees.

4.1 The architecture of TreeHouse

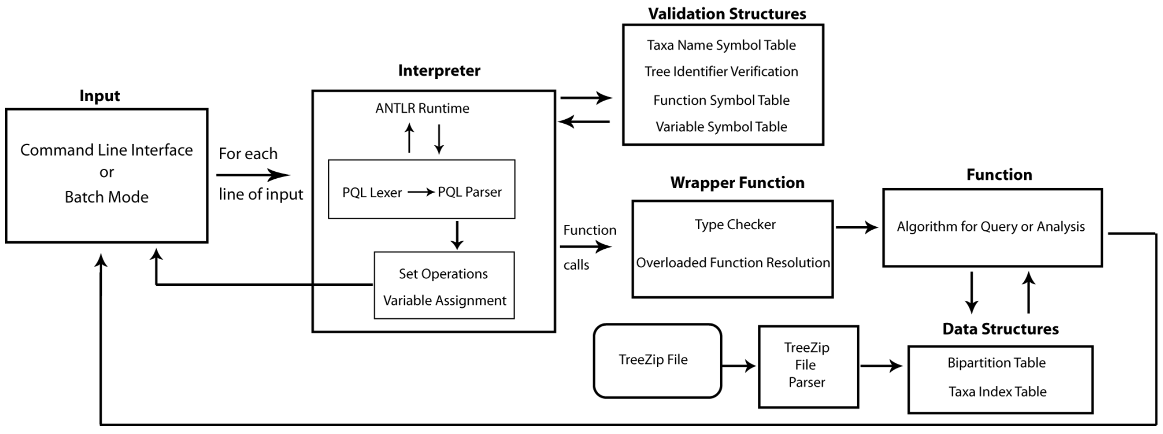


Figure 4.2: A graphical representation of the TreeHouse system architecture.

A graphical layout of the code base for TreeHouse is presented in Figure 4.2. TreeHouse is laid out in a few major sections. The first section to activate when the program is loaded is the TreeZip file parser. When TreeHouse is executed it requires an input TreeZip file which it decompresses, parses, and loads into memory. These are the trees which will be queried or analyzed. Once the trees are loaded, the system is ready to start processing user input commands. Users are able to either execute a batch script with commands in it or interact with the system via a command line interface. Each command is terminated by a new line character and is processed one at a time. The interpreter determines the type of command and passes the input to different functions depending on the content. Some commands, such as the union of two sets of trees, are handled within the parser. Other commands, such as queries over the set of trees, require additional logic. For many commands, error checking also occurs at this part of the process. Function and variable names are checked and resolved against their respective symbol tables. Any command using a set of trees or a taxon identifier goes through extra error and validation as those two data types behave slightly different than the others in the system. When a function is called, the parameters are passed to a function pointer for a wrapper function.

Every user facing function in the system has a wrapper function. This function contains the logic which handles type checking and overloaded function resolution. If the parameters passed to the function are a legal function call, algorithm is run, else an error is returned. Many of the algorithms in TreeHouse interact with a common core set of data structures. These data structures contain the data which is loaded from the TreeZip file and will be discussed at length in the next section. Once the algorithm has completed, its result is returned to the command line, or, when in batch mode, written to a file. The system is then ready to parse and handle the next line of input.

A benefit to this system design is that it allows developers to rapidly prototype and add functions to the system. All a developer has to do is implement their functions in the TreeHouse code base and create a wrapper for any functions that they want to allow users to call. In many ways developing new set of functions in

TreeHouse is similar to defining a library of functions. Since parsing and interaction are handled by the system the developer just has to write their algorithm and provide the interface, or wrapper function in TreeHouse, for how the algorithm should be called.

The TreeHouse interpreter has been developed concurrently with our querying and analysis algorithms. Advances in our algorithmic techniques have led to the implementation of language features which have allowed TreeHouse to transition from a search engine to an analysis platform. In the initial prototype, which was solely a search engine, the only input data type was taxa names and the only output was tree identifiers. TreeHouse quickly outgrew those limitations and required a more robust language for interaction.

4.2 PQL: a language for interacting with phylogenetic trees

At the core of the TreeHouse interpreter is our Phylogenetic Query Language, or PQL, which defines the way a user can interact with the system. Our language is specified by a context-free grammar which is expressed in Extended BackusNaur Form (EBNF). ANTLR [72], an LL(*) parser generator, is used to processes the grammar and produce a C target recognizer for the language. Blocks of C++ code are attached to each element in the grammar and provide instructions for each recognized command. The grammar combined with these instructions allows ANTLR to produce an interpretor for the PQL language. TreeHouse provides the interface to that interpreter through an interactive command line interface and a batch mode.

ANTLR was chosen for its power and flexibility. Its parsers are used by major software products such as Twitter, Hadoop and NetBeans. Its LL(*) method for generating parsers allows for grammars to be defined in a more natural manner than most parser generators while still providing important debugging information which is often missing from GLR and PEG parser generators.

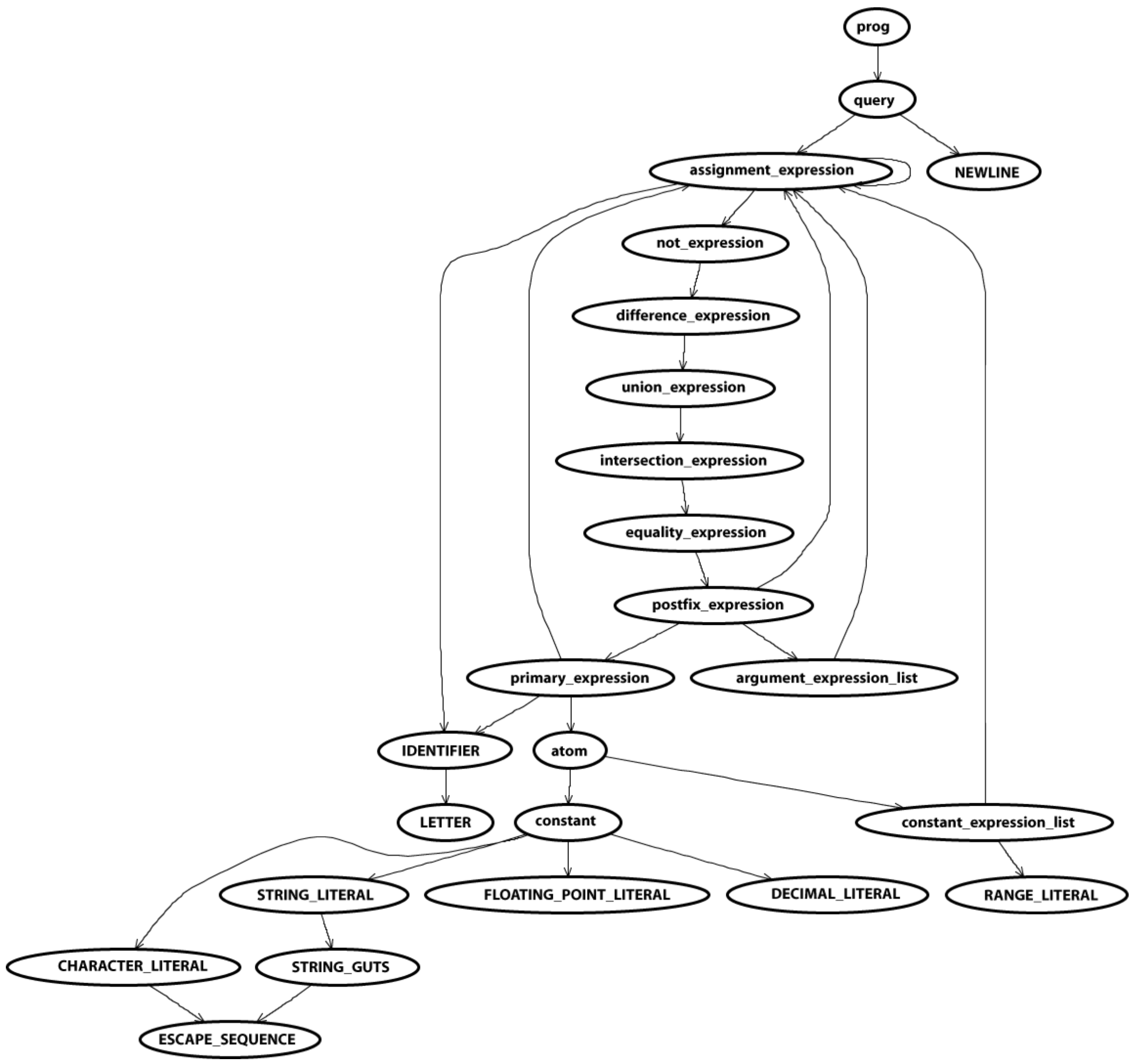


Figure 4.3: A graphical representation of the PQL parser.

The PQL language was modeled after a simplified version of the C language. There are many similar constructs and data types. Figure 4.3 shows a graphical representation of the PQL parser and lexer. The full definition of both the parser and the lexer can be found in Appendix B. The symbols in all capital letters are tokens produced by the lexer, all other symbols represent parser rules. Variables, integers, characters, floats, and strings are supported by the language. To handle groups of data we have indexable lists which act much like the vectors in C++. Lists in PQL can hold any of the primitive data types and are also able to be nested in one another. Lists are restricted such that each item in the list must be of the same type. Lists are defined as a comma separated series of items enclosed in square braces. There are also features to PQL that are different from other C inspired languages and beyond the primary data types which are common to many languages, PQL has two data types specifically set up for handling trees. These are the tree and taxon identifiers.

The trees loaded into the system are represented by their integer value tree identifiers. Each tree is given an identifier based on the order they are loaded into TreeHouse. A set of trees is defined as comma separated series of tree identifiers enclosed in curly braces. Sets of trees are limited to containing only references to trees that are loaded into the system. Users may reference sets of contiguous trees using by using the range operator which is two periods placed between the starting and ending integer values of the range. For instance, 0..100 references the inclusive set of trees from 0 to 100. When a user declares a tree set the values in the tree set are bounds checked. Any negative values and any values greater than the maximum number of trees in the system are removed. When these sets of tree identifiers are passed into TreeHouse functions they are expanded and processed as trees.

Taxa names are handled as a special case of strings. While standard strings are case sensitive and mutable, taxa names are processed as case insensitive and immutable. Each string is case insensitively checked against a symbol table as it is parsed to determine if the string should be treated as a taxon identifier or a regular string. Taxon identifiers can also be linked with other information beyond the taxon's name. Optionally, the user can load additional data into the system. This information could be the full taxonomic classification for a taxon or even information about certain traits the taxon has.

Users are not able to extend PQL by defining new data types at runtime. Extending the language to handle more data types would require modifications to the grammar. Given the limited number of data types in PQL, we've chosen to make the implementation of language dynamically typed. We feel that dynamically typed languages often have a lower barrier to entry for those who are less familiar with computer programming.

Function calls in TreeHouse look much like the function calls in C and other C like languages. The postfix_expression rule in the grammar detects a function identifier and a pair of parentheses which may or may not enclose an argument list. These command names are checked against the TreeHouse function symbol table to identify whether or not the user input a legal function. An example of the parse tree resulting from a simple consensus command is shown in Figure 4.4. The depth and repetitiveness of the tree is due to the fact that as that rules are checked in terms of priority. If a match is found the rule is used. If there is not a match, the input is passed to the next rule until a match is found or the input is deemed not to match any rules.

The fundamental operations when working with sets of trees are set union(+), set intersection($\wedge$), set difference($-$). We have also implemented a not(!) operator which functions as a set difference against the loaded set of trees. Using these operators, users can combine the results of multiple queries to create arbitrarily complex commands. In TreeHouse each of these operators were given their own rule in the grammar in order to enforce the desired order of operations. The order of priority from greatest to least is intersection, union, difference, and then not. The

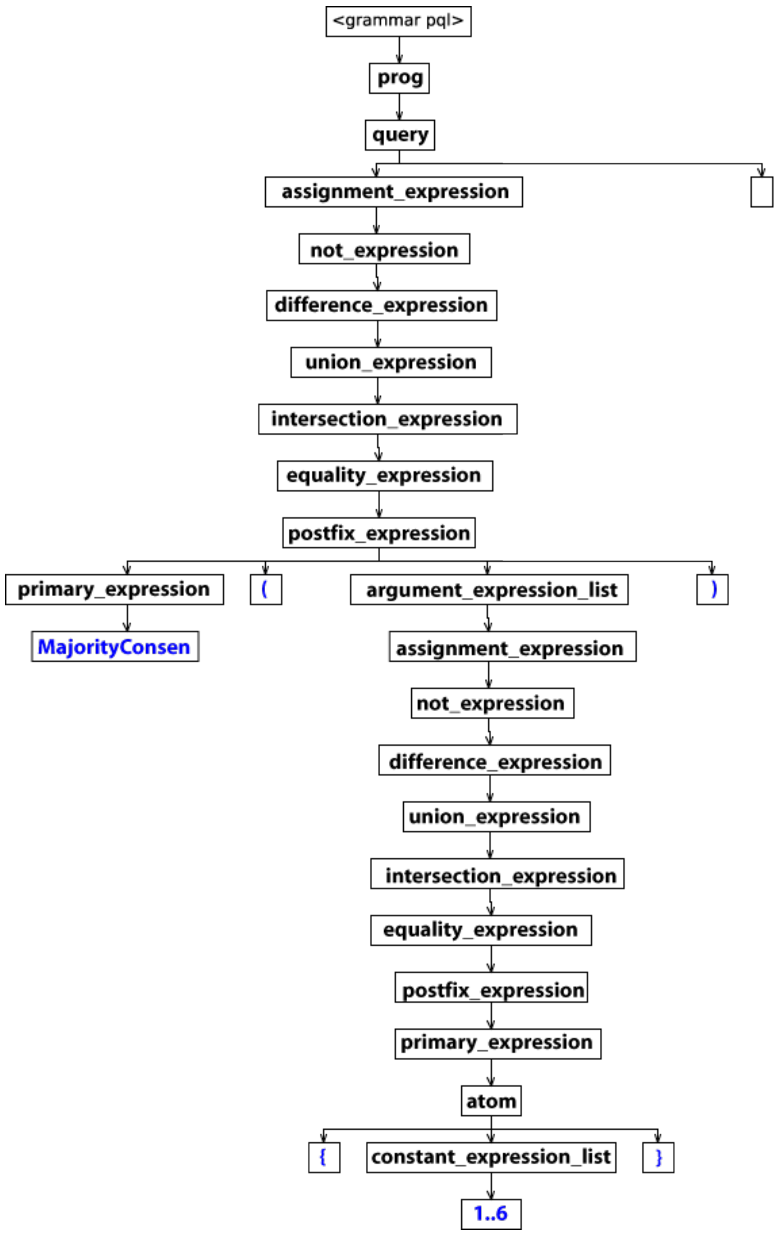


Figure 4.4: A graphical representation of a the parse tree for a single consensus command in TreeHouse.

weakest operator is the assignment operator(=).

PQL is not intended to be a general purpose programming language, it is meant to be a domain specific language for interacting with sets of trees. Since this language is meant to provide a relatively simple interface between a researcher and their phylogenetic data, users are not able to define new functions at run time. This does not mean that TreeHouse is not extensible. New functions may be added to the system, but they are written in C++ and require TreeHouse to be recompiled for them to executed. Care has been taken to separate the functions of the interpreter from the analysis and query algorithms. Users developing for TreeHouse can implement new features without touching the grammar.

4.3 Bit string representation of clades

Sets of trees loaded into TreeHouse are in the TreeZip format. They have been traversed with a depth first search which collects each tree's clades and stores them in a compressed format. The clades that make up Tree T_{12} are shown in Figure 4.5. TreeHouse decompresses the clades and converts them into a special format for storage. Within the TreeHouse data structures each unique clade functions as a key. This requires that each clade has a unique representation within the system. While a user searches for a clade by using a textual representation, TreeHouse stores each clade as a bit string. Each bit string is the length of the number of taxa in the data set with each taxa being assigned an index in the bit string. An example of one such ordering is shown in Table 4.1. Here each taxon in the set is assigned a position in the bit string based on alphanumerical order. To create a bit string representing a clade, the bits corresponding to the taxa in the clade set to a value of one. All other bits in the string are set to zero. The bit strings associated with the four clades shown in Figure 4.5 are given in Table 4.1.

Bit strings encodings are relative to the rooting of the tree. Two trees with the same unrooted topology, and therefore the same bipartitions, could be rooted such that they have different clades. Furthermore a single bipartition could be represented

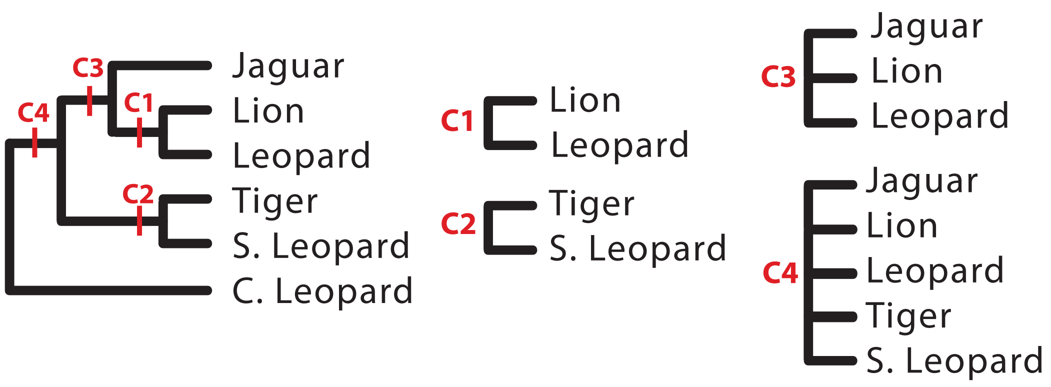


Figure 4.5: This figure shows the four clades that make up Tree T_{12}.

Taxa Name	Cheetah	Cougar	C. Leopard	Jaguar	Leopard	Lion	S. Leopard	Tiger
Index	0	1	2	3	4	5	6	7
C1 Value	0	0	0	0	1	1	0	0
C2 Value	0	0	0	0	0	0	1	1
C3 Value	0	0	0	1	1	1	0	0
C4 Value	0	0	0	1	1	1	1	1

Table 4.1: This table shows the bit string indices for each of the eight taxa in the data set. Also show are the bit string encoding for each of the four clades shown in Figure 4.5

in TreeHouse by two different clades. For instance, Figure 4.6 shows two different trees which each contain the same bipartition. Each tree has the [Lion, Leopard, Jaguar | Tiger, S. Leopard, C. Leopard] bipartition. The nodes defining the clades that represent this bipartition are marked with a red dot in the two trees. While they both have the same bipartition, the clades made up of the descendants of the marked node are inverse of one another. Since two trees with the same bipartition can have different clades it is important to have multiple tools for querying sets of trees. This way edge based queries can be used if the user wants to find a relationship regardless of the root and clades can be used when the position of the root matters to the user.

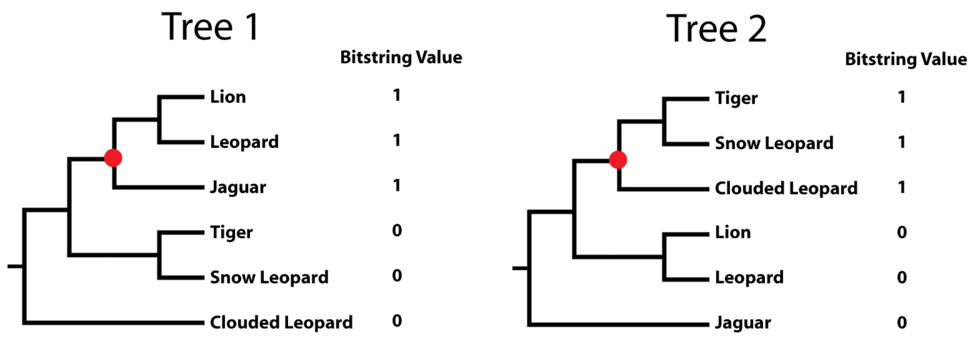


Figure 4.6: Two trees, each containing the relationship [Lion, Leopard, Jaguar | Tiger, S. Leopard, C. Leopard] are shown. The node marked in red is used to compute the encoding for the relationship. The taxa which are descendants of the node are labeled as ones and all other taxa are labeled as zeros.

5. SEARCHING SETS OF TREES

5.1 Motivation

Search algorithms are a fundamental in computer science and have seen some use in phylogenetics. Within phylogenetics, the main focus for search algorithms has been to answer the question, "Which trees contain a certain set of taxa?" This especially important when searching a large taxonomically diverse database such as TreeBASE. What has been less studied is how to answer, "Which trees contain a certain relationship between taxa?" This question is very important when we are dealing with the tens of thousands of trees that have been returned by a Bayesian analysis. In this case, we know that all the trees in the set contain the same set of taxa. What differentiates the trees is their structure. Providing answers to this question can help biologist better understand the data that supports their published phylogenetic trees.

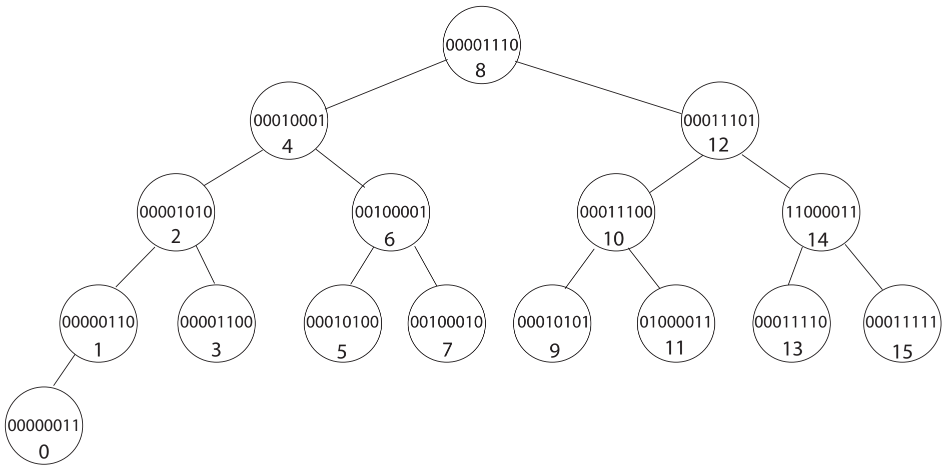


Figure 5.1: A graphical representation of the BST for the *Panthera* data set.

To address the need for structural searching for phylogenetic trees, we have developed a suite of search algorithms which are implemented in TreeHouse. These algorithms provide novel ways to search sets of trees based on their structure. The fundamental search structure in TreeHouse is a binary search tree where the keys are unique clade identifiers and the values are the sets of trees which contain the key clade. This binary search tree is implemented as a red-black tree due to the self balancing characteristics.

Index	Size	Clade	Trees
0	2*	00000011	3, 8, 11, 13
1	2	00000110	11
2	2	00001010	9
3	2	00001100	1, 2, 3, 5, 7, 8, 13
4	2	00010001	6
5	2	00010100	10
6	2	00100001	2
7	2	00100010	4
8	3*	00001110	10, 12
9	3	00010101	6
10	3	00011100	0, 1, 3, 5, 7, 8, 9, 11, 13
11	3	01000011	8
12	4*	00011101	0, 1, 2, 4, 5, 6
13	4	00011110	2, 10, 12
14	4	11000011	8
15	5*	00011111	0, 1, 3, 5, 6, 7, 9, 10, 11, 12, 13

Table 5.1: The in-order traversal of the binary search tree for the example data set represented as a table with the first clade of each size marked.

Along with its use as a search tree the data structure does double duty as an ordered list of clades. To better explain the different algorithms that use the search structure and how they work, two visual representations of the binary search tree are provided in this document. The first as a tree in Figure 5.1 and the second as a table in Table 5.1. This is done solely for clarity of documentation as they are both referring to a single structure in TreeHouse. In Figure 5.1 the clade keys and their position in an in-order traversal are shown. These positions are the same as those shown in Table 5.1. The table also shows the tree identifiers which are associated with each clade. Since the structure is used in these two primary ways it will be referred to as BST+T which is short for “binary search tree and table.”

5.2 Size of the BST+T

The size of the search structure is dictated by three variables of the trees which are loaded into TreeHouse. The length of the keys in the search structure is determined by the number of total taxa t in the set. The number of keys in the search structure is determined by the number of unique clades in the set of trees. The maximum possible number of clades is on the order of 2^t. This has the potential to be a very large number for even a small amount of taxa, but in practice the number of unique clades in a data set is much smaller. There are practical and biological reasons why the number of unique clades do not grow at this rate. Each clade represents a biological relationship between species, and all biological relationships are not equally supported.

To illustrate this point, we can use [Human, Flounder, Parakeet, and Chimpanzee] as the set of organisms for a group of phylogenetic trees. In such such a data set it would be possible for any combination of those organisms to make up a clade, however, if we examine the three possible clades which contain human and one other taxa, [Human, Flounder], [Human, Parakeet], and [Human, Chimpanzee], one of the clades should jump out as far more likely than the other two. This is an extreme case where the [Human, Chimpanzee] relationship was designed to be far more likely than the other options. While, trees could contain the [Human, Flounder] clade instead, the significance of those trees may be called into question.

	Data set 1	Data set 2	Data set 3	Data set 4
Taxa	150	567	950	950
Trees	20,000	33,306	100	200
Unique Clades	1168	2444	5582	9668
Compressed lines	126	522	504	555
Total Tree Ids	2,940,000	18,784,584	70,197	157,437
Stored Tree ids	911,129	3,674,140	38,369	87,055

Table 5.2: Information about the size of the search structure for 4 data sets.

Practically, TreeHouse does not expect to see data where all of the trees in a set agree on every relationship, but random relationships, while providing the theoretical maximum size of for the number of keys, are not a good model for the types of data that biologists work with. Each data set that has been analyzed with TreeHouse has a certain degree of relationship sharing between trees in the set. For instance, Data set #2 from Table 5.2 has 567 taxa and 33,306 trees. The total number of clades possible for a set of taxa this size is about 4.8×10^{170}. This number is far larger than the total number of clades which can be represented by the 33,306 trees in the data set. At most each tree could contain 565 clades assuming each tree was binary which puts the maximum possible number of clades represented at 18,817,890 clades. In order for that to translate into 18,817,890 distinct keys in the data structure each of those clades would have to uniquely appear in a single tree. This would mean that no relationships in the data set would be supported by more than a single tree, which would make it a data set of questionable use. What we actually see when we examine this data set is that there are 2,444 unique entries in the search table. There is a high degree of sharing of clades between the trees, or to put it another way, there are clades in this data set with a high degree of support. Similar patterns appear when looking at other data sets. While we can not provide an exact definition of an upper bound for the number of keys in the system based on the biological and practical considerations, we observed the size of the search structure to be quite reasonable for the data we've analyzed.

The final variable in the size of the search structure is defined by the number of trees in the system. The value linked to each clade in the search structure needs to represent the set of tree identifiers for the trees which contain that clade. This means that for the 33,306 trees in Data Set #2 18,817,890 tree identifiers need to be represented. To limit the amount of memory required for this task some compression is used. For each entry there is a compression bit which denotes if the stored tree identifiers are the tree which have the clade or the trees which do not have the clade. The smaller of the two is used and stored. This means that if every tree has a clade the compression bit would be set to one and an empty set is stored. The effectiveness of this method of compression is dependent on the features of the tree set. The more sharing between trees the more effective the compression is. For a data set which no clade is shared by more than 50% of the trees there would be no compression. The sharing of relationships between trees which allows for a small number of key entries also allows for the this compression scheme to be effective.

Table 5.3 shows how the example data set would be stored. The "Comp" column represents the compression bit and is set to one where compression is used. The "stored size" column shows the set of the set of trees which is stored to represent the trees in the "Trees" column. For this data set compression is only used on for two clades. The clade at index 10 is shared among 9 trees so the inverse can be stored for a space savings of 4 tree identifiers. The clade at index 15 is shared by 11 trees and can be stored with only 3 for a space savings of 8 tree identifiers. Data sets with more sharing between trees will have a higher percentage of compression.

Index	Clade	Comp	Stored size	Trees
0	00000011	0	4	3, 8, 11, 13
1	00000110	0	1	11
2	00001010	0	1	9
3	00001100	0	7	1, 2, 3, 5, 7, 8, 13
4	00010001	0	1	6
5	00010100	0	1	10
6	00100001	0	1	2
7	00100010	0	1	4
8	00001110	0	2	10, 12
9	00010101	0	1	6
10	00011100	1	5	0, 1, 3, 5, 7, 8, 9, 11, 13
11	01000011	0	1	8
12	00011101	0	6	0, 1, 2, 4, 5, 6
13	00011110	0	3	2, 10, 12
14	11000011	0	1	8
15	00011111	1	3	0, 1, 3, 5, 6, 7, 9, 10, 11, 12, 13

Table 5.3: The in-order traversal of the binary search tree for the example data set represented as a table with compression bit and compressed sizes shown.

5.3 Methods for searching

There are two primary methods for searching with the BST+T. The first navigates the BST+T as a BST and the second navigates it as if it was a linked list. Searching for an exact clade is the prototypical example for the first group of queries and the searching for a clade of unknown size serves the most basic example of the second type of query.

5.3.1 Clade queries

TreeHouse supports multiple methods for querying a set of trees. The first method is a clade search. In this case the user is looking for the set of trees which contain a specific clade. This type of search is looking for a single direct match between the query input and a clade in the the BST+T. In TreeHouse, the query taxa are converted to a bit string then that bit string is compared against the binary search tree. If the query value is smaller than the value in the structure the left child is visited, if it is larder the right child is visited, and if the two strings match a result is returned. If a no match is made a childless node an empty set is returned as it is determined that the result is not in the BST+T. This type of search requires $\mathcal{O}(\log UniqueClades)$ comparisons where $UniqueClades$ is the number of keys in search structure. The comparitor function for the BST+Thas been selected such that bit strings with more ones are considered larger than bit strings with less ones. If two bit strings have the same number of ones their relative position is decided by their integer values.

As an example, a user could search for the clade [Lion, Leopard]. Using the bit string indices in Table 4.1 the queried clade would be converted to the bit string [00001100]. Starting at the root, which is marked as index 8 in Figure 5.2, the query bit string is compared the bit string in the structure. The query bit string is smaller than the bit string at the root and so the search proceeds to the left child which is marked with a 4 in the figure. The query bit string is smaller than this clade as well and again proceeds to the left child. The query bit string is larger than the clade and so it proceeds to the right child where a match is found. This process required four comparisons to find the match. Once a match is found the tree identifiers associated with the clade are returned to the user. The tree identifiers are not shown in the figure, but are available in the table at index 3 and consist of trees [1,2,3,5,7,8,13].

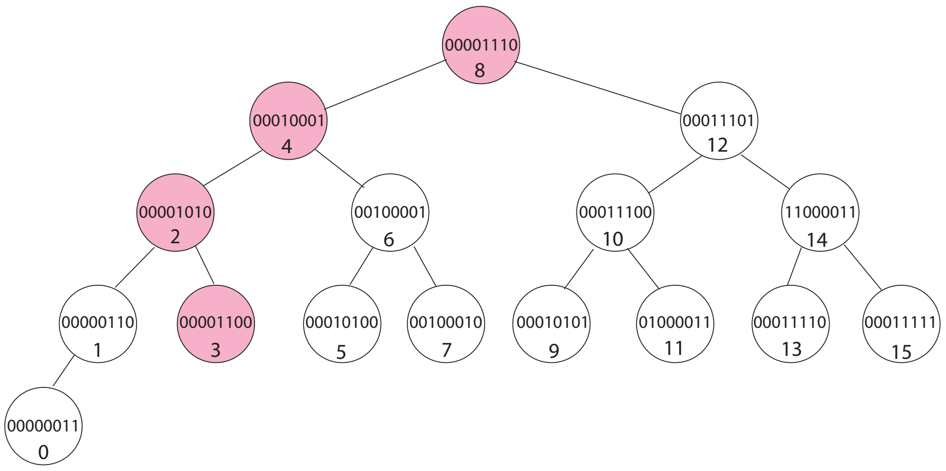


Figure 5.2: A graphical representation of the BST for the *Panthera* data set with the nodes used in a search for the [Lion, Leopard] clade marked in pink.

5.3.2 Subtree queries

By combining multiple clade queries more advanced searches can be built. One such example allows users to directly query a data set for the set of trees which contain a subtree. Since all trees can be defined as their set of clades, TreeHouse can take an query subtree as a Newick string and parse it into clades. Once these query clades are parsed and encoded as bit strings, TreeHouse uses a series of clade search operations to find subtree matches. The algorithm runs an independent clade search for each of the query subtree clades and returns the set intersection of the results of each search. The resulting trees are the ones that contained each of the clades in the query subtree.

The look up time for each individual clade search is on the order of $\mathcal{O}(\log UC)$ time where UC is the number of keys in search structure. The number of searches required to return a solution is equal to the number of clades in the query subtree. The results of of each search must be combined with intersection operations. The intersection requires a number of comparisons relative to the number of tree identifiers resulting from each search. With just two clades the set intersection would require $2 * (treesInResult1 + treesInResult2) - 1$ comparisons to combine the results of each search.

For example, search("(Jaguar, (Leopard, Lion));") is equivalent to the intersection of results from cladeSearch([Jaguar, Lion, Leopard]) and cladeSearch([Leopard, Lion]). The search for the [Jaguar, Lion, Leopard] clade matches with the node marked as 10 which requires three comparisons and returns a result set of [0, 1, 3, 5, 7, 8, 9, 11, 13]. The query for [Leopard, Lion] matches with the node marked 3 and returns the set [1, 2, 3, 5, 7, 8, 13]. The results from each search are combined with a set intersection operation resulting in [1, 3, 5, 7, 8, 13].

5.3.3 Variable size clade query

The next type of search is one where the query could match with multiple clades in the BST+T. For instance a user could search for any clade containing the taxa [Lion, Leopard, Snow Leopard]. This is more general than the previously discussed clade search in that any clade containing the query taxa would be considered a match. Any bit string where the indices relating to the query taxa are set to one is considered a match and the taxa in other positions are ignored. This means that there are 2^{t-qt} possible clades that could be a match to the search, where t is the total number of taxa in the set and qt is the number of taxa in the query. This is another case within TreeHouse where the number of potential clades are often larger than the number of clades which are seen in the data sets. It would be inefficient to run to a separate search for each of the potential bit strings which could provide a match when there are more bit strings than there are entries in the the BST+T. To handle a search where there could be many possible matches the BST+T is navigated with an in-order traversal and each traversed node is checked for a match.

The BST+T contains certain optimization for the queries which rely on an in-order traversal. One such optimization is in the choice of the comparator operation used in building and traversing the BST+T. Bit string identifiers are not primarily sorted by their lexicographical or integer values. Instead, each bit string is first sorted by the number of ones that it contains. This allows the bit strings to be placed in the BST+T such that an in-order traversal prints the clades from the smallest to the largest, with clades integer value deciding the ordering between two clades of the same size. Pointers to the first clade of each size are also attached to structure. These pointers are used to allow the search algorithm to avoid checking parts of the BST+T which could not possibly hold a matching clade.

For instance, a user could search for any clade containing the taxa [Lion, Leopard, Jaguar, Tiger]. A matching clade must contain at least four taxa. Therefore Tree-House uses the pointers to start the search at the first clade of size four, which is at index 12 in Table 5.1. The search proceeds through the rest of the BST+T checking each entry for a match, but the ordering of the clades and the use of pointers has allowed the search to skip 12 of the 16 entries. This search matches with the entries at indices 13, and 15 in the BST+T. The tree identifiers attached to each matching clade are combined using set union operations and returned to the user as the result, which for this search is all of the tree identifiers except the one for tree 8. So the result would look like [0, 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13].

The effectiveness of this range limiting optimization grows with the number of taxa in the input query, but the growth rate is actually dependent on the topologies of the phylogenies in the data set. To illustrate this point, we can consider two simplified data sets each with a single tree. These trees are shown in Figure 5.3. Each tree has eight taxa but have very different tree shapes. The first tree which we will refer to as Tree A, is balanced. Each internal node has the same number of children. The second tree, which we will refer to as Tree B, is maximally unbalanced. They each have the same number of clades, but the sizes of the clades are very different. Notice that tree A has four clades of size two where as Tree B only has one clade of size two. If a user searched for a clade with at least four taxa in it on a data set consisting of Tree A only two clades would be considered. If a user made the same search on a data set consisting only of Tree B, four clades would need to be considered.

For a completely balanced tree like Tree A, the relationship between the number of taxa in the search and the amount of clades which can be skipped is related to the depth of the tree. The exact number can be determined by the function $2^{ceil(log_2(qt))}$. For a completely unbalanced tree like Tree B, the optimization eliminates the need to consider clades linearly with the number of searched taxa and the exact number can be determined by the function $qt-2$. While these are extreme cases, they give an idea of how the distribution of the clade sizes within a data set affect the optimization.

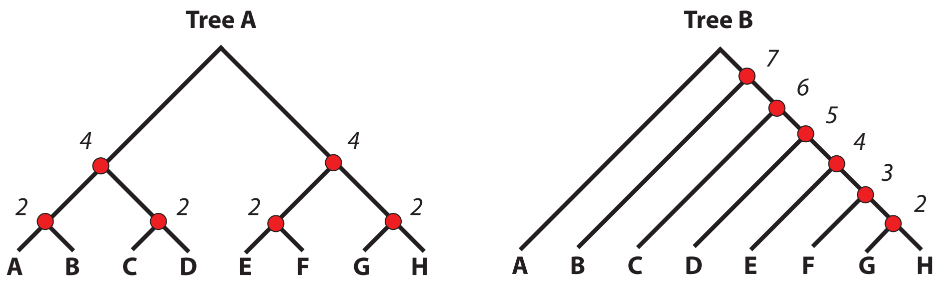


Figure 5.3: An example of how the shape of a tree can impact the sizes of the clades in the phylogeny.

The other major optimization that effects the in-order traversal searches is allowing the search to terminate early if all possible trees are already in the result set. For instance, a query requesting all the trees that contain a clade containing [Lion, Leopard] would start at index 0 and find matches at index 3, 8, 10, and 12. The search would find matches at indices 13 and 15 as well, but after the match at index 12 all possible trees have been added to the set. Early termination in these cases allows the search algorithm to save the work of unneeded comparisons but also the associated with unnecessary matches. Had the search been allowed to continue to find matches 14 tree identifiers would be unnecessarily added to the set.

5.3.4 K-tet queries

A K-tet is a broad way of defining an edge based relationship within phylogenies. Two popular examples of edge based relationships are bipartitions and quartets. A bipartition defines the position of each taxa relative to the an edge and a quartet defines the position of four taxa relative to an edge. TreeHouse provides methods for searching for K-tets of any size. Searching for K-tets is a bit more complicated than searching for a clade and uses a modified version of the logic for an in-order traversal clade search. This is because a K-tet can be represented by multiple clades in the search structure. For a K-tet with qt query taxa there are $2 \times 2^{t-qt}$ possible bit string encodings of clades that could represent the query.

A K-tet query is similar to running two variable size clade searches with one for each set of taxa on either side of the edge. There are a few differences. The first difference is in how a matching clade is determined. To find a clade that matches a K-tet all of the taxa on one side of the edge must be set to ones and all of the taxa on the other side of the edge must be set to zeros. Two bit strings are built, one to represent a clade made of the taxa on each side of the edge. For example, if the query was for the K-tet [Lion, Leopard | C. Leopard, Jaguar], a bit string [00001100] would be built to represent the possible [Lion, Leopard] clade and another bit string [00110000] would be built to represent the possible [C. Leopard, Jaguar] clade. A match is then determined by a series of bitwise operations. Let the first bit string be called BS_1, the second bit string be called BS_2 and the entry in the BST+T which they are compared against be called BS_C.

A match between BS_1 and the BS_C then occurs when $BS_1 \& BS_C == BS_1$ and $BS_2 \& BS_C == 0$. If a match is not found between BS_1 and the BS_C a check it made between BS_2 and the BS_C.

The optimization which uses pointers to reduce the amount of the BST+T which needs to be traversed also functions slightly differently. Where as in the previous example the algorithm was only concerned with a clade having enough ones to provide a match, now a clade needs to have enough zeros to be a match as well. For instance a K-tet query for the relationship [Jaguar, Leopard, Lion | C. Leopard, S. Leopard, Tiger] could only match with entries in the search structure that have at least three ones and three zeros. The requirement on the number of ones allows to search to skip all of the clades of size two, meaning the search would start at index 8. The requirement on the number of zeros allows the search to end earlier and stop after checking index 14.

Another difference between clade queries and K-tet queries is that extra work must be done to ensure the correct results on taxa heterogeneous data sets for K-tet queries. This is because a zero in a bit string just means that the corresponding taxa is not in the clade. That taxa might be on the other side of the edge which defines the clade or it might not be in the tree at all. For instance, a query for the K-tet [Tiger, S. Leopard | Jaguar, C.Leopard] would match with the clade [Tiger, S. Leopard] at index 0. The trees with this clade include tree T_8. However, tree T_8

does not have the C. Leopard at all. To ensure the the results are accurate a flag in TreeHouse is set for heterogeneous data sets. When this flag is on extra work is done before the results are returned. A search is run to find all the trees which contain all of the taxa in the taxa in the query K-tet. The results of this search are combined with the results of the ktet query with a set intersection to remove any false positives that may be present. This is only done for edge based searches on heterogeneous data sets.

5.3.5 Other search functions

TreeHouse also provides functionality to find trees with duplicate topology to an input tree or to filter all the duplicate topologies from a set of trees. There is also functionality to limit the results based on where a tree appears in the input file. This feature is useful in situations where file position has meaning. For instance, the results of a Bayesian search are often written to file in the order in which they are produced. In this case, the feature could be used to exclude or explore the burn-in trees which are the trees produced before the search converged upon the posterior distribution.

TreeHouse also supports similarity based searches. In these searches a distance function, such as the Robinson-Foulds distance, is selected at the trees most similar to the query tree based on the selected distance are returned. These searches can be useful to find which trees are most similar to a hypothesis of interest when none are a direct match.

5.3.6 Combining queries with set operations

Set intersection is used behind the scenes as part of the subtree search functionality but it is also available directly to users to modify the results of their searches. Along with set intersection, we've also implemented set union and set difference to allow users further control over their searches. There is also a “not” operator which computes the set difference against all loaded trees. Using these operators, users can combine multiple searches to create arbitrarily complex queries to fit their needs.

5.4 Comparison to other search structures

The BST+T in TreeHouse was selected as a trade off between running times for queries where the exact clade is known and queries where multiple clades could provide a match. A hashtable would have provided quicker results for searches where an exact match is needed. The hashtable’s $\mathcal{O}(1)$ lookup time trumps the BST’s $\mathcal{O}(\log n)$ lookup time. However, since a hashtable is an unordered data structure the optimization used in the variable size clade searches and K-tet searches to avoid traversing unnecessary portions of the search structure would not have been possible. Using a hashtable would have amplified the differences in the runtimes between the two types of searches in TreeHouse. The exact searches which are already the fastest would become faster and the slower searches which rely on a traversal of the search structure would become even slower.

5.5 Experimental Results

The experiments use real data sets to illustrate the running time of the TreeHouse search algorithms. All experiments were run on commodity hardware with an Intel Core2 quad-core CPU with each core running at 2.50GHz. The machine has 4 gigs of ram and the operating system is Ubuntu Linux 12.04 LTS.

5.5.1 Clade query

In TreeHouse a clade query is handled by navigating the BST+T as a BST. The run time of this type of operation is determined by multiple variables. The first variable is the number of unique clades in the data structure as that determines the depth of the BST+T. The second variable to consider is the number of taxa in the data set as the length of the bit strings have an effect on how long it takes to process a bit string comparison. The third important variable is the number of trees in the data set. This value is only important for the running time of a successful search, but can be very important as it often takes longer to allocate the memory to return the solution than it does to run the search. We evaluate the impact of the number of trees by running two experiments on the same data set. In the first set experiment we compute the average run time for a successful search for different size queries. This data can then be compared to a second set of experiments where the average time for an unsuccessful search is computed over the same query sizes. Since TreeHouse is quite fast the values are computed by running multiple searches sequentially then dividing the total time by the number of searches run.

Clade queries are some of the fastest queries in TreeHouse. This is for two reasons. There is a limited number of comparisons required to get to a match. Since the algorithm traverses the BST+T like a binary search tree the majority of the clades are not required to be tested in any single query. The second reasons is that only a single match with in the structure is possible. This is especially important for data sets with many trees. Successful queries must allocate memory for the results to be returned to the user. The more trees which have to be returned the longer the search will take. The effect of the number of trees can be seen in Figure 5.4. This plot shows the average time it take to compute queries of different sizes on a data set with 567 taxa and 33,306 trees. The first thing to note is that all of the searches no matter if they were successful or unsuccessful and regardless of the size only take a fraction of a second. Unsuccessful and successful searches require the same amount of work up until the point where a match is or is not found. When a match is found, results are returned. The differences in run times shown in the plot are completely due to the time it takes to return the results to the user. This is why the unsuccessful searches are practically flat relative to the number of taxa in the query. The size of the query has a very small impact on the number of comparisons that are done and how long those comparisons take to compute.

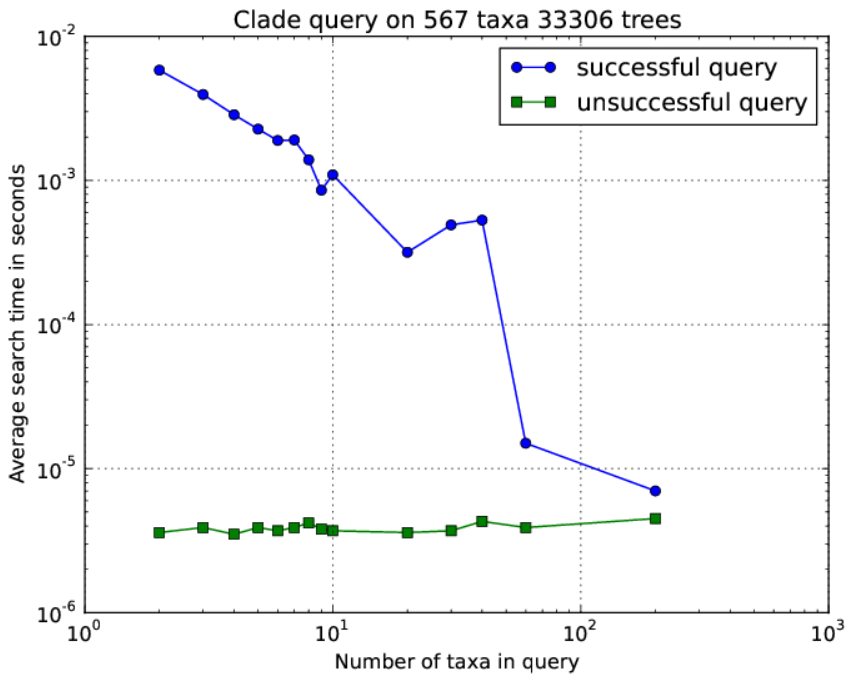


Figure 5.4: Timing results for clade queries on the data set with 567 taxa and 33,306 trees. Successful query times are an average of 10,000 searches of each size. unsuccessful query times are from an average of 100,000 searches of each size.

As the size of the query grows the successful queries are shorter on average. This has everything to do with the distribution of clades among the trees in the data set. As the size of the query grows the average number of trees returned by a successful search goes down and therefore it is quicker to return the results. This can be seen in Figure 5.5 which shows the average number of trees returned by the successful search for the different queries. For a data set like this one, where there are many trees, the total runtime is dominated by the number of trees returned by a search.

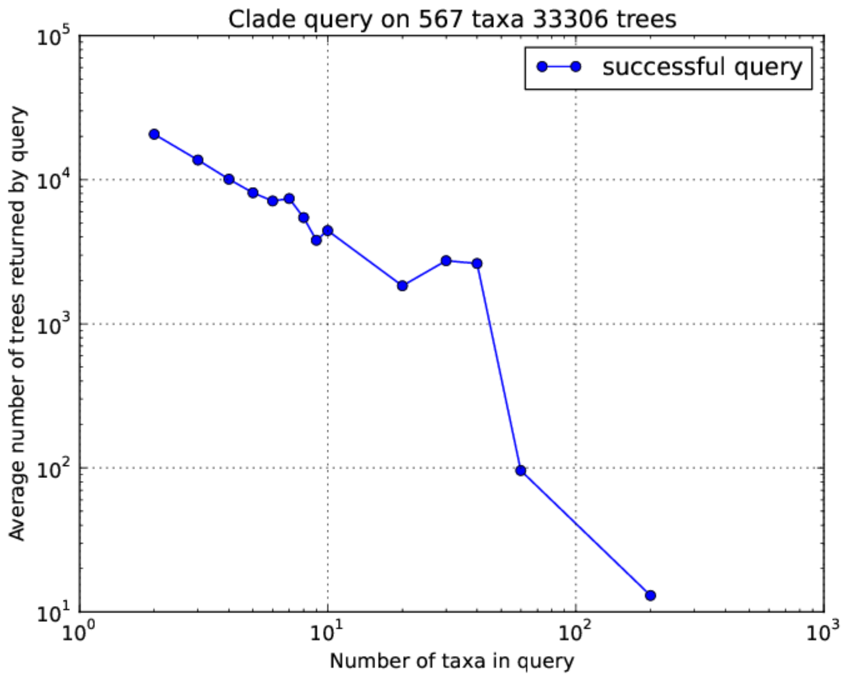


Figure 5.5: The number of trees reported is an average from 10,000 random successful clade searches on the data set with 567 taxa and 33,306 trees.

This effect is less prevalent in data sets with fewer trees. For example, a data set with 950 taxa and only 200 trees is shown in Figure 5.6. For this data set the search times between the successful and unsuccessful searches is much more similar. This is due to the smaller number of trees. In this data set the average search time for the slowest of the queries, a successful query of size two, is only 0.0000115 seconds long.

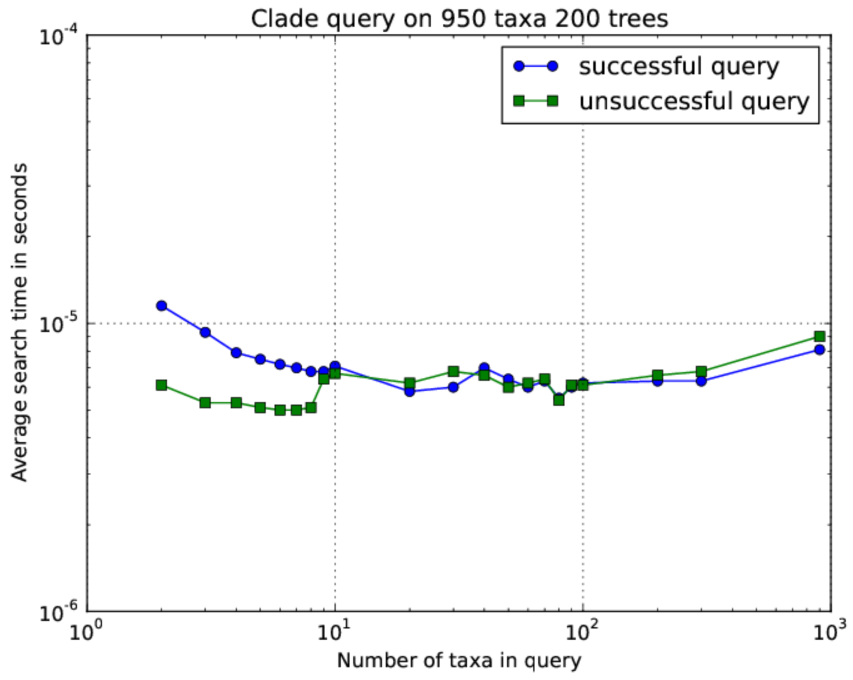


Figure 5.6: Timing results for clade queries on the data set with 950 taxa and 200 trees. Successful and unsuccessful query times are an average of 100,000 searches each for each query size.

5.5.2 Variable size clade query

The variable size clade query navigates the search structure using an in-order traversal. This method also utilizes a few optimizations to minimize the run time. The first optimizations is to use pointers in the table to skip entries in which are too small to be a match. The second optimization allows the search to terminate early when once all possible trees are found. To measure the effect of these optimizations experiments were run with each optimization turned on and off.

Figure 5.7 shows two sets of experiments on Data Set #1. Searches of various sizes were run and timed. The values along the x-axis show the number of taxa in the query and the y-axis shows the average running time of a successful query of

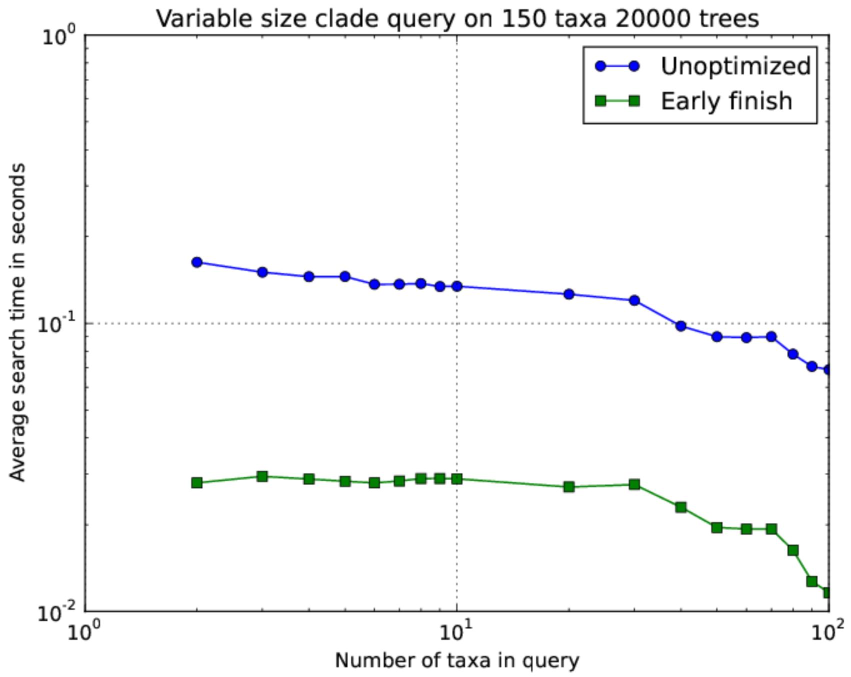


Figure 5.7: Unoptimized and optimized timing results for variable clade queries on the data set with 150 taxa and 20,000 trees. The unoptimized and and optimized times come from an average of 10,000 searches of each size for each version of the algorithm.

that size on this data set. The blue line marked with circles shows the unoptimized query algorithm. The average runtime for a successful search containing two taxa is about .16 seconds. As the number of taxa in the query grows the search gets faster. This is because as the query grows in size there are less clades in the BST+T which could provide match to the query. Less matches means less work is done to return the results. The green line marked with squares shows the time with the early finish optimization turned on. This is the optimization which allows the search to stop once all the trees in the data set have been added to the result set. The average time for a successful query for two taxa with this optimization turned on is .027 seconds, which is more than six times faster than the unoptimized search time for the same queries.

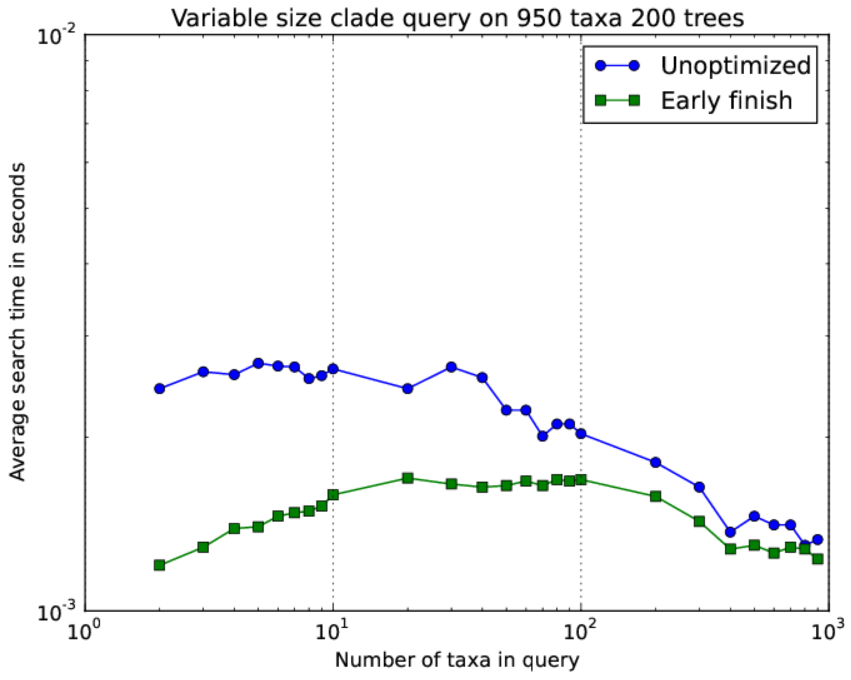


Figure 5.8: Unoptimized and optimized timing results for variable clade queries on the data set with 950 taxa and 200 trees. The unoptimized and and optimized times come from an average of 10,000 searches of each size for each version of the algorithm.

The effect of this optimization is most prominent on data sets with many trees. This is because the work that is being skipped involves allocating memory for the result trees and determining if they are already in the result set. This work load is much smaller when there are fewer trees and so there is less speed up from the optimization. We show this with plot in Figure 5.8. These experiments are conducted similarly to the ones just discussed but on a different data set. This data set has 950 taxa but only 200 trees. The first thing to notice is that average runtimes on this data set are much faster than on Data set #1. A optimized successful two taxa query averages 0.0012 seconds. The optimized algorithm does not exhibit the same general downward trend that is exhibited by the unoptimized algorithm. Instead there is a bump in the middle. This is due to the nature of the optimization. Queries with few taxa are likely to find many matches in the BST+T. This makes it more likely on average that all of the trees will be added to the result set and the search can terminate early. As the size of the queries grow matches are less likely to be found and the effect of the optimization is weakened.

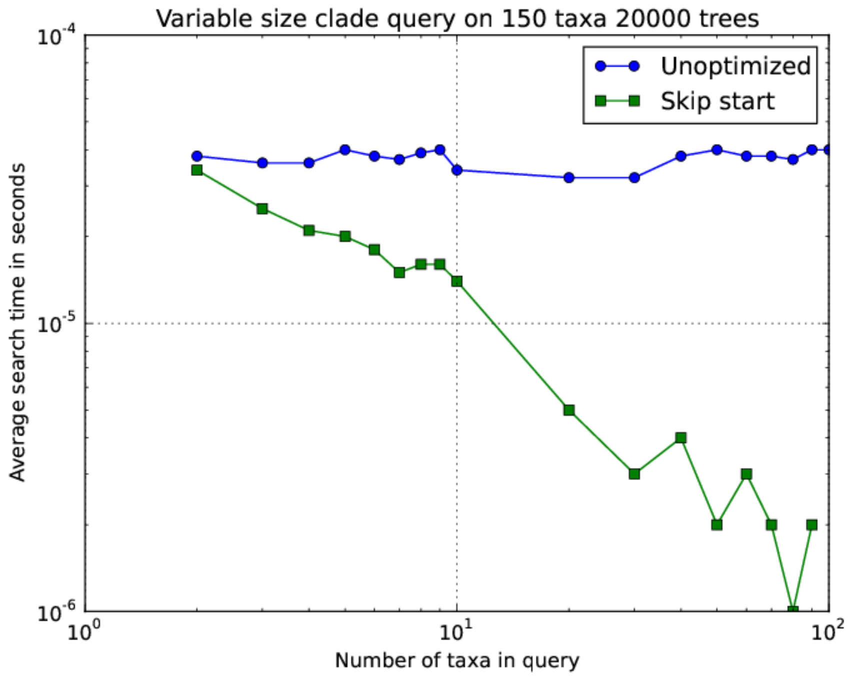


Figure 5.9: Unoptimized and optimized timing results for variable clade queries on the data set with 150 taxa and 20,000 trees using a second optimization. This optimization allows the algorithm to skip clades which are known to be too small to possibly provide a match with the query relationship. The unoptimized and and optimized times come from an average of 10,000 searches of each size for each version of the algorithm.

The early termination optimization has no effect on unsuccessful queries. Since unsuccessful queries do not find any matches in the BST+T they are not able to terminate early. However, unsuccessful searches make for an ideal experimental set for exploring certain aspects of the how the data sets related to the search algorithms as they allow us to remove a variable from the analysis. Unsuccessful queries are only effected by the number of taxa in the data set and the number of unique clades.

Figure 5.9 shows a set of experiments similar to those in Figure 5.7 except these experiments provide the average time for an unsuccessful query. The blue line is once again the unoptimized search algorithm. However, this time the optimization that has been turned on is the ability for the search to skip clades which are too small to contain a possible match to the query relationship. The unoptimized algorithm hold fairly steady as the optimized algorithm gets faster with the growing query size. This drop in running time is expected. As the queries grow in size there is less of the BST+T that needs to be traversed. The unoptimized algorithm traverses the whole search structure regardless of the size of the search so its speed holds steady.

Figure 5.10 shows the speed of unsucessful queries on the 950 taxa 200 tree data set. In contrast to the two figures showing differing behavior across the two data sets for successful search times the unsuccessful searches behave more similarly. This is because the number of trees in the data set has no effect on the time it takes to process an unsuccessful query.

In TreeHouse misses are much faster than successful searches. In fact, missing is so fast under certain conditions that is becomes difficult to accurately track the amount of time spent on the query. As the green line marked with squares, which represents the optimized algorithm, drops below 10^{-5} seconds we begin to reach the limits of our ability to produce an accurate time for the algorithm. In this experiment the search algorithms were run 10,000 times in a row and for some of the values the

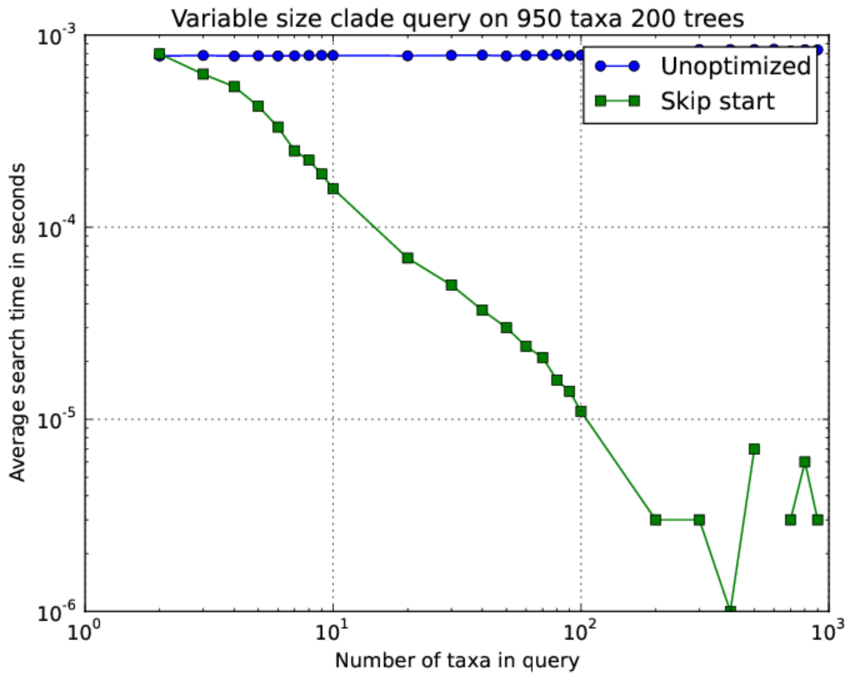


Figure 5.10: The unoptimized and and optimized times come from an average of 10,000 searches of each size for each version of the algorithm.

timing mechanism still reported zero total seconds for all 10,000 searches.

5.5.3 K-tet queries

K-tet queries function very similarly to the variable clade size queries and use the basic optimization. A major difference is that in a K-tet query special care needs to be taken for taxa heterogeneous data sets which is not required for the clade based searches or K-tet queries on taxa homogeneous data sets.

The K-tet query algorithm in TreeHouse works for heterogeneous trees similarly to the way it works for taxa homogeneous trees with one major change. The search algorithm compares the query relationship to each entry in the BST+T and produces a set which contains all trees related to matching relationships. Then, if the set of trees is heterogeneous, the algorithm traverses a secondary structure which simply stores all the trivial relationships. These are relationships that do not contain any information bout the structure of the tree, but simply keep track of which trees have which taxa. This search produces the set of trees which contain all of the taxa in the query relationship. The result of the search is the intersection of the set of trees which matched the query relationship and the set of trees which contain all of the taxa in the query relationship.

A set of experiments were run to measure the impact of this extra work on a query. To conduct this experiment, searches were run on homogeneous data sets which were treated as taxa heterogeneous by the query algorithms. This was done in order to have a direct comparison between the time it would take to do the same search on a taxa heterogeneous data set as it would to do it on a homogeneous data set. Figure 5.11 shows the time average search time for the two runs of the query algorithm. While the heterogeneous version does require extra work and therefore extra time, it is still well under a second in average runtime for the queries we examined.

5.5.4 Exploring a tree set with relationship queries

TreeHouse introduces the ability to search a set of trees with edge based relationships. This allows users to search for trees containing a certain bipartition, quartet, or anything in between. To understand how the results of these searches differ, it helps to understand how these relationships are related to one another in the context of our searching algorithm. Given a taxa homogeneous set of trees the simplest informative search is a 2|2 pronounced as “2 by 2” which is more commonly known as a quartet. Searching for something simpler, such as [A, | C, D], would return every tree since the trivial bipartition that attaches A to the rest of the tree would satisfy the query relationship. Quartets are rather common. A tree with n taxa contains

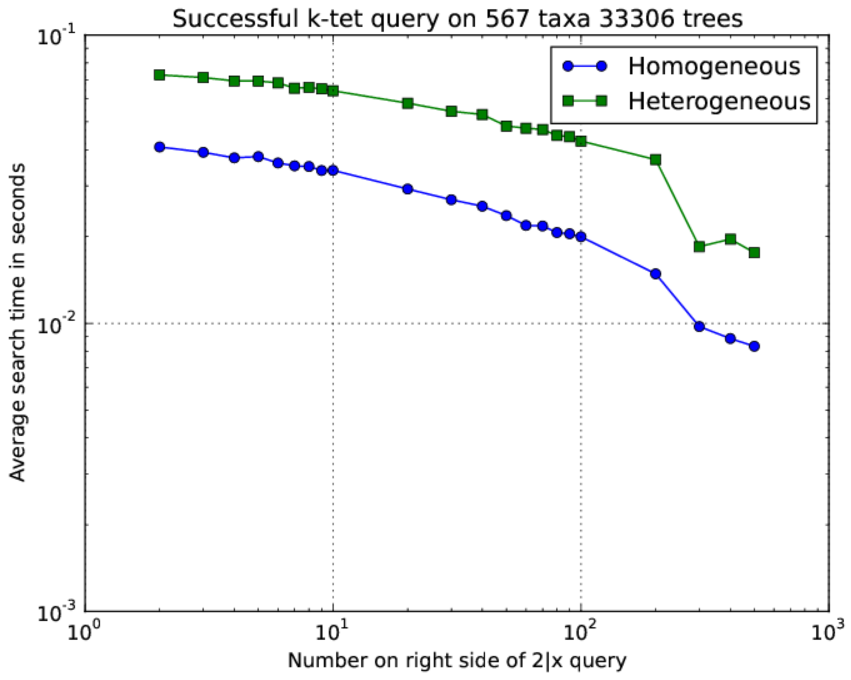


Figure 5.11: Heterogeneous and homogeneous timing results for K-tet queries on the data set with 567 taxa and 33,306 trees. The heterogeneous and homogeneous experiment times come from an average of 10,000 searches of each size for each version of the algorithm.

$\binom{n}{4}$ quartets of the total possible $3 \times \binom{n}{4}$ quartets that could be found in trees with n taxa.

As taxa are added to the search on either side of the relationship the search becomes more strict. The results for searching [A, B | C, D, E] are a subset of the results for searching [A, B | C, D]. The strictest relationships that you can search for are bipartitions. These are the relationships that define the placement of each taxon in the set.

To illustrate the impact that the query relationship size has on search results, we've conducted experiments on two data sets. The first data set contains 33,306 trees with 567 taxa that were generated from 12 runs of Mr. Bayes and is shown in

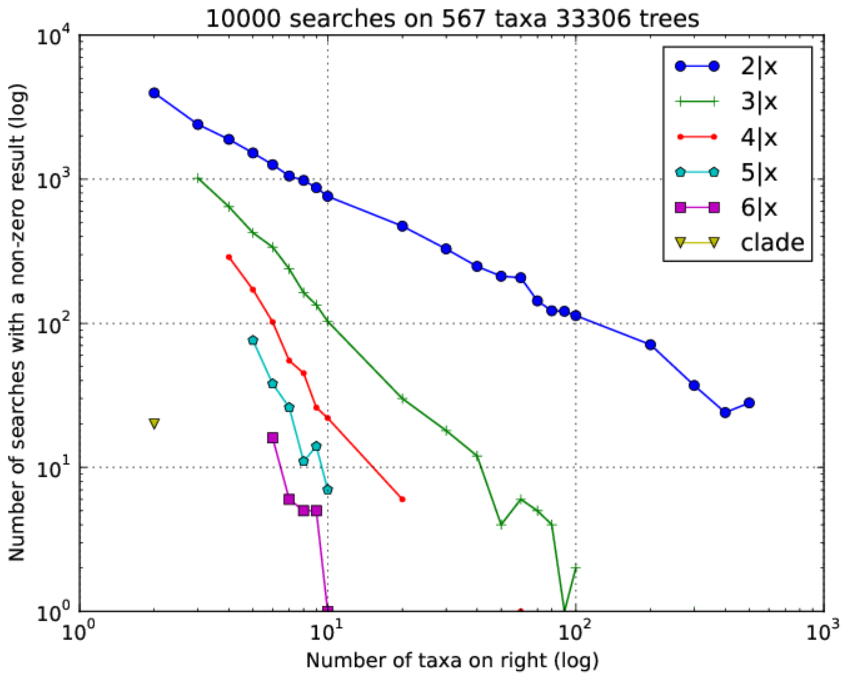


Figure 5.12: This figure shows the number of searches returning some number of trees for different sized relationships on the 567 taxa 33,306 tree data set. Each line represents different sized relationships for instance the blue line represents "2 by" searches such as "2 by 1," "2 by 2," "2 by 3," ... "2 by 500." The point on the plot labelled clade is the number of clade searches of the size on the x-axis which returned a result. Each data point represents the results of 10,000 random searches. Some lines end before 500 on the x axis because they had no successful searches for some values and 0 is not represented on this log scale plot.

Figure 5.12. The second data set contains 200 trees each with 950 taxa and is shown in Figure 5.13. For each experiment we generated 10,000 random query relationships at varying sizes. We recorded the number of those 10,000 searches which matched at least one tree in the set on the y-axis. Both experiments show similar patterns though the number of trees and taxa in the two data sets are quite different. The first thing to note is that any query relationship with a single taxon on one side will always return all of the trees in the set. As we move along the x-axis from this

common starting point, the size of the query relationship increases and the likelihood of matching a search decreases. Since smaller search queries are more general than larger search queries, we expect to see this trend results decreasing as the query relationships get larger. What is interesting is how soon some of the relationships stop getting any results at all. For instance, in Figure 5.12 the "6 by" set of searches stops returning any results after "6 by 10" in our experiments and only 2 of the 10,000 searches for "6 by 10" returned any result. These searches only used about 2.8 percent of the total taxa yet barely any of them returned results.

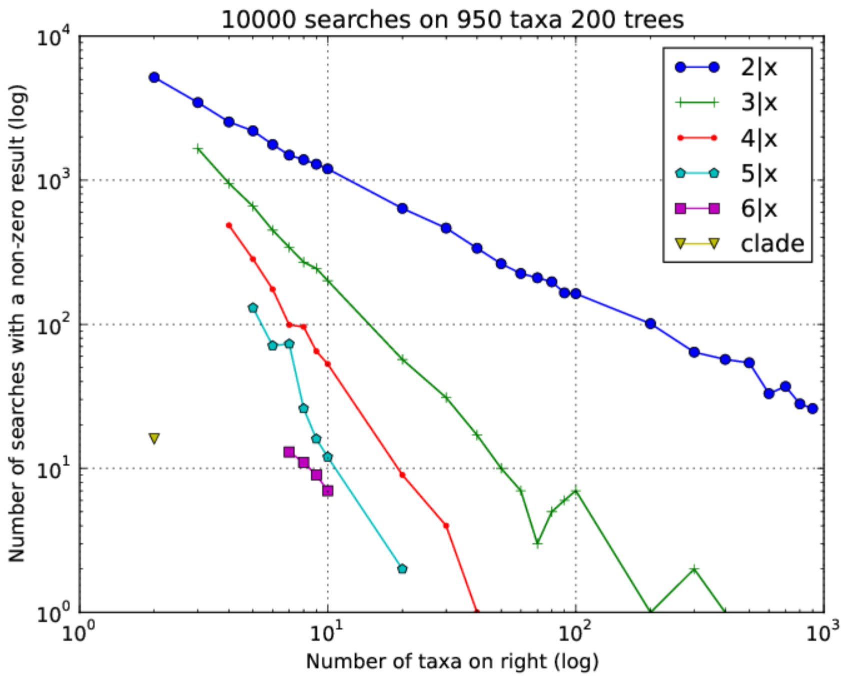


Figure 5.13: This figure shows the number of searches returning some number of trees for different sized relationships on the 950 taxa 200 tree data set.

The results of the set of "6 by 10" searches provides an interesting contrast to the "2 by" set of experiments. Notice that the "2 by 500" run had 28 of 10,000 successful searches as compared to the 2 successful searches when our query relationship was “6 by 10.” In this case a search with 502 of the 567 total taxa is more likely than one only using 16 taxa due to the way that the taxa are distributed in the query relationships. This is because while the “2 by 2” is far more likely than the “2 by 500” search, the total number of taxa isn’t only thing governing the likelihood of search results. It also matter how many taxa are on each side of the relationship. The more taxa on the smaller side of the bipartition, the less likely that you are to find a match. In fact there is an exponential decline in search results as the number of taxa on the smaller side of the relationship is increased. This effect is demonstrated in Figure 5.14. We can see from the plot that while “2 by 10” and “6 by 6” are both searching for 12 taxa, “2 by 10” is far more likely. This effect is powerful enough that it explains how searching for a “2 by 500” relationship can return more results than searching for a “6 by 10” relationship.

The impact of size of the query relationship has on the search is worth keeping in mind as it has implications for how one might approach searching with in this relationship based framework. For instance, since quartets are far more common than bipartitions or other larger relationships, they can be very useful when exploring a new set of trees. On the other hand, if you are familiar with the data set and are only interested in very specific relationships, something closer to a bipartition my be more appropriate. When dealing with large numbers of taxa the idea of typing all the taxa names to search for a bipartition may be daunting, but luckily it’s unnecessary. We’ve implemented a shortcut for a bipartition requiring the user to only input the smaller of the two sides. But, even that may be unnecessary. When we look at our search results, we can see the differentiating power of even relatively small numbers of taxa. It may be best to start with the smallest relationship that defines the trees you are interested and add more taxa only if the initial search is found to be

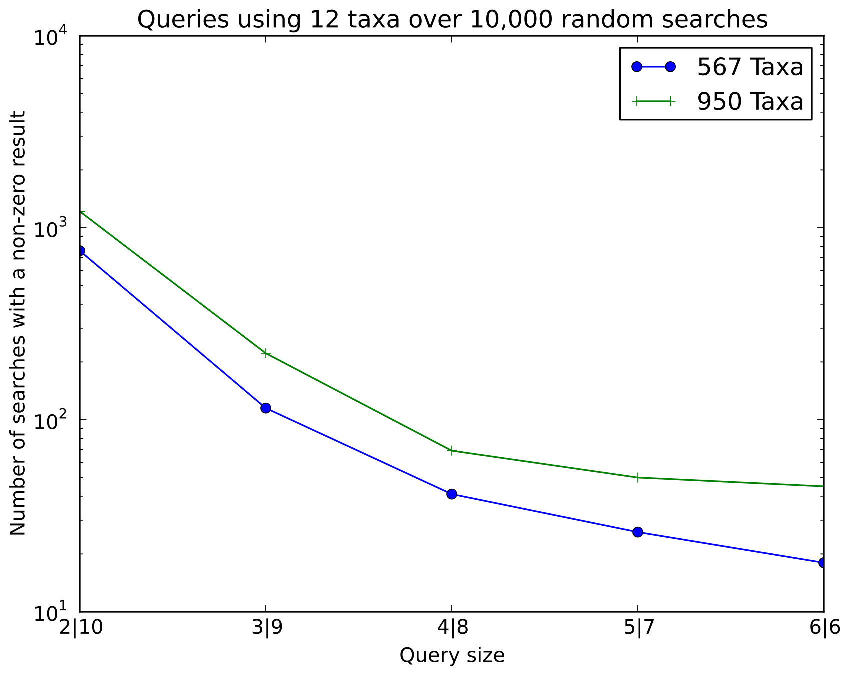


Figure 5.14: This figure shows the number of searches returning some number of trees for different relationships, each with 12 taxa, on the 567 taxa 33,306 tree and 950 taxa 200 tree data sets. Each data point represents the results of 10,000 random searches.

too broad. It's also worth remembering that the you can combine multiple query relationships to further refine a search. This allows a user to quickly search for a quartet, review the results, and if further filtering is necessary, use another quartet or a larger relationship to refine the search results.

5.6 Conclusions

TreeHouse provides users with a fast and robust method for structurally searching phylogenetic trees. This is important as allows users to explore their data sets in ways that were not previously possible. TreeHouse handles a wide range of search queries, works for both taxa heterogeneous and homogeneous searches, and has been shown to quickly return results for large data sets. Many of the search features in TreeHouse are unique to this platform. Previous work has solely focused on searching with subtrees. TreeHouse provides this functionality but also allows for other types of search. Clades are fundamental to the biological discussion of phylogenetic trees, but there have not been search tools geared toward querying trees in this way. Also new to my work are edge based relationship queries. While bipartitions and quartets have been previously defined, but they are were not used in queries until now. Completely unique to my work is the concept of the K-tet which extends the way users can define edge based relationships for the purpose of queries.

There is an overwhelming amount of phylogenetic data being produced, and algorithms can help make sense of that data. Structural search should be as fundamental a task as consensus. Quick computation of the consensus of hundreds of thousands of trees has been available to the field, but it has been much harder to find which of those trees support a given hypothesis. TreeHouse can help bridge that gap.

6. TOOLS FOR TREE ANALYSIS

The initial goal of TreeHouse was to create a fast and efficient way to conduct structural searches on a large set of trees. The goal for TreeHouse was expanded when it was realized the power of search is compounded by analysis. The new model for TreeHouse became one where a user can quickly search for a set of trees which match a certain criteria then conduct an analysis on that set of trees. To meet these new goals a series of analysis algorithms have been implemented in TreeHouse. The analysis algorithms will be discussed in three sections.

This section covers many functions such as consensus and distance measures which are available outside of TreeHouse. Also discussed in this section are new ideas such as least conflict consensus and relational entropy. Methods for handling taxa heterogeneous data sets are also discussed. There is interest in these algorithms for two reasons. The first is that TreeHouse places these algorithms in a context where they can be quickly and easily combined. TreeHouse allows a user to search for a set of trees, refine them if needed, and directly pass them to analysis function which can turn data into information. The second is that they serve as components in more complex analysis functions which are covered in the following two sections. Those sections give examples of the types of novel in depth analysis tools that can be built using the tools in TreeHouse.

6.1 Consensus

Consensus methods summarize a set of trees as a single consensus tree. A consensus tree built from the bipartitions of the set of trees. TreeHouse has the ability to construct four types of bipartition based consensus trees. Each method uses a different set of rules for determining which bipartitions from the set of input trees should appear in the consensus tree.

Strict consensus trees are constructed from only the bipartitions that appear in each input tree. Applying a strict consensus algorithm to our example *Panthera* data set produces a star topology since there are no bipartitions shared by all of the taxa in the set.

Majority consensus trees are often constructed from bipartitions that appear in more than half of the input trees. Majority consensus can also refer to trees with any threshold for inclusion between more than half of the trees and all of the trees. It is possible to define 99%, 75% or 53% majority trees. A 100% majority tree is know as a strict consensus tree. The minimum threshold for inclusions for majority consensus trees is at being in more than half of the input trees. This is because bipartitions that appear in the input set of trees may be incompatible with one another. Two incompatible bipartitions can not appear in the same tree. By limiting the bipartitions to those that appeared in more than half the trees, the possibility of incompatible bipartitions being included in the consensus tree is excluded.

The first step in constructing a greedy consensus tree is to sort the set of bipartitions by how common they are in the input tree set. The bipartitions that appear in all of the trees are added to the consensus tree. Any bipartition which is in conflict with a bipartition in the tree is removed as a potential consensus bipartition. Of the remaining bipartitions, the next more common is added and any bipartitions which conflict with it are excluded from the set of potential consensus bipartitions. This is continued until there are no non-conflicting potential bipartitions left. The resulting greedy consensus tree contains all the bipartitions in the strict and majority consensus trees, but also attempts to fill in the most common bipartitions which appeared in half or less of the input trees.

Some what similar to the greedy consensus tree is the least conflicting consensus tree. This tree is computed by adding bipartitions based on how few other bipartitions they are incompatible with. The tree starts with all of bipartitions in the strict consensus tree since they are in conflict with no other bipartitions in the set. The remaining set of bipartitions is ranked based on how few bipartitions in the set they conflict with and the least conflicting bipartition is added to the set. The bipartitions which do conflict with any bipartition in consensus tree are removed as potential consensus bipartitions and the remain bipartitions are re-ranked. This process continues until there are no remaining bipartitions that could be added to the tree. This type of consensus tree has similarities to the greedy consensus in that the consensus tree is fully resolved. However, since it is built with a different inclusion criteria it can be structurally different than the greedy consensus tree.

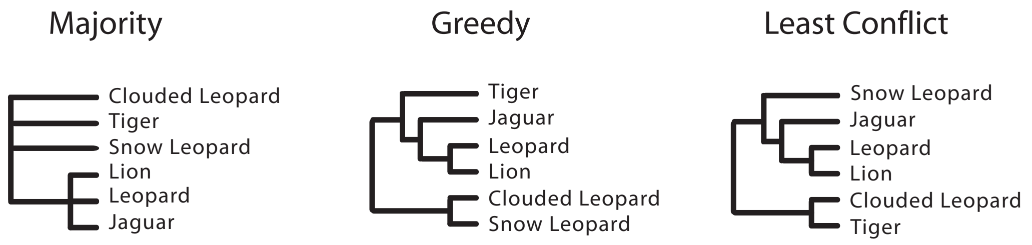


Figure 6.1: Three different consensus trees for 13 of the 14 trees in the *Panthera* data set from Figure 1.4. Tree 8 contains a different set of taxa than the other 13 trees. It was removed because the consensus methods shown require all of the input trees to contain the same taxa.

Figure 6.1 shows example consensus trees for the 50% majority, greedy and least conflicting consensus tree. The strict consensus tree contains no bipartitions and is not shown. The *Panthera* data set was used for these computations but was slightly modified. Tree 8 which is the tree that contains the cheetah and cougar was removed from the consensus computations. This is because the consensus computations are only well defined when each tree in the consensus computation is taxa heterogeneous. Due to the diversity of relationships in the tree set the majority consensus tree is not well resolved and only contains a single bipartition. Greedy and least conflicting consensus trees are always fully resolved and so they can provide some insight into the data set which is missing from a strict or majority consensus tree. Notice that the one bipartition that is in the majority consensus tree is also in the greedy consensus tree. All the bipartitions in the majority consensus will always appear in the greedy consensus.

The greedy and least conflicting consensus tree share a similar topology but the position of the snow leopard and tiger are swapped. These trees share two clades but say very different things about how the tiger, snow leopard, and clouded leopard are related to the other cats.

6.1.1 Resolution rate

The resolution rate of a tree is the percent of bipartition that appear in a tree divided by the $n-3$ possible bipartitions, where n is the number of taxa, that could have appeared in the tree. The resolution rate of a consensus tree can be used to measure the similarity of the set of trees that made up the input into the consensus algorithm. The resolution rate of a single tree may also be computed. This value is useful in determining if a tree is fully bifurcating as some other measures only work with fully bifurcating trees.

6.2 Distance between trees

A fundamental question when dealing with any two objects is, ”How similar are they?” This question is often answered with a distance measure. Different distance measures compute similarity based on different features. The Robinson-Foulds [78] distance uses bipartitions and the quartet distance uses quartets. Cosine similarity takes a slightly different approach to using bipartitions as compared to the Robinson-Foulds distance.

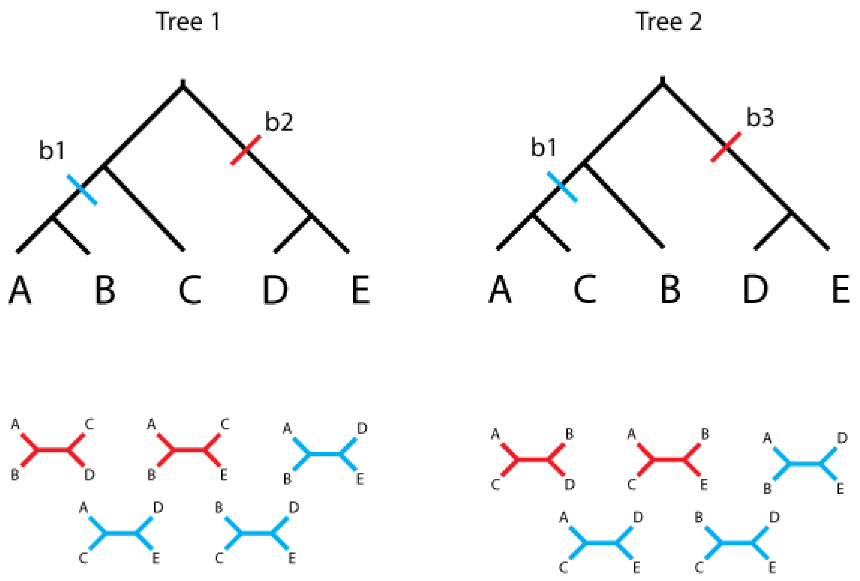


Figure 6.2: Two examples of evolutionary trees depicting the relationships between 5 taxa to show the differences between the Robinson-Foulds distance and the quartet distance. Each bipartition is labeled and the quartets making up each tree are shown. Bipartitions and quartets show in blue are shared between the two trees. Those shown in red are not shared between the two trees.

6.2.1 Robinson-Foulds distance

The Robinson-Foulds (RF) distance is ubiquitous in the field and quick to compute compared to other measures. The RF distance between two trees is computed based on the number of bipartitions that differ between them. Let $\Sigma(T)$ be the set of bipartitions defined by all edges in tree T. The RF distance between trees T_1 and T_2 is defined by Equation 6.1.

$$d_{RF}(T_1, T_2) = \frac{|\Sigma(T_1) - \Sigma(T_2)| + |\Sigma(T_2) - \Sigma(T_1)|}{2} \tag{6.1}$$

In Figure 6.2, consider the RF distance between trees $Tree$ 1 and $Tree$ 2. The set of bipartitions defined for tree $Tree$ 1 is $\Sigma(Tree\ 1) = \{AB|CDE, ABC|DE\}$. $Tree$ 2 is $\Sigma(Tree\ 2) = \{AC|BDE, ABC|DE\}$. The number of bipartitions appearing in $Tree$ 1 and not $Tree$ 2 (i.e., $|\Sigma(Tree\ 1) - \Sigma(Tree\ 2)|$) is 1, since $\{AB|CDE\}$ does not appear in $Tree$ 2. Similarly, the number of bipartitions in $Tree$ 2 but not in $Tree$ 1 is 1. Hence, $d_{RF}(Tree\ 1, Tree\ 2) = 1$. The largest possible RF distance between two binary trees is $(n - 3)$, which would be a maximum RF value of 2 in the example shown in Figure 6.2.

Computing the RF distance between trees can be done quickly with recent algorithmic developments. HashRF [92], HashRF(p, q) [90], MrsRF [56], and Phlash [57] have consecutively set the bar for both speed and the number of trees handled. TreeHouse has an implementation of the RF distance which is intended for quick computations. When it is necessary to produce large all to all distance matrices TreeHouse provides a function to pass a set of trees to Phlash.

6.2.2 Quartet distance

The quartet distance [10] is very similar to the RF distance. The quartet distance between two trees is based on the number of quartets that differ between them. Let $Q(T)$ be the set of quartets contained in tree T. The quartet distance between trees T_1 and T_2 is defined by Equation 6.2.

$$d_{Quartet}(T_1, T_2) = \frac{|Q(T_1) - Q(T_2)| + |Q(T_2) - Q(T_1)|}{2} \tag{6.2}$$

For example, in Figure 6.2 there are two quartets unique to $Tree$ 1 and two that are unique to $Tree$ 2. This produces a quartet distance of $\frac{2+2}{2} = 2$.

Computing the quartet distance between trees is more computationally intensive than computing the RF distance. This is due to the sheer number of comparisons to be made between the trees. While there are only $n-3$ bipartitions in a tree there are $\binom{n}{4}$ quartets. Quartet distances can be computed using QDist [52] or with the more efficient algorithm QuickQuartet [15]. TreeHouse has an implementation of of a quartet distance measure which is intended for small computations. TreeHouse also provides the ability to pass trees to QuickQuartet when a faster algorithm is needed.

6.2.3 Clade distance

The clade distance is distance measure implemented in TreeHouse. It is based on the same calculations as the RF distance but counts clades instead of bipartitions. Let $Cl(T)$ be the set of clades defined by all edges in tree T. The clade distance between trees T_1 and T_2 is defined by Equation 6.3.

$$d_{Clade}(T_1, T_2) = \frac{|Cl(T_1) - Cl(T_2)| + |Cl(T_2) - Cl(T_1)|}{2} \tag{6.3}$$

The clade distance is very similar to the RF distance in many situations, though there are exceptions. Since the concept of a clade does not make sense on an unrooted tree, the clade distance is only applicable to comparing rooted trees. This distance is also affected by how the trees are rooted. Two trees could have the same bipartitions, but have different clades due to rooting. A benefit of the clade distance is that it can be used to compare trees with differing taxa, since only the taxa within a clade are considered when determining if two clades are the same. The largest possible clade distance between two binary trees is $(n-2)$.

The relationship between the clade distance and the RF distance can be shown in Figure 6.2. Consider Tree 1 which has two bipartitions but three clades. Tree 1 has the clades [A,B], [A,B,C], and [D,E]. Bipartition b_1 defines the clade [A,B] and bipartition b_2 defines two clades, [A,B,C] and [D,E]. The bipartition nearest to the root will always define two clades, one on either side of the root. The clade distance between the trees in Figure 6.2 is 1 which is the same as the RF distance. However, the values are interpreted slightly differently since the clade distance has a maximum of three and the RF distance has a maximum of two. For an example on how rooting can affect the outcome of the distance measures, Figure 4.6 shows two trees with the same three bipartitions. However, due to being rooted differently these two trees do not have all of the same clades. The RF distance between these trees is 0 while the clade distance is 2.

6.2.4 Cosine similarity

TreeHouse also implements a cosine similarity measure. To compute this measure, a bit string with a place for each unique bipartition in the set of trees is created. Bipartitions contained in a tree are marked with a one, and bipartitions that a tree does not contain are marked with a zero. This bit string uniquely defines the tree. The similarity between two trees can then be computed as the cosine similarity of their bit strings using the equation in Equation 6.4, where the bit string representing the bipartitions contained in one tree is A and the bit string representing the bipartitions in the other tree is B.

$$cosinesimilarity = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum\limits_{i=1}^{n} A_i \times B_i}{\sqrt{\sum\limits_{i=1}^{n} (A_i)^2}} \tag{6.4}$$

In the general case of cosine similarity the possible values fall within the range of -1 to 1. However, our application of the measure is limited to values between 0 and 1 because none of our entries in the bit strings are negative. A value of 1 means that the two trees had all of the same bipartitions and therefore have the same topology. A value of 0 means they are independent and have no biparititons in common. This method for computing distances between trees can be very quick and useful in analyses such as clustering which require fast distance computations. The cosine similarity is especially efficient for sparse vectors as only the non-zero dimensions need to be considered. Since a tree can only have $n - 3$ bipartitions, where n is the number of taxa in the tree, the maximum number of positions that need to be considered to compute the cosine similarity is $2(n - 3)$.

6.3 Distance within a tree

TreeHouse also provides algorithms to compute various distances within a single tree. These distances are defined by the placement of taxa and the tree's topology. For any taxa in a tree, TreeHouse can compute the length of the path between that taxa and the root of the tree. This value is the depth of the taxa and is dependent on the number of branching events in the path between that taxa and the root. In the tree in Figure 6.3 the clouded leopard has a depth of 1, the tiger has a depth of 3 and the lion has a depth of 4.

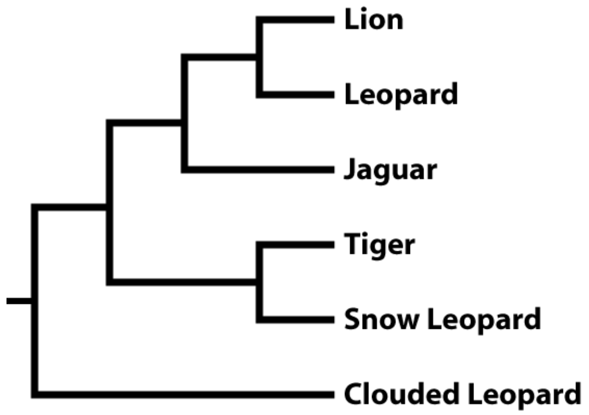


Figure 6.3: An example tree of genus *Panthera* used for examples of tree depth and balance. This tree has a maximum depth of 4, minimum depth of 1, and an average depth of 3. Its C value is .6.

For any two taxa in a tree, TreeHouse can compute the most recent common ancestor for the two taxa. This is the smallest clade which contains both of the input taxa. For instance, for the tree in Figure 6.3, the smallest clade that contains lion and jaguar is the clade that contains lion, jaguar, and leopard. TreeHouse can compute the distance from the taxa to this most recent common ancestor. The distance from lion to the most recent common ancestor of lion and jaguar is 2. TreeHouse will also compute the distance between two taxa which is defined as the minimum path between the two taxa in the tree. For two taxa A and B, the distance From A to B is equal to the distance from A to the most common ancestor of A and B plus the distance from B to the most recent common ancestor of A and B. In our example the distance from lion to jaguar would be 3.

The shape of a tree is defined by its topology. Currently, TreeHouse supports a few tree shape measures. The maximum depth of a tree is computed to be the longest path from the root to a taxon. The tree in Figure 6.3 has a depth of 4. The average depth of a tree is the average length of the paths from the root to the taxa which is 3 for our example tree. TreeHouse also computes Colless' index of imbalance [14], also known as C. This measure is based on the difference in the number of taxa under the right and child of each node. A perfectly balanced tree has a score of 0. A score of 1 denotes maximum imbalance. The C value for our example tree is .6. Equation 6.5 can be used to compute this value. The algorithm traverses a tree and computes the absolute value of the difference between the number of taxa under the right side T_R of a node and the left side T_L of a node for each i internal nodes. The sum of these values is then divided by the maximum possible difference to produce the value C.

$$c = \frac{\sum_i |T_R - T_L|}{\frac{(n-1)(n-2)}{2}} \tag{6.5}$$

6.4 Relational entropy

TreeHouse provides some functions to identify rogue taxa within a set of trees. A rogue taxon is a taxon which is an equally good fit in many positions in the trees. This may be due to some combination of the taxa selected, the sequences used, sequence alignment or model of evolution used in the inference of the trees. The extreme case would be where the taxa appears to be an equal fit in all positions in the tree to the tree inference algorithm. Taxa that behave this way can appear on both sides of many bipartitions. This makes summarizing a set of trees which contain a rogue taxon as a consensus tree very difficult. While much of the structure may be similar in the majority of trees, the indecision about the placement of a single rogue taxon causes the bipartitions in the set of trees to vary more than the structure would actually suggest. This is because for two trees to have the same bipartition the placement of every taxa relative to an edge must match.

Since the hallmark of a rogue taxon is that it appears in multiple positions in a tree, the "rogueness" of a taxon can be measured based on how many positions it appears in with in a set of trees. This done by computing the clade level entropy for a taxon. The clades in a set of trees are divided into tiers based on the number of taxa on each clade.

In determining rogueness, the number of relationships a taxon appears in at each tier of the clade table is computed. A taxa that appears in many sister relationships will be represented in more of the size 2 clades than a taxon that always appears in a sister relationship with the same taxa. By determining how many relationships at each tier a taxa appears in, as well as how common each of those relationships are, the clade level entropy for a taxa can be computed.

Table 6.1 show the results of this type of analysis for the *Panthera* example data set. Since this data was generated by multiple labs the results can not be interpreted the same ways as had they been generated by a single inference algorithm, however they still show which taxa have the most varied placement within the data set. In this case the Snow Leopard has been paired with the most other taxa in sister relationships.

This function can provide a quick way to determine which if any taxa appear in more than the average number of relationships per tier. Detecting rogue taxa in a data can allow biologists to modify or remove them from their sequence alignment and re-run their inference algorithm without their disruption.

	Number of different clades			
Taxa	Size 2	Size 3	Size 4	Size 5
Cheetah	0	0	1	0
C. Leopard	2	0	0	0
Cougar	0	1	1	0
Jaguar	2	2	2	1
Leopard	2	2	2	1
Lion	3	3	2	1
S. Leopard	4	2	2	1
Tiger	3	2	2	1

Table 6.1: The number of unique clades of different sizes that each taxa appears in.

6.5 Quartet exposure

For a given tree, TreeHouse can print all of the quartets contained in the tree. The enumeration of quartets provides the foundation for the way TreeHouse computes the quartet distance between two trees. While the consensus functions in TreeHouse are currently limited to working with bipartitions, TreeHouse can use its quartet enumeration function to produce the set of quartets which are common to all the trees in a set. TreeHouse can also compute the quartets that are common to some percentage of trees in a set. This output would form the basis for a quartet based consensus method if paired with a compatible tree building algorithm. While we are unable to directly compute the consensus tree, the information is still useful in that we can compute the number of quartets that would define the consensus tree. This is a measure of consensus resolution that is different from that computed by the other consensus resolution rate functions in TreeHouse which work with bipartitions.

6.6 Analysis of heterogeneous tree sets

While our search algorithms have been adapted to handle both taxa homogeneous and heterogeneous tree sets, the same is not true for the majority of the analysis algorithms we have discussed. Many distance and consensus algorithms are based on a definition of a bipartition which does not so easily extend to cover taxa heterogeneous sets. For instance, any distance measure defined by bipartitions, such as the Robinson-Foulds distance, is based on the number of bipartitions different between two trees. Given any two trees which contain two different taxa sets, it is not possible for the trees to have any bipartition in common. Since there can be no common bipartitions, the result would always be the maximum distance and therefore uninformative. Many definitions of consensus methods are similarly defined by matching bipartitions and are not well defined on taxa heterogeneous sets of trees.

One method of handling this situation is to use TreeHouse's search functions to return a set of trees which all contain the exact same taxa set. This subset of the total trees can then be passed to distance and consensus functions because the subset is heterogeneous.

Given the trees in Figure 6.4, one could homogenize the set by simply excluding tree T_8. This could be done by querying for the trees which contain C. Leopard as all

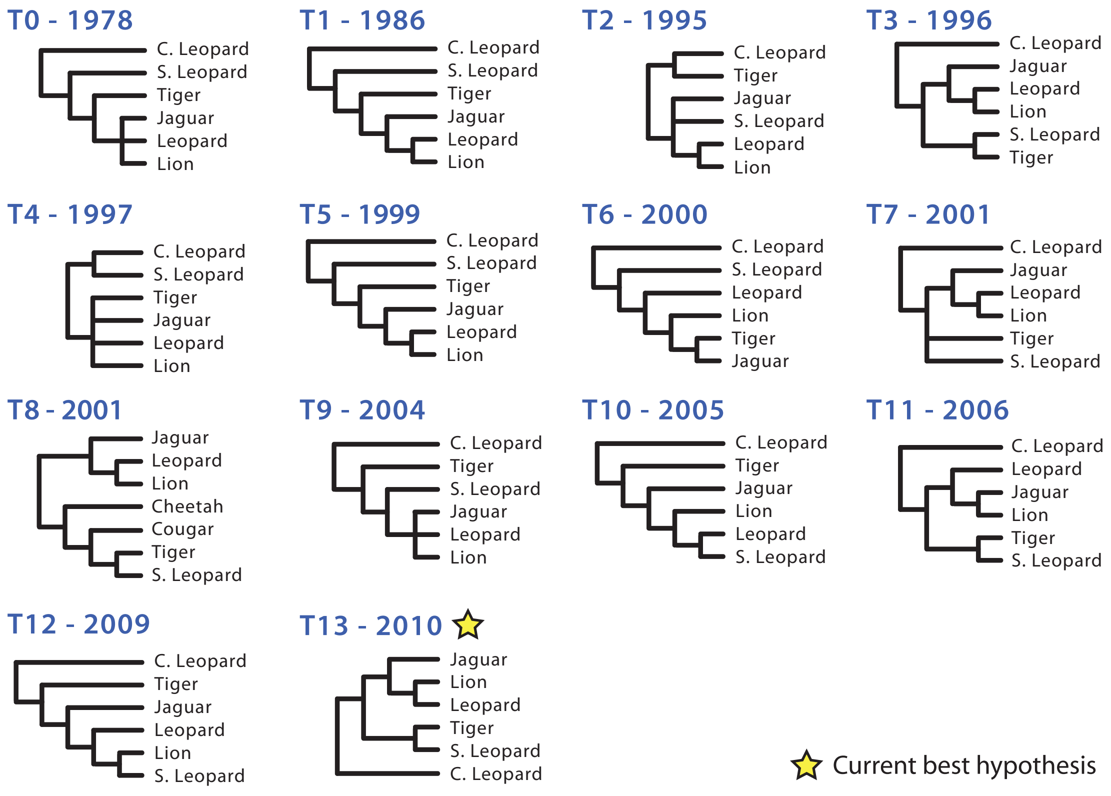


Figure 6.4: Multiple hypothesis of genus *Panthera* over the last 30 years.

of the trees which contain C. Leopard cover the same set of taxa. The resulting set could be passed to the analysis algorithms which require taxa homogeneous trees.

6.6.1 *Taxa searching and distinguishing taxa*

There are a few functions in TreeHouse that are only useful for searching and analyzing trees with different sets of taxa. The taxa search function is one such example. This search returns all the trees in the set which contain the query taxa. Another is the concept of distinguishing taxa between two sets of trees. We define the distinguishing taxa between two sets of trees to be the taxa in that appear in all of the trees in the one of the sets but none of the trees in the other. This can be a useful measure for differentiating sets of trees but it is only useful when the trees cover different sets of taxa.

6.7 Taxa masking

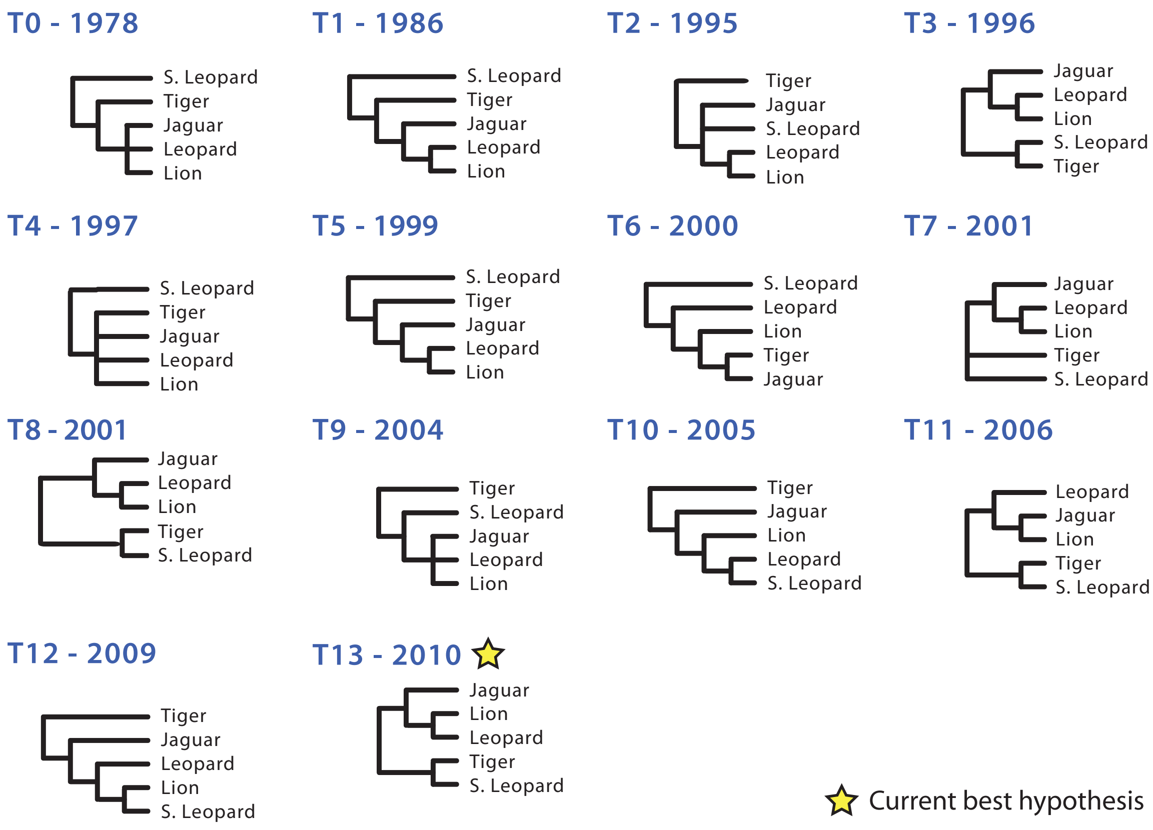


Figure 6.5: A visualization of the multiple hypothesis of genus *Panthera* over the last 30 years with Clouded Leopard, Cheetah, and Cougar removed from each tree. Extra branches have been collapsed. The tree numbers match the citations in Figure 6.4

Queries may be used to return a set of trees which are taxa homogeneous but in doing so we may be excluding information about taxa of interest in trees that cover more taxa that just those we are directly studying. This can be seen in the *Panthera* data set. Removing a single tree, in this case tree T_8, we convert the data set from being taxa heterogeneous to taxa homogeneous, but we have also lost the information tree T_8 contains about the relationships between the five cats common among all the trees in the set. To address this, TreeHouse provides functions to mask taxa in a set of trees. Masked taxa are pruned from the data set and their branches are collapsed. This leaves trees which look as though the masked taxa were never in the analysis. This function can be used to homogenize a data set so that functions which are only well defined on taxa homogeneous trees can be applied. Figure 6.5 shows the 14 trees in our heterogeneous data set with C. Leopard, Cheetah, and Cougar removed and extra branches collapsed. These trees may now be used as the input to any of the analysis functions that require taxa homogeneous trees.

Index	Size	Clade	Trees
0	2*	00000011	3, 8, 11, 13
1	2	00000110	11
2	2	00001100	1, 2, 3, 5, 7, 8, 13
3	2	00001010	9
4	2	00010001	6
5	2	00010100	10
6	3*	00001110	10, 12
7	3	00010101	6
8	3	00011100	0, 1, 3, 5, 7, 8, 9, 11, 13
9	4*	00011110	2, 10, 12
10	4	00011101	0, 1, 2, 4, 5, 6
11	5*	00011111	0, 1, 3, 5, 6, 7, 9, 10, 11, 12, 13

Table 6.2: The BST+T for our 14 example trees from genus *Panthera* after C. Leopard, Cheetah, and Cougar have been removed and extra branches collapsed. The bits representing C. Leopard, Cheetah, and Cougar have also been removed from the bit strings.

The bipartition table for the masked taxa set is shown in Table 6.2 and has only 12 unique clades. This table is smaller than the unmasked table shown in Table 5.1

which has 16 unique bipartitions. Using taxa masking to produce a homogeneous set of trees provides the user with a trade off in the amount and type of information in the set. The unmasked set of trees contains more bipartition information, however the presence of taxa heterogeneity between the trees limits the types of analyses that can be conducted. Homogenizing the set opens it up to many more types of analysis but a cost. In our example we lost about 25% of the total unique bipartitions in the homogenizing process, but in return we are able to construct a consensus tree of the results or compute the Robinson-Foulds distance between two of the masked trees. It also clarifies the relationships between the common taxa. Tree T_8 which had two taxa that no other tree had and was missing a taxa common to all the other trees can now be seen to have the same relationships in the remaining five taxa as trees T_3 and T_{13}.

7. ADVANCED ANALYSIS: DECISION TREES AND DISTINGUISHING BIPARTITIONS*

7.1 Why compare sets of trees

Groups of trees can be defined in many ways. They can come from clustering algorithms, they can be a product of how they were inferred by grouping trees from the different chains, runs, algorithms, or labs. They can also be the result of multiple queries on a tree set. No matter where the groups come from, analysis measures can tell us about about the groupings. In the case of trees coming from multiple runs, we can determine how topologically distinct those runs were. The same measure applied to a grouping of trees from a clustering algorithm can tell us something about whether there are distinct sub-hypothesis of the true tree in the data. This section explores algorithms used to analyze sets of trees.

Understanding the ways in which sets of trees differ from one another is important in answering questions about convergence. To say two groups of trees have converged is to say they have come from the same region of tree space. This is important question when dealing with trees resulting from a phylogenetic inference algorithm. If the trees have not converged to a single region of tree space there may be multiple distinct sets of trees which represent different hypotheses of the true tree. Understanding if and how sets of trees differ can lead to insights about the algorithms that produced them.

* Part of the data reported in this section is reprinted with permission from *"Using decision trees to study the convergence of phylogenetic analyses."* by G. Brammer, & T. L.Williams, (2010, May). In the proceedings from the 2010 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB), (pp. 1-8). © IEEE 2011.

7.2 Decision tree depth and bipartition separability

Measuring the similarity between sets of phylogenies can be thought of as a data mining style classification problem. We can ask, "How complicated would a classifier need to be to separate the phylogenies into their groupings based on their features?" We can train a classifier, in this case a decision tree, on the input trees using their bipartitions as features. The level of structural distinction in the input sets of phylogenies is directly related to the size and shape of the trained decision tree.

7.3 Decision trees

Decision tree learning is a predictive model for mapping observations to outcomes. Decision trees are built with a classification at the leaves and features at the internal nodes. For our data, the classifications could be the run, algorithm, cluster, query or other grouping of phylogenies and the internal nodes are bipartitions, and the branches leading from the those nodes denote either the presence or absence of that bipartition. The features crossed by tracing a path from the root of the tree to a leaf node describe a set of features where all trees with those features have the classification at the leaf node.

7.3.1 Algorithm

Our approach consists of two steps. First, we construct a *bipartition table* that consists of the unique bipartitions associated with our set of phylogenetic trees. The bipartition table also contains class labels concerning the run of the phylogenetic search heuristic that produced each of the evolutionary trees. The second step concerns using the bipartition table to actually create the decision using the ID3 (Iterative Dichotomiser 3) data-mining algorithm [59].

Step 1: The first step is to construct a bipartition table. Consider the set of twelve evolutionary trees depicted in Figure 7.1a. First, we enumerate all of the bipartitions in each of the evolutionary trees. Figure 7.1b shows the set of unique bipartitions for the twelve trees in our example. Each unique bipartition, b_i, will represent a column in the bipartition table. The rows of the table will consist of the input tree ids (t_i) of interest. For each column labeled b_i, a '0' or '1' is assigned to each row based on whether tree t_i contains that bipartition. Consider Figure 7.1. Bipartition b_0 appears in trees t_0, t_1, t_2, and t_3. It does not appear in the remaining eight trees. The data structure used in computing the decision trees is a transformation of the BST+T which is used in the TreeHouse search functions.

$t_0 : (((a,b),(c,d)),(e,f));$	$t_6 : (((a,d),(b,c)),(e,f));$
$t_1 : (((a,b),(c,e)),(d,f));$	$t_7 : (((a,d),(b,c)),(e,f));$
$t_2 : (((a,b),(c,f)),(d,e));$	$t_8 : (((a,d),(b,f)),(c,e));$
$t_3 : (((a,b),(c,d)),(e,f));$	$t_9 : (((a,f),(b,c)),(d,e));$
$t_4 : (((a,c),(b,d)),(e,f));$	$t_{10} : (((a,f),(b,d)),(c,e));$
$t_5 : (((a,e),(b,d)),(c,f));$	$t_{11} : (((a,e),(b,c)),(d,f));$

(a) Newick strings

$b_0 : ab\|cdef$	$b_5 : bc\|adef$	$b_{10} : ce\|acef$
$b_1 : ac\|bdef$	$b_6 : bd\|acef$	$b_{11} : cf\|acde$
$b_2 : ad\|bcef$	$b_7 : be\|acdf$	$b_{12} : de\|abcf$
$b_3 : ae\|bcdf$	$b_8 : bf\|acde$	$b_{13} : df\|abce$
$b_4 : af\|bcde$	$b_9 : cd\|abef$	$b_{14} : ef\|abcd$

(b) Unique bipartitions

Figure 7.1: Twelve evolutionary trees used as input to build the bipartition table in Table 7.1. (a) shows the Newick representation of the twelve phylogenetic trees of interest. (b) provides a listing of the unique bipartitions that appear across the twelve trees. ©2011 IEEE

Bipartition table																
Trees	b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}	b_{13}	b_{14}	run
t_0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	r_1
t_1	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	r_1
t_2	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	r_1
t_3	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	r_1
t_4	0	1	0	0	0	0	1	0	0	0	0	0	0	0	1	r_1
t_5	0	0	0	1	0	0	1	0	0	0	0	1	0	0	1	r_1
t_6	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	r_2
t_7	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	r_2
t_8	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	r_2
t_9	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	r_2
t_{10}	0	0	0	0	1	0	1	0	0	0	1	0	0	0	0	r_2
t_{11}	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0	r_2

Table 7.1: A bipartition table depicting the presence ('1') or absence ('0') of each bipartition (labeled b_i), in the twelve trees shown in Figure 7.1 along with a column denoting the run that generated the tree. The list of bipartitions are our features and the run is our class label since we know what trees a particular run generated. In this example, there are two runs that generated the twelve trees. A bipartition table will serve as input for the construction of a decision tree. ©2011 IEEE

Step 2: The next task is to build a decision tree from the bipartition table. Once the bipartition table is constructed, we use our implementation of the ID3 algorithm, which is a top-down greedy approach that selects features based on information gain. For our problem, the features are bipartitions, which are the internal nodes of the decision tree. Class labels (runs and algorithm) are the leaves of the decision tree. Class labels are also considered target features.

The ID3 algorithm takes a feature set (the presence or absence of each bipartition) and a target feature (i.e., which run the evolutionary trees came from) and computes a decision tree, where each node in the tree represents the a bipartition and each edge represents the presence or absence of that bipartition. Table 7.1 shows an example bipartition table as derived from the Newick strings in Figure 7.1a. Note the last column in the bipartition table is run label and not bipartition information. The

ID3 algorithm computes the information gain of each bipartition relative to the run label. Information gain represents how well the bipartition information correlates with the run labels. The bipartition with the best information gain is selected and added as the root of the decision tree.

Equation 7.1 measures the entropy of a collection of trees relative to their labels. For a given set S of examples, the sum function iterates over all the different example values with p_i being the percentage of the examples that have that value. To compute the entropy of the run labels, S is the rightmost column of Table 7.1. We would iterate over the labels r_1 and r_2 with p_i being the percentage of each label.

$$Entropy(S) = \sum_{i=1}^{c} -p_i \; log_2 p_i \tag{7.1}$$

Equation 7.2 computes information gain, where S is the set of examples and A is an attribute. In this equation, $Values(A)$ is the set of values that the attribute can have. For each v in $Values(A)$, S_v is the a subset of S which has that value.

$$Gain(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{S} Entropy(s_v) \tag{7.2}$$

In our example in Figure 7.2, bipartition b_0 is the root. All the trees with bipartition b_0 come from run r_1. Hence, the right child node is marked as run r_1 and has no children. The trees that do not have bipartition b_0 are from both runs so the process repeats to calculate the node with the most information gain. The process continues until all the branches terminate into leaf nodes. Figure 7.2 shows the resulting decision tree.

Figure 7.2 shows a sample decision tree where the all the trees are classified as coming from run 1 (r_1) of an algorithm or run 2 (r_2) of the same algorithm. Three bipartitions, b_0, b_4, and b_6 are used in the decision tree to classify all of the input trees.

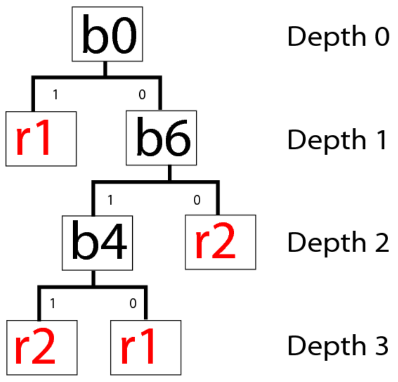


Figure 7.2: A decision tree constructed based on the input from Table 7.1. Here, 3 bipartitions (out of 15) are needed to classify the twelve input trees by what run generated them. ©2011 IEEE

From each internal node there are two branches, the left branch denotes the presence of the bipartition and the right branch denotes the absence of the bipartition. Each path tells us something about the phylogenies in the input sets. For instance, we can follow the left most path from $b_0- > r_1$ which tells us that all trees with b_0 are classified as r_1. The right most path, $b_0- > b_6- > r_2$ tells us that all trees which do not have b_0 or b_6 were classified as r_2.

7.4 Interpreting decision trees

Once the decision tree is created, we can analyze its features to learn about the input sets of trees. The depth of a decision tree is the crucial feature for determining whether a phylogenetic analysis, consisting of multiple runs, has converged. We define the depth of a decision tree to be the length of the longest path from the root to a leaf node. Deep (high depth) decision trees reflect a high level of sharing among the evolutionary trees since many evolutionary relationships have to be consulted in order to classify what runs produced the evolutionary trees. Shallow decision trees reflect less sharing of information of across the runs. Hence, deep decision trees provide strong evidence that a phylogenetic analysis converged while extremely shallow trees reflect non-convergence.

7.4.1 Edge cases: distinguishing bipartitions and non-separable sets

We have proposed the term *distinguishing bipartition* to identify a bipartition which can be used to distinguish between two groups of trees. These are bipartitions that appear in all of the trees in one group and none of the trees in another group. These bipartitions result in a decision tree of depth one meaning that a single feature is needed to classify all of the input trees. A distinguishing bipartition means that the two sets of phylogenies represent distinct hypotheses of the true relationships between the taxa. The distinguishing bipartitions are major differences between the hypotheses. It only takes a single distinguishing bipartition to result in a decision tree of depth one, but many could be present in the sets of trees. TreeHouse provides an algorithm to return the number of distinguishing bipartitions between two sets of trees.

While it is possible for the decision tree to have a depth of one, it is also possible for a decision tree to be unable to classify all the trees into their classifications based on their bipartitions. One example of this is when two trees with the same topology are classified into different groups. Since each tree shares the same bipartitions, and therefore shares the same feature set, no features can be used to tell them apart. Tree sets that can not be matched with their classifications by any combination of their features are said to be non-separable. Non-separable sets overlap topologically and can be said to have converged.

7.5 Background

Our work leverages the idea that the depth of a decision tree can be used to measure the quality of a cluster. A similar idea has been used to develop new clustering methods [50]. Decision trees and feature selection are well researched areas in the machine learning community but, they have not yet been thoroughly applied to phylogenetic data sets. In many ways, the work most similar in methods to ours is that which attempts to measure the quality of clusters produced by clustering phylogenetic trees. Stockham, Wang, and Warnow [88] cluster phylogenetic trees and use the concept of information loss as a measure of cluster quality, but do not address the issue of convergence. Unlike our work, most techniques for detecting convergence use tree scores [76], but rarely do they compare the evolutionary relationships contained in the trees to each other. Comparing tree scores can be misleading since trees with similar scores are not necessarily topologically similar, leading to misleading results [34, 68, 67, 91]. Hence the fact that a search that found two different optima in tree space could be detected with a topology based method but left undetected by relying on tree scores alone.

7.6 Experimental results

We've used decision trees to measure the level of convergence between trees produced by multiple runs of the MrBayes algorithm. We've used datasets #1, #2, and #5 in our analysis which are described in detail in the appendix.

7.6.1 150 taxa Bayesian trees

The first data set has 150 taxa and 20,000 phylogenies. The phylogenies come from two runs of equal size. The depth of the decision tree created in comparing the two runs is 1. Since the depth of the decision tree is 1, further analysis was done to determine the number of distinguishing bipartitions between the two runs of the algorithm. There were found to be 3 distinguishing bipartitions. These three bipartitions represent the phylogenetic information that separated the runs of the phylogenetic inference algorithm. The division between the runs may be an artifact of how the inference algorithm navigated the solution space with each run being stuck in its own local optima. No matter the explanation, the implication is that the two searches did not cover the same area of tree space.

These distinguishing bipartitions contain information about the divide between the two sets of trees that would not appear in a standard consensus tree. Since the two runs we analyzed are the same size, a distinguishing bipartition can not appear in the majority of the trees. This interesting information which is important to understanding how the searches navigated the solution space is hidden by a consensus analysis.

Decision Tree Depth for Dataset #2: 567 taxa Bayesian trees												
run	0	1	2	3	4	5	6	7	8	9	10	11
0	-	1	1	1	1	1	1	1	1	1	1	1
1	-	-	1	1	1	11	1	1	1	1	1	1
2	-	-	-	1	1	1	1	1	1	1	1	1
3	-	-	-	-	1	1	1	1	1	2	2	1
4	-	-	-	-	-	1	1	1	1	1	1	1
5	-	-	-	-	-	-	1	1	1	1	1	1
6	-	-	-	-	-	-	-	1	1	1	1	1
7	-	-	-	-	-	-	-	-	1	1	1	1
8	-	-	-	-	-	-	-	-	-	1	1	1
9	-	-	-	-	-	-	-	-	-	-	2	1
10	-	-	-	-	-	-	-	-	-	-	-	1
11	-	-	-	-	-	-	-	-	-	-	-	-

Table 7.2: This table shows the depth of the decision trees created by comparing the twelve runs of data set 2.

7.6.2 567 taxa Bayesian trees

Data set #2 contains phylogenies from 12 runs of MrBayes with each tree containing 567 taxa. The depths of the decision trees created in comparing each run to each other run is presented in Table 7.2. The majority of run to run comparisons resulted in decision trees of depth 1. Only 4 run to run comparisons do not have distinguishing bipartitions and result in deeper decision trees. Of the 4 deeper decision trees 3 are of depth 2 and the deepest decision tree is of depth 11. From this we can say that the two most similar runs are run 1 and run 5. For the decision trees of depth 1, we computed the number of total distinguishing bipartitions between the runs which are presented in Table 7.3. The number of distinguishing bipartitions that separate the runs of trees vary from none to 41.

Given the high number of distinguishing bipartitions present between runs in this set of phylogenies, it would be difficult to conclude that these runs have converged to the same area of tree space. The trees in this data set may be generally similar to one another but there are strong distinctions between the bipartitions reported from each run.

Distinguishing Bipartitions for Dataset #2: 567 taxa Bayesian trees												
run	0	1	2	3	4	5	6	7	8	9	10	11
0	-	15	20	13	23	16	41	17	20	26	24	34
1	-	-	11	2	9	-	32	3	3	10	9	20
2	-	-	-	1	4	13	40	11	11	4	4	30
3	-	-	-	-	3	2	26	1	1	-	-	22
4	-	-	-	-	-	10	38	11	11	3	3	27
5	-	-	-	-	-	-	33	3	3	11	10	20
6	-	-	-	-	-	-	-	30	30	39	36	40
7	-	-	-	-	-	-	-	-	1	10	9	21
8	-	-	-	-	-	-	-	-	-	10	12	24
9	-	-	-	-	-	-	-	-	-	-	-	30
10	-	-	-	-	-	-	-	-	-	-	-	29
11	-	-	-	-	-	-	-	-	-	-	-	-

Table 7.3: This table shows the number of distinguishing bipartition which separate the twelve runs of data set 2.

7.6.3 567 taxa maximum parsimony trees

We have also applied decision trees to evolutionary trees obtained from two maximum parsimony (MP) heuristics, Rec-I-DCM33 and Pauprat. Table A.3 shows the number of evolutionary trees collected from each run. Each of the MP heuristics required five runs to produce its set of evolutionary trees. Runs r_0, r_1, r_2, r_3, and r_4 were generated by Pauprat while the remaining runs were obtained from Rec-I-DCM33.

Table 7.4 provides the depth of the resulting decision trees using runs as the target feature. The results show that the Rec-I-DCM33 runs are quite self-similar. Many of the run-by-run comparisons are non-separable, which we denote by the label NS. In other words, the two runs being compared returned at least one identical tree. As a result, the bipartition information cannot separate the evolutionary trees across the runs.

Decision Tree Depth for Dataset #3: 567 taxa maximum parsimony trees										
run	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
r_0	-	15	15	17	13	11	10	13	11	13
r_1	-	-	14	16	20	14	11	10	12	12
r_2	-	-	-	13	15	10	11	10	9	11
r_3	-	-	-	-	18	13	9	11	9	13
r_4	-	-	-	-	-	13	12	12	12	11
r_5	-	-	-	-	-	-	NS	NS	39	NS
r_6	-	-	-	-	-	-	-	NS	NS	NS
r_7	-	-	-	-	-	-	-	-	52	NS
r_8	-	-	-	-	-	-	-	-	-	NS

Table 7.4: The depth of the decision trees resulting from the comparison of maximum parsimony runs. NS stands for non-separable which means that the bipartitions from each run are so similar in these cases that the ID3 algorithm can not classify them.

The depths of the decision trees for the three data sets we studied show that the maximum parsimony trees represent a better example of convergence than the Bayesian trees. The Bayesian trees seemed to have gotten stuck in different local optima from run to run while the trees inferred from the parsimony analysis we studied seemed to have settled on a single optimum across the ten runs.

7.7 Conclusions

Decision trees can be used to measure the separation between sets of trees based on the distinction of the features of each group. A single bipartition could be the feature which defines a set of trees. This feature centric method for comparing sets of trees illuminates specific differences in which features are separating two sets of trees. With a group of trees representing a phylogenetic hypothesis, the features which different between the groups represent the ways in which the two hypothesis are different.

Determining whether a phylogenetic analysis has converged is an important problem in phylogenetics. Without convergence, the robustness of a phylogenetic analysis is unclear and leads to inaccurate hypotheses of how the taxa evolved from a common ancestor. Decision trees can be an effective measure of convergence for a phylogenetic analysis.

8. ADVANCED ANALYSIS: CLUSTERING AND PEAK SUPPORT *

The work with decision trees showed that multiple runs of a Bayesian analysis do not necessarily converge to a single hypothesis. The analysis with decision trees allows for users to determine if two groups of trees converged to the same place in tree space. This analysis is useful, but relied on the user having a meaningful grouping of trees, e.g. the runs which generated them or their parsimony score. A set of trees may contain groupings which don't align with how the trees were generated. For instance, a single Baysian run might return trees from two optima in tree space. Clustering algorithms can use the differences between trees in a set to attempt to identify separate groups. These algorithms can be useful in answering questions about convergence without the need of outside information about how the trees where generated or their scores.

Also discussed in this section is *p*-support, which is short for peak support. This is a measure developed to show the impact of different optima in the data set. Once optima are identified, we can compare the support for each relationship across optima and within the set as a whole. This measure serves to identify relationships which have a varied level of support across the different optima. Relationships which are supported by each of the identified optima are likely to be supported by further runs of the phylogenetic inference algorithm that generated the trees. Those which are supported by fewer optima may or may not be supported in further runs and should be investigated further. Understanding the variance between runs of phylogenetic

* Part of the data reported in this section is reprinted with permission from *"A new support Measure to Quantify the Impact of Local Optima in phylogenetic Analyses."* by Grant Brammer, Seung-Jin Sul, and Tiffani L. Williams, in Evolutionary Bioinformatics Online 7 (2011): 159. © Brammer, Sul, and Williams.

inference algorithms can provide users with another level of confidence in their results.

8.1 Optima and convergence

Trees coming from a single analysis provide interesting data sets for comparative study. Given the NP-Hard nature of the phylogenetic inference problem we can not know if we have found the globally optimal trees until we have visited and scored every tree in tree space. Full exploration of the solution space is impractical for all but the smallest datasets. Search algorithms navigate through tree space returning the best scoring trees they find. Algorithms are not immune to be stuck in a single optimum and may explore multiple optima as part of a search. When multiple optima are explored, some may be sampled more than others. This is not necessarily because the more sampled optima are better than the others, but because the inference algorithm was stuck there longer. These algorithmic issues can be reflected in the search results.

Since the popular inference algorithms navigate from one tree to the next by mutating a tree's topology, groups of topologically similar good scoring trees can form local optima for the search algorithms. Each optimum can be thought of as a distinct hypothesis of the true phylogeny. This is because the trees within an optimum share similar characteristics, yet they are distinct from the trees in other optima. Detecting the presence of these optima can allow a researcher to better understand how the support for different relationships is manifested in the tree set. Detection and understanding of optima is especially important for Bayesian inference analyses.

Bayesian inference, as implemented in the MrBayes [35] software package, is one of the most popular approaches for reconstructing evolutionary trees. This package uses Markov Chain Monte Carlo (MCMC) sampling which is a very versatile, yet computationally intensive, procedure to produces samples of parameter values from the posterior distribution. Unfortunately, MCMC simulations are not without their problems. It can be difficult to assess whether an MCMC analysis has converged to the correct distribution and the resulting samples can only be said to have come from the true distribution after convergence [74]. For phylogenetic inference, non-convergence implies that the MCMC sampling did not sample from the distribution of the true evolutionary tree, leading to an inaccurate estimate of the true tree. Detecting and understanding the presence of optima in the data set can help researchers make a better informed decision as to whether their analysis has reached convergence.

8.2 Clustering

Clustering refers to the task of placing data into sets such that the data in each set is more similar to the data with in the same set than it is to the data contained in the other sets. Sets created this way are referred to as clusters. Since the placement of data requires optimizing multiple clusters, clustering is a multi-objective optimization problem. Different algorithms define optimal clusters in different ways and use different methods to compute the optimal clusters. Clustering algorithms are dependent on the measures used to define the similarity between data points. Different clustering results can be obtained by changing the optimality criteria.

In general, a cluster is not precisely defined and only has a common theme of being a group of data. However, when applying the concept of clustering to a phylogenetic trees, we can define certain goals for clustering algorithms. We use clustering as a method of estimating distinct groups of topologically similar trees with in a set. The trees in a set may be returned from one or more local optima, but the inference algorithms do not make users aware of potential optima in the data, they only return the set of good scoring trees. Clustering algorithms provide a way to estimate the presence of these optima after the analysis is competed.

8.3 Types of clustering

Two popular clustering algorithms are implemented in TreeHouse. K-means and agglomerative are each different approaches to the clustering problem with there own strengths and weaknesses.

8.3.1 k-Means

k-means is a centroid based clustering algorithm which attempts to partition the trees into k clusters. Each tree is placed into the cluster with the nearest mean based on some distance function selected by the user. Defining the best possible clusters is an NP-Hard optimization problem, but one with well known heuristics which quickly converge to a locally optimal solution.

8.3.2 Agglomerative hierarchical clustering

TreeHouse also contains an implementation of an agglomerative hierarchical clustering algorithm. Hierarchical clustering attempts to build a tree of the data placing similar data points near each other. In the case of trees hierarchical clustering builds a tree of trees with similar trees near one another. Hierarchical clustering can be done in two major ways. The first is a bottom up approach where each data point starts in a cluster by itself. Pairs of similar clusters are merged as the tree hierarchy is formed. This is known as agglomerative clustering and is the method implemented in TreeHouse. The other option is to use a decisive clustering algorithm which is a top down approach to creating a hierarchical clustering. In this method all the data points start in a single cluster. Clusters are split as the hierarchy is formed. Both methods are generally implemented as greedy algorithms. Agglomerative clustering was chosen for inclusion in TreeHouse for its general case $\mathcal{O}(n^3)$ running time. While this is slow for large data sets it is faster than the general case for divisive clustering which runs in $\mathcal{O}(2^n)$ time.

8.3.3 Measuring cluster quality

The average silhouette width is a popular measure for assessing the validity of clusters. When applied to tree space, the silhouette width is a measure of how well matched a tree is to its assigned cluster. The value is computed for each tree in the set and average values are computed for each cluster or across the whole data set.

$$sill(i) = \frac{NeighborDist(i) - OwnDist(i)}{\max\{OwnDist(i), NeighborDist(i)\}} \tag{8.1}$$

The silhouette of a tree i is defined in Equation 8.1. $OwnDist(i)$ is the average distance between the tree i and the rest of the trees in the same cluster as i. This value can be thought of as how well the tree i is matched with its own cluster. Well matched trees will have low $OwnDist(i)$ values. $NeighborDist(i)$ is the average distance between the tree i and all the trees of the closest cluster which it is not assigned. This value is computed by taking the average distance between i and all the trees in a single cluster. This is repeated for each cluster and the minimum value is returned. The cluster with the lowest value is referred to as the neighboring cluster for i and is the cluster other than the one that i is assigned which would be the best fit for i.

The silhouette width reported as a value from -1 to 1. Values close to 1 require that average distance from the other trees in its assigned cluster to be smaller than the average distance to the trees in the neighboring cluster. Trees with values close to 1 are considered to be well placed. Values close to -1 require that the trees in the neighboring cluster are less dissimilar to i that the trees in its assigned cluster. Trees with negative values are considered not to be well placed. Values close to 0 would denote a tree that is on the edge of two clusters.

The average $sill(i)$ value for a whole cluster of trees is a measure of how tightly grouped and distinct the cluster is. The average $sill(i)$ for a whole data set can be taken as how well the data has been clustered.

While this measure was defined for validating the result of clustering algorithms, it can be as a measure of quality for groups of trees no matter how the groups are defined. We can think of it as answering the question, "If a clustering algorithm had created this grouping of trees would the algorithm have been considered to do a good job?"

$\frac{ISim_{ave}}{ESim_{ave}}$ is similar to the Average Silhouette width. It is the average distance from a tree to the other trees in its own cluster over the average distance for a tree to the trees in every other cluster. The major difference in the calculations is that the silhouette width only considers the next closest cluster in scoring a data point and the $\frac{ISim_{ave}}{ESim_{ave}}$ considers all other clusters for each data point. Due to each measure's use of data points outside of the cluster as part of their calculations, neither measure is effective in directly determining if a single cluster is the best fit for the data.

8.4 Estimating optima with clustering

The first step in measuring the impact of optima on a set of trees is to identify the presence of optima. In our work on the development of p-support, which attempts to quantify the impact of optima on the summarization of a dataset, we put forth a method called PeakMapper. PeakMapper estimated optima, or peaks, but relied on many outside packages for its computations. The same general method for estimating optima in a set of trees is available in TreeHouse, though TreeHouse provides the user with more options. TreeHouse provides two different clustering algorithms and multiple methods for examining cluster quality. The original experiments with PeakMapper relied on CLUTO [43] for its clustering functions and the $\frac{ISim_{ave}}{ESim_{ave}}$ as a measure of cluster quality. The clustering algorithms perform similarly and $\frac{ISim_{ave}}{ESim_{ave}}$ is very similar to the average silhouette width which is also implemented in TreeHouse. One method of estimating optima is presented here and the results from this analysis can be used as the input to algorithms which analyze sets of trees, such as p-support.

The general process for estimating the number of optima in a set of trees is presented here in four steps.

Step 1: Given a set of t trees, we use a k-means algorithm to produce a clustering. The default distance measure in TreeHouse is the Robinson-Foulds distance but other distance measures can be used and may produce different results. Since the true number of clusters which best fit the data is unknown, we test a range of k values, and evaluate the fit of each. Fitness of a clustering in TreeHouse can be measured in terms of the average silhouette width.

Step 2: For clustering quality measures such as the silhouette width or the $\frac{ISim_{ave}}{ESim_{ave}}$, we cannot generate a fitness score for situations where $k = 1$ best describes the data. This is because both measures rely on the distance between a tree and the trees in a neighboring cluster. With a single cluster there is no neighboring cluster to use in the computation. Instead, we check whether any of the clusterings fit the data. If not, we reject the hypothesis that there are multiple optima in favor of a single optimum being the best fit for the data.

Step 3: Once we have computed the clusterings of our data, we select the number of clusters that maximizes the similarity among the trees of interest while minimizing the total number of clusters. In selecting a p value which best fits the data, we use the elbow criteria. The selection criteria favors choosing a p so that adding additional optima does not add sufficient information. In other words, we choose a p value such that increasing it does not increase the silhouette ratio significantly.

Step 4: We can then use multidimensional-scaling (MDS) plots to see the relative positions of the trees within their local optima. TreeHouse can call dredviz [100, 99] which is a freely available open source for reducing the dimensionality of large data sets so that they can be visualized. MDS computes a position for each tree in a 2 dimensional plane which best fits their higher dimensional relationships. Mapping the high dimensional relationships into the best fitting points on a plane is a NP-hard optimization problem. While imperfect MDS is provides powerful visual feedback about the relationship between trees and acts as a another check to the validity of a clustering. Aside from this study, other systematists have used MDS in phylogenetics [31]. However, they do not use MDS in the context of optima analysis and visualization as explored here.

8.4.1 Experimental results

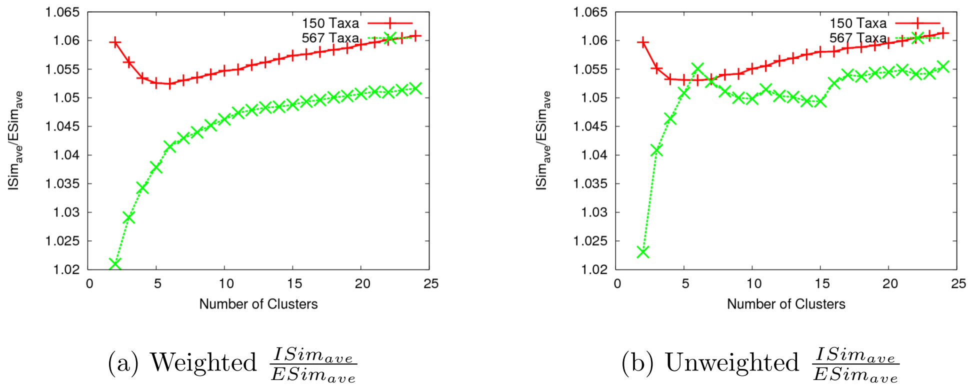


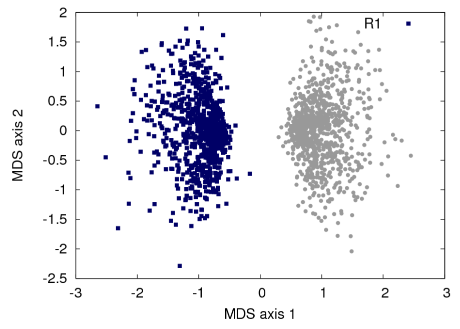
Figure 8.1: Selecting the appropriate number of clusters, k. Larger $\frac{ISim_{ave}}{ESim_{ave}}$ values are preferred since they indicate a better clustering of the data.

Figure 8.1 shows the weighted and unweighted $\frac{ISim_{ave}}{ESim_{ave}}$ values for different clusterings of the 150 taxa, 20,000 tree set. The weighted value takes the size of a cluster into the calculation of the $\frac{ISim_{ave}}{ESim_{ave}}$ so that smaller clusters contribute less to the total score. This analysis considers both the weighted and unweighted values, though in many cases one value may be sufficient. Applying the elbow criteria to both the weighted and unweighted data suggests that the optimal number of clusters is at $k = 2$. Thus, p= 2 since the data set contains two local optima, $\mathcal{P}0$ and $\mathcal{P}1$. What is the composition of the two local optima in this data set? Table 8.1 and Figure 8.2 shows that both optima are composed of half of the 20,000 total trees, and each optimum is composed of trees from a single run. Optimum $\mathcal{P}0$ is composed of trees from run $R1$ from the Bayesian analysis. Optimum $\mathcal{P}1$ consists of trees from run $R0$. There is no overlap of trees between the two optima as each run is contained within a single local optimum.

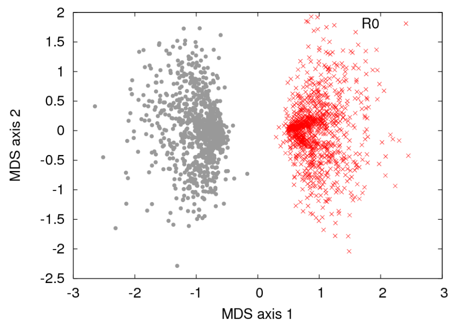
	size		**resolution rate**		
optima	*trees*	%	*majority*	*strict*	**runs** (%)
$\mathcal{P}0$	10,000	50%	90.5%	34.7%	R1 (100%)
$\mathcal{P}1$	10,000	50%	89.1%	37.4%	R0 (100%)

Table 8.1: Detailed information for the two local optima found for the 150 taxa trees. For each optimum, we list the number of trees, resolution rates of the majority and strict consensus trees, and run labels of the trees. The resolution rate reported in this table is a measure of how resolved the consensus tree is. If the consensus tree was completely binary we would have a value of 100% whereas if it was a star phylogeny it would have a value of 0%

Even though the 567 taxa data set is more demanding in terms of having more trees and runs than the 150 taxa data set, we approach it with the same process to estimate the number of optima in the data set. Examining Figure 8.1 with the



(a) Optimum $\mathcal{P}0$



(b) Optimum $\mathcal{P}1$

Figure 8.2: *150 taxa, $p = 2$:* Each plot shows a single optimum with the trees in the optimum colored to represent the Bayesian runs they came from. The MDS values are computed as a Euclidean embedding of the data points. The r value for this MDS analysis is 0.79, where $r = correlation(original, reconstruction)$.

elbow criteria in mind suggests that $k =$ 6 is the optimal clustering for the data set. Since $k =$ 6 is the clear peak for the unweighted data and is with in the elbow for the weighted data, we will proceed with it as our choice and select the $p =$ 6 as the number of local optima for this phylogenetic analysis.

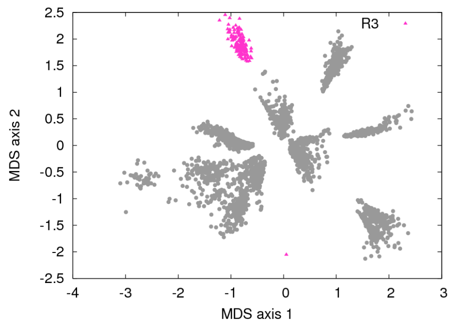
We now explore the make up of the optima and ask "What are the traits of the six optima in the data set?" Table 8.2 provides statistics regarding the composition of the six optima for the 567 taxa trees. The trees in runs $R0$, $R6$, and $R11$ are each fully contained in single local optimum, $\mathcal{P}3$, $\mathcal{P}1$, and $\mathcal{P}2$ respectively. These trees are placed in these optima alone with no other trees from other runs. Hence, we can say that the trees from each run are contained in their own optimum with no mixing or overlap with trees from other runs. These runs settled into distinct local optima in tree space.

Alternatively, optima $\mathcal{P}4$ and $\mathcal{P}5$ contain trees from multiple runs. Optimum $\mathcal{P}4$ contains trees from runs $R1$, $R5$, $R7$, and $R8$. These runs appear wholly in $\mathcal{P}4$, and

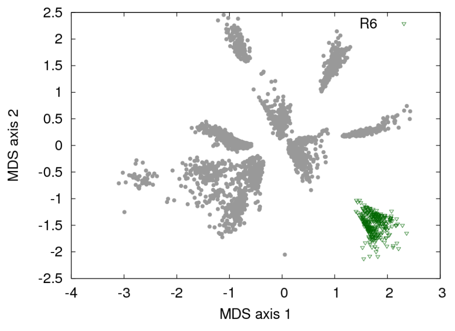
optima	**size**		**resolution rate**		**runs** (%)
	trees	*%*	*majority*	*strict*	
$\mathcal{P}0$	1,537	4.6%	94.2%	68.4%	$R3$ (100.0%)
$\mathcal{P}1$	2,986	9.0%	95.6%	64.7%	$R6$ (100.0%)
$\mathcal{P}2$	2,164	6.5%	95.6%	66.8%	$R11$ (100.0%)
$\mathcal{P}3$	3,177	9.5%	95.4%	65.1%	$R0$ (100.0%)
$\mathcal{P}4$	11,312	34.0%	94.0%	61.0%	$R1$ (28.1%) $R5$ (26.4%) $R7$ (26.4%) $R8$ (19.1%)
$\mathcal{P}5$	12,130	36.4%	92.4%	58.3%	$R2$ (26.2%) $R3$ (13.5%) $R4$ (24.6%) $R9$ (17.8%) $R10$ (17.8%)

Table 8.2: Detailed information for the six local optima found by our analysis approach on the 567 taxa trees. For each optimum, we list the number of trees, resolution rates of the majority and strict consensus trees, and run labels of the trees.

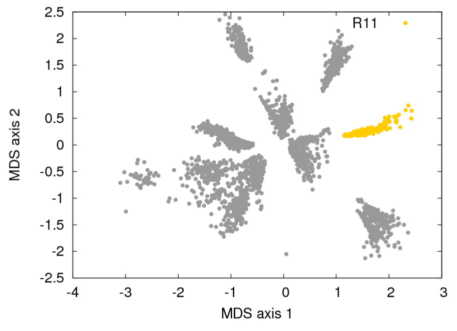
in no other optima. We can say that these four runs have stabilized to the same optimum but do not overlap with the runs in any other optima. Local optimum $\mathcal{P}5$ is very similar to optimum $\mathcal{P}4$. It is mainly composed of runs that wholly are contained with in the optimum. Runs $R2$, $R4$, $R9$ and $R10$ are all contained in this optimum. The one exception is the placement of run $R3$. This run is split between two optima. About half the run appears in optimum $\mathcal{P}4$ with four other runs and the other half of run $R3$ appears in optimum $\mathcal{P}0$ by itself. This shows that run $R3$ is split between two different optima. It is the only run in either of our data sets to exhibit this behavior. This behavior may be the result of the phylogenetic search settling into one optimum for the first part of the search only to find a better optimum as the search progressed. The information in Table 8.2 is represented visually in Figure 8.3.



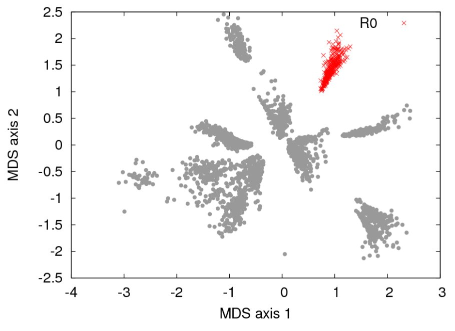
(a) Optimum $\mathcal{P}0$



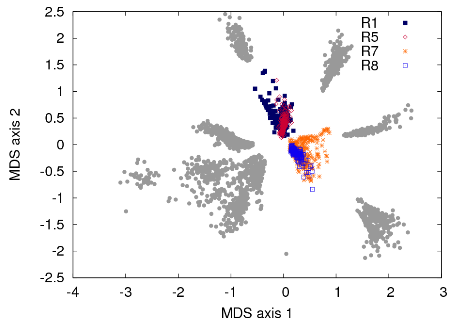
(b) Optimum $\mathcal{P}1$



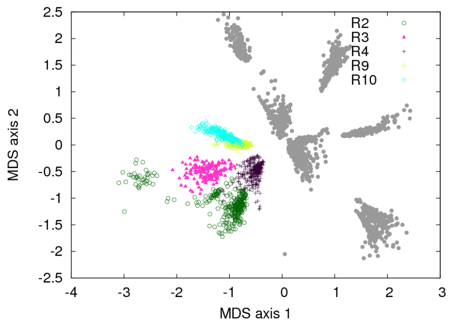
(c) Optimum $\mathcal{P}2$



(d) Optimum $\mathcal{P}3$



(e) Optimum $\mathcal{P}4$



(f) Optimum $\mathcal{P}5$

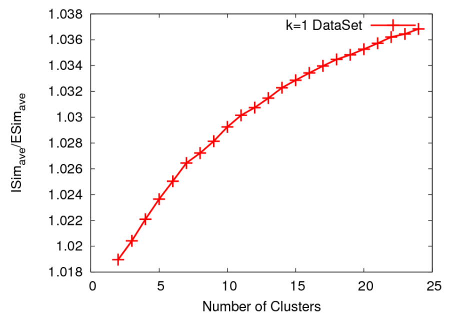
Figure 8.3: *567 taxa, $p = 6$:* Each plot shows a single optima with the trees in the optima colored to represent the Bayesian runs they came from. The MDS values are computed as a Euclidean embedding of the data points. The r value for this MDS analysis is 0.84, where $r = correlation(original, reconstruction)$.

In this data set our analysis shows six local optima each distinct from one another. The only mixing in terms of runs appearing in multiple optima occurs with $R3$ appearing in two optima. Knowledge of these trends in the data have the potential to inform the process of summarizing of the trees. For instance, since the behavior exhibited by run $R3$ could be explained that it found one optimum early in the search only to abandon it later for another optimum, it may be useful to remove the section of $R3$ contained in the lesser scoring of the two optima from the summarization. This same idea could be applied to whole sets of optima. Given that we have six optima and know which trees make up each optima, it becomes possible to select the optima with the best overall likelihood scores and set aside the trees representing the less likely optima.

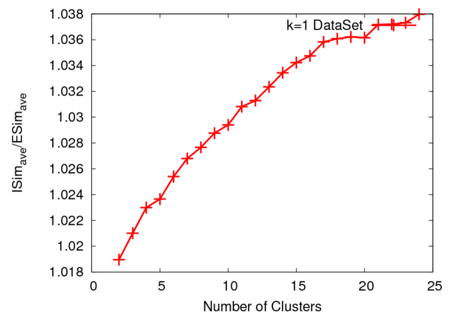
8.4.2 Detecting single optimum tree collections

We have shown the effectiveness of our algorithm in detecting when a data set has stabilized to multiple optima instead of a single optimum. Since neither data set we analyzed contained only a single optimum, we chose to create such a data set in order show how our algorithm would perform given this case. To represent this case we used a single run from our 150 taxa data set to create a 10,000 tree data set with a single optimum.

We begin by clustering our data set using $k = 2$ through $k = 24$. We then examine the $\frac{ISim_{ave}}{ESim_{ave}}$ ratio in Figure 8.4. Applying the elbow criteria can be a little tricky in this case. There is no obvious place where the gains in the $\frac{ISim_{ave}}{ESim_{ave}}$ ratio taper off. There is a fairly steady increase in the $\frac{ISim_{ave}}{ESim_{ave}}$ ratio as the k values increase. This is a strong sign that the most appropriate clustering is actually $k = 1$. As we increase k to its maximum value of n (number of trees) we expect the $\frac{ISim_{ave}}{ESim_{ave}}$ ratio to rise. We are looking for a peak in these values before it it becomes a continual



(a) Weighted $\frac{ISim_{ave}}{ESim_{ave}}$



(b) Unweighted $\frac{ISim_{ave}}{ESim_{ave}}$

Figure 8.4: Selecting the appropriate number of clusters, k, when the optimal number of clusters is 1. Larger $\frac{ISim_{ave}}{ESim_{ave}}$ values are preferred since they indicate a better clustering of the data.

rise. Since this isn't found, we assume a k value of one and therefore a p value of one.

8.4.3 Conclusions

Using our approach, we analyzed two published Bayesian studies. Data set #1 covers 150 taxa of desert algae and green plants [48] and Data set #2 covers 567 taxa of angiosperms [85]. The 150 taxa data set consists of 20,000 trees from two runs of the MrBayes phylogenetic heuristic. The 567 taxa data set contained 33,306 trees from 12 Bayesian runs. Both of these tree collections have high majority consensus resolution rates. Our approach shows that both tree sets contain multiple local optima—there are two and six local optima found for the 150 and 567 taxa data sets, respectively. Hence, high consensus resolution rates do not exclude the possibility of a tree set containing multiple optima. These data sets present two interesting cases: the number of trees in the optimum in the 150 taxa data set are of equal size while they are disproportional in the 567 taxa data set. These cases show how the distribution of trees across optima can impact the resulting majority consensus tree. These data sets make the case for a support measure which can quantify the impact of optima on the branches of a consensus tree. p-support is intended to provide this information.

It has been assumed that when multiple Bayesian analyses converge that there is a single optimum present in the data set. We have shown that this is an assumption that should be investigated further. Not only is it possible for multiple optima to be present it is possible for a single run of a search algorithm to contain multiple local optima. We have introduced tools to help detect and quantify the impact of local optima on the bipartitions which are presented in the final tree.

8.5 Peak support

8.5.1 Motivation

With our work exploring convergence and distinguishing bipartitions, we identified multiple optima in some tree sets. The trees in an optimum share some common phylogenetic information with the other trees in the optimum. However, each optimum clusters independently and is separated by distinguishing bipartitions from the other optima. Each optimum contains its own hypothesis of the true tree. Since phylogenetic search algorithms attempt to find the best tree in tree space and not evenly sample each optima, the search may over sample some optima and under sample others. Computing a majority consensus tree of such a data set can lead to some hypotheses being given more weight than others, not because they are more likely but simply because of how the data was sampled. To address this issue we have developed p-support as a way of showing the impact of these distinct hypothesis on the final consensus tree.

8.5.2 Definition of p-support

We define the p-support of a bipartition as the percentage of p local optima with majority support for that bipartition. A p-support value of 100% means that a bipartition was supported by each optimum whereas 0% implies that the bipartition was not strongly supported by any of the p optima. p-support can be viewed as a measure of precision at the local optimum level, much the same way that bootstrap [21] and taxa jackknifing [83] support are measures of precision at the character and taxa level. High p-support values signal that a bipartition is in high agreement across the local optima and, therefore, is less likely to be overturned by additional analysis. Similarly to other common support measures, p-support can be useful in identifying the areas of a tree that may benefit the most from additional data and analysis. In this way, support measures are a useful tool in illuminating new problems and hypotheses [26].

The most critical feature to the p-support measure is the identification of the p local optima of an analysis. These local optima serve as the input to the p-support calculation. We have developed PeakMapper as well as a more streamlined process available in TreeHouse to determine how many local optima are contained in a data set, as well as which trees are contained in each optimum. While we present a technique to use clustering to identify the optima among the trees, p-support is independent of how the optima are determined. Any method that identifies local optima can be used with our p-support measure. For instance, if tree islands [51] were detected and labeled in a data set, that information could be used to compute p-support. Once the optima are identified, p-support can be computed for the tree collection and annotated on majority and strict consensus trees. These support values can be viewed in standard tree viewing packages such as FigTree or with

TreeHouse's display tree function.

p-support was defined and tested outside of the TreeHouse framework. It has since been implemented with in TreeHouse. The experiments shown were originally done with the standalone implementation of *p*-support using PeakMapper. They have since been verified with the new TreeHouse implementation. Since *p*-support values are deterministic, the results for the new and original analysis are the same.

8.5.3 Comparison to common support measures

Bremer support, also known as the decay index, support index, or simply SI [9] measures how many steps from the most parsimonious trees it takes to lose a branch in the consensus of the near-most-parsimonious trees. A branch in one of the most parsimonious trees is strongly supported if it is also contained in the near-most-parsimonious trees. There are similarities in the intuition behind Bremer support and *p*-support. Both methods consider the prevalence of a bipartition across sets of trees. Where Bremer support creates subsets of trees by iteratively relaxing the threshold for near-most-parsimonious trees to be included into the consensus, *p*-support considers local optima. Where Bremer support seeks to compute the the point at which the bipartition stops appearing in the consensus of the subset of trees, *p*-support computes the prevalence of each bipartition in each optimum. In both cases highly supported bipartitions are highly corroborated in the data set. While Bremer support is effective only in parsimony analyses, *p*-support is not limited by the method which the trees are computed. *p*-support can be used in any analysis which contains multiple optima.

Bootstrap support [21] is computed by running a set of analyses with the input sequence alignment resampled such that some characters are include twice or more and others are not included at all. This simulates the effect that reweighting or revising the data might have on the output trees. Hence, the rate that a bipartition appears in the resulting trees is a measure of how robust a bipartition is to changes in the sequence alignment. Bootstrap support is similar to taxa jackknifing [46, 75] which samples the taxa set to generate input data for a set of phylogenetic analyses. The resulting trees are used to compute a consensus tree to identify areas of disagreement among the trees. This is a measure of stability in regards to the deletion of taxa.

Each of these measures defines the support of a bipartition based on its stability under different methods of perturbation of the input sequence data. These methods are very complementary to p-support which is the only support measure of corroboration across optima in the data set. In fact, there is no strict guarantee that the trees generated from a bootstrap or jackknife analysis fall into a single optimum themselves. Thus, measuring the p-support of the trees resulting from a bootstrap or jackknife analysis may provide further information.

8.5.4 Computing and visualizing p-support.

Once the optima in a set of trees have been identified, we can compute the p-support value for each clade in our data set based on those optima. The p-support value of a clade is the percentage of optima which contain majority support for the clade. In order to visualize the p-support values for each unique clade in the data set, we developed the p-map. Our visualization technique compares the range of p-support values on the y-axis to the percentage of total support on the x-axis. To increase readability, jitter is applied along the y-axis only, allowing overlapping points to form into lanes. The position on the x-axis is absolute meaning that any point on the right side of the line marking 50% support is a clade that would appear in the majority consensus trees. The shading of each point in a p-map represents the standard deviation in support for each clade between clusters. Points shaded in blue are equally supported across clusters and points shaded in red are highly supported in some clusters and barely supported in others.

8.5.5 Experimental results

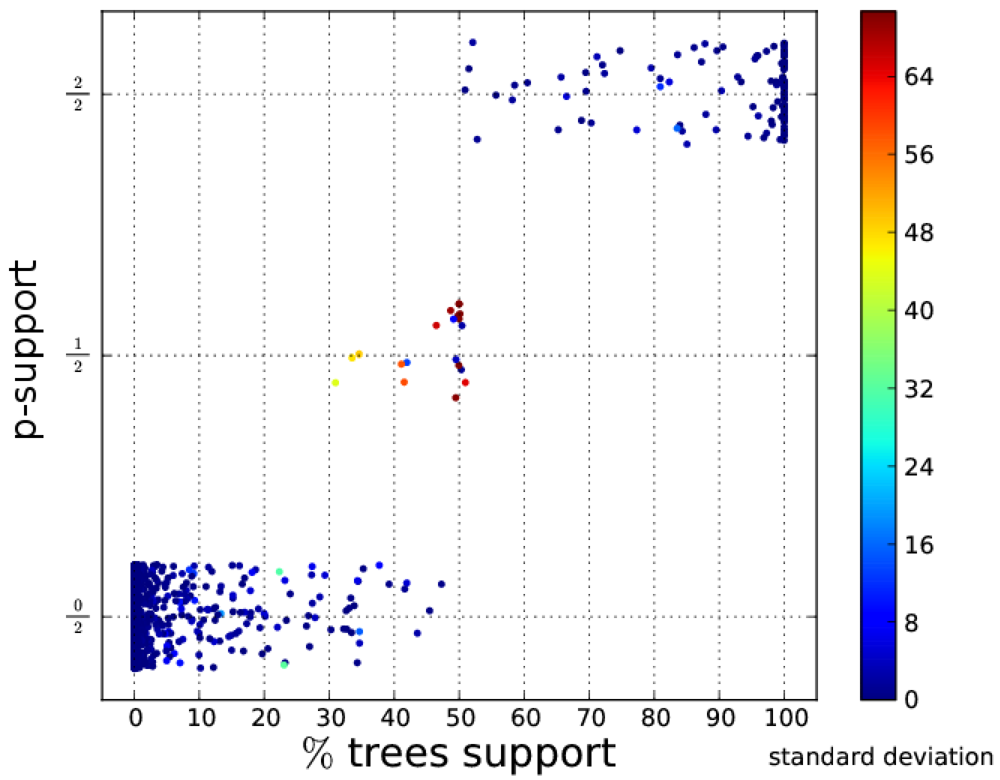


Figure 8.5: A p-map, p-support values plotted against the percent of trees containing the bipartition for the 150 taxa data set with a p value of 2. Points greater than 50% on the x-axis would appear in a majority consensus tree. The points are shaded to reflect the standard deviation in the support values from the different optima. Dark blue points mean the bipartition was equally supported among the optima whereas red points mean there was a large difference in the amount of support for a bipartition among the optima.

Using the the optima estimations in Section 8.4 for Data set #1 and Data set #2 we compute the p-support for the clades in each tree set.

Summarizing the of 20,000 trees of Data set #1 as a consensus tree without acknowledging the presence of the two local optima ignores the distinct competing hypotheses that exist in this data set. We represent the influence of these hypotheses by annotating the resulting consensus tree with the p-support values of each branch in the consensus. For example, in a majority consensus tree, our annotation illuminates majority bipartitions that are supported by a subset of the local optima. Figure 8.5 shows that there are three bipartitions that would appear in the majority consensus tree but are only supported by one of the two local optima. These bipartitions have a high standard deviation of p-support meaning that they are supported by one optimum much more than the other. Since there is disagreement among the optima, these bipartitions more likely to be overturned by further runs of the search heuristic than those agreed on by all of the optima.

Figure 8.6 shows the p-support values plotted against the percentage of trees containing each bipartition for the clades in Data set #2. Notice that there are a number of bipartitions which would appear in the majority consensus of the set but are supported by only a subset of the optima found in the search. For instance, there are seven bipartitions which appear in the majority consensus tree, but they are only supported by 50% of the optima. There is even a bipartition which is only supported by 2 of the 6 optima yet it has over 50% majority support and therefore appears on the majority consensus tree. Due to size disparity between optima, there are some bipartitions supported by only half the optima but are still able to achieve over 70% majority support. These bipartitions are also interesting due to the high standard deviation in p-support. Some optima heavily support those bipartitions while other optima barely support them if at all. The bipartitions with low p-support are the

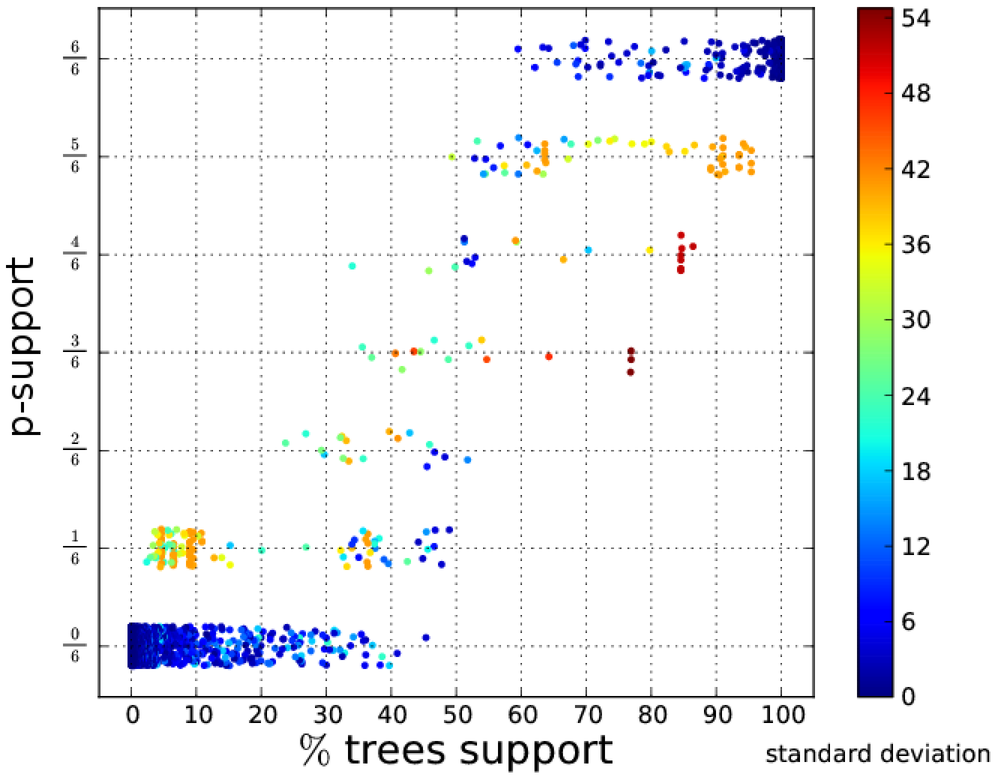


Figure 8.6: p-support values plotted against the percent of trees containing the bipartition for the 567 taxa data set with a p value of 6. Points greater than 50% on the x-axis would appear in a majority consensus tree. The points are shaded to reflect the standard deviation in the support values from the different optima. Dark blue points mean the bipartition was equally supported among the optima whereas red points mean there was a large difference in the amount of support for a bipartition among the optima.

ones most likely to be effected by further analysis or new data. Bipartitions that are common to all 6 local optima are likely to appear in future runs of the heuristic search algorithm while bipartitions supported by a subset of optima are less likely to be present in a future run. To this end it may be appropriate to collapse bipartitions with low p-support.

8.5.6 Conclusions

Overall, our work presents systematists with a new measure called p-support for quantifying the robustness of inferred relationships in an evolutionary tree. We hope that p-support can provide researchers and the community at large with more information about the results of phylogenetic analyses—especially in regards to which regions of the tree may benefit most from further investigation

The p-support analysis also serves as an example of the types of analysis that can be conducted with in TreeHouse and the power of having multiple algorithms availble with in a single framework. Peak support relies on clustering to determine optima, and consensus methods to identify which relationships are supported by the optima. Clustering relies on distance measures for the actual grouping of trees as well as the measurement of clustering quality. Having all of these functions with in the same framework makes the analysis much easier for the user.

9. REPRODUCIBILTY

9.1 Validation of TreeHouse

A major component of the scientific method is the validation of the research by the community. Validation requires that the community is given access to the data and tools necessary for to repeat the experiments. Funding organizations and scientific journals have put policies into place that require researchers to make their data and methods available to other researchers. Within the field of phylogenetics, it is often required that the sequence data used to infer the phylogeny is published in GenBank [3] and the inferred phylogeny is published on TreeBASE. However, much of the information required to repeat the experiment and turn the sequences into a tree are lost.

The computational nature of the methods can cause other issues for reproducible science. Analyses are often conducted with software that is neither multi-platform nor open source. When software is available, there is still a question is to whether it will run the same on two different machines. Operating systems, hardware, software dependencies, and runtime environment differences can lead to different results. Furthermore, the version of the software can make a big difference in the results of an analysis. Once software is installed and working as expected, there becomes a question as to what settings were used to conduct the analysis. Much of the software used in computational biology allows the user to tweak a variety of settings to adjust the analysis to better fit the user's data or the type of analysis being done. All of the settings are a required component of checking and reproducing an analysis. However, much of this information is not readily available to the scientific community.

As a developer of piece of software which hosts algorithms for computational analysis, I'm aware of the reproducibility issue on two levels. The first is the validation of TreeHouse and its algorithms. TreeHouse is being made available to the community as an open source project. This allows others to validate that the algorithms contained in the system work as described.

TreeHouse uses features from open source libraries such as Boost, ncurses, and readline. The availability of these open source libraries is a boon to those that write software. They go a long way in preventing the need to reinvent the wheel for many common functions. However, for every library that the programmer uses to get their software running, there is one more library that the end user has to install. TreeHouse also uses some features in the C++11 standard. These features are implemented in some version of some compilers and not in others. These dependencies form the environment in which TreeHouse was developed and tested and can often be a barrier to installing a piece of research software.

In order to lower the barrier of entry to TreeHouse, it is being made available as part of a virtual machine image that captures the development environment for the project. This allows users to download the machine image and load a runtime environment which contains an installed version of TreeHouse, the source code, all of its dependencies, examples, and documentation. This method allows users to test drive the software before deciding if it is worth while to install it on their own systems. It also provides a template for a known working installation environment. As compilers and libraries continue to develop, features may change which could break or improve functionality within TreeHouse. The virtual image provides a snap shot of how it worked when it was published.

9.2 Reproducing analyses with TreeHouse

In the development of TreeHouse, I was also aware that the intended users are scientists running experiments. So, while I wanted to make sure that the work I had done with the algorithms in TreeHouse was reproducible, I also wanted to make sure that any work done using TreeHouse for analysis would also be reproducible. TreeHouse offers two modes of interaction, the first is an interactive command line and the second is the execution of batch scripts. TreeHouse provides a logging feature which documents the user input in the interactive mode as a batch script so that complicated series of commands can be easily repeated.

For instance, we can use TreeHouse to address whether two runs of a phylogenetic inference algorithm converged to the same place in tree space. One way that we might conduct this experiment is to cluster the trees then compare the way that the clustering algorithm grouped the trees with a grouping based on the runs that generated the trees using the rand index. The results of this experiment might appear in a paper but due to perceived importance and space limits certain details may be omitted. However, with TreeHouse the whole experiment can be distributed so that others can follow the exact steps. The example analysis might take the form of the five commands shown in Listing 9.1

Listing 9.1: Rand index comparison

```
run1 = {0..9999}
run2 = {10000..19999}
runs = [run1,run2]
clustering = kmeans({0..19999},2,RF)
rand_index(runs,clustering)
```

With the included batch script and the TreeZip file, any one can run the analysis themselves. More importantly, it takes away any ambiguity about how the analysis was conducted. All of the variables are defined for the users and it is clear that the clustering algorithm used was k-means with a k value of 2, and the Robinson-Foulds distance was used as the distance measure for the clustering algorithm. This not only resolves any confusion about how the analysis was conducted but provides a platform for further analysis. Given the features of TreeHouse, it would be easy to see if the results of a agglomerative clustering algorithm matched those of the k-means clustering algorithm or take the analysis a step further and determine if there are any distinguishing bipartitions between the runs.

TreeZip [55] has relieved many memory concerns with storing large sets of trees. Space constraints are should not prevent researchers from storing or making public the sets of trees used in their analyses. I hope that the logging features and virtual machine distribution of TreeHouse will allow and encourage researchers to make more of their data available to the community.

10. CONCLUSIONS AND FUTURE WORK

10.1 Conclusions

The goal of this research has been to provide algorithms which allow biologists to better understand their data sets. What began as a search engine for phylogenetic trees has become a platform for information discovery and exploration. In TreeHouse, a user can find more than just the trees that match a query but a range of information their trees. Whether it's questions about how far two taxa are from one another in a single tree or if there are patterns or anomalies in whole sets, TreeHouse has algorithms to help explore those questions.

We have shown that the search algorithms in TreeHouse are fast and effective for both taxa homogeneous and taxa heterogeneous sets of trees. We have also shown the search power of relationships defined by a smaller number of taxa. These edge based searches provide a united way to consider quartets, K-tets, and bipartitions in either taxa homogeneous or heterogeneous trees. Furthermore, the development of a model of querying led to the development of the idea of K-tets as the a way to unify edge based relationships. Our work with K-tets, as a fundamental unit of search, impacted the way we think of distance measures and freed us from the idea that building blocks of trees where either bipartitions or quartets.

The analysis algorithms in the TreeHouse have also lead to novel developments. The work with decision trees demonstrated that sets of trees thought to have converged to a single place in tree space may not have reached convergence. This led to the development of the idea of distinguishing bipartitions and the suggestion that the previous methods for detecting convergence, which do not consider tree topology, were insufficient. This work lead to applying clustering algorithms to sets of trees to estimate local optima in the tree sets that were previously undetected. We have shown that optima are detectable and presented p-support as a measure to quantify the impact of those optima on the resulting tree.

TreeHouse has also lead to the development of other novel analysis tools. Computing consensus trees from the bipartition table led us to implement methods for detecting which trees in a set could or couldn't appear in the same tree. This computation formed the basis for building a new type of consensus tree which is build from the least conflicting bipartitions. The adaptation of the data structures and algorithms to handle heterogeneous taxa sets to facilitate searching inspired the idea of taxa masking as a way to apply familiar analyses to data sets which they would not otherwise be applicable.

10.2 Future work

TreeHouse has evolved from a phylogenetic search engine into a platform for exploring trees. It seems that every new analysis algorithm poses as many new questions as they solve. While we know enough about phylogenetic trees to produce and publish them, it seems at times like we have barely scratched the surface of potential understanding. To this end, there is an ever growing list of problems and questions that TreeHouse may provide solutions for in the future. A few future directions for project have been selected to give the reader a better idea of the TreeHouse's potential as a platform for phylogenetic analysis.

TreeHouse currently uses clustering and decision trees to find patterns in a set of trees. These tools have been used to address the issue of convergence and detect if there are multiple optima in a data set. With some work these methods may also be valuable for determining the burn in of a phylogenetic inference. Burn in is a term used to refer to the samples from a Bayesian analysis which occur before the analysis reaches stability. These trees are discarded as they are not part of the estimated distribution. Since determining convergence is a difficult problem, determining the cut off between the burn in and the posterior distribution can also be difficult. The same sorts of analysis tools which we have used to detect an patterns in amongst trees may also be useful for determining what can be safely discarded as burn in.

TreeHouse introduces the ability to query a set of trees based on relationships. This work on search algorithms has taken an approach which presents bipartitions and quartets as the two end points on the scale of all possible edge based relationships. Bipartitions and quartets have been explored and are fairly well understood, but the relationships which reside between bipartitions and quartets, the K-tets, are not well understood. TreeHouse can already compute matches between K-tets and trees as part of the search algorithms and it is only steps away from allowing K-tets to be used in other contexts. K-tets could be used as the basis of distance measure or as the building blocks in consensus algorithms. They are more robust to conflicting taxa than bipartitions but may be faster to use, at least in some cases, than quartets. These unexplored relationships could be powerful tools in future analysis and TreeHouse is an good position to begin that exploration.

The development of TreeHouse has been guided by the idea that better understanding of the data that goes into a published tree will lead to better published trees. TreeHouse has focused mainly on the topological aspects of trees, but future development may expand with analysis algorithms that utilize the score of a tree based on a specified model of evolution. Bringing in this aspect would could open new questions and lead to the development of new algorithms. It would also require TreeHouse to handle different data types. The ability to score trees based on a model requires that TreeHouse be aware of the model of evolutions and the sequence alignment. This would mark a major shift in development for the project but could provide new an important avenues of research.

REFERENCES

[1] Anne Margrethe Audelin, Jan Gerstoft, Niels Obel, Lars Mathiesen, Alex Laursen, Court Pedersen, Henrik Nielsen, Janne Jensen, Lars Nielsen, Claus Nielsen, et al. Molecular phylogenetics of transmitted drug resistance in newly diagnosed hiv type 1 individuals in denmark, a nation-wide study. *AIDS Research and Human Retroviruses*, 27(12):1283–1290, 2011.

[2] Adrian C. Barbrook, Christopher J. Howe, Norman Blake, and Peter Robinson. The phylogeny of the Canterbury Tales. *Nature*, 394:839, 1998.

[3] Dennis A. Benson, Ilene Karsch-Mizrachi, David J. Lipman, James Ostell, Barbara A. Rapp, and David L Wheeler. Genbank. *Nucleic Acids Research*, 28(1):15–18, 2000.

[4] Olaf Bininda-Emonds. Ratchet implementation in PAUP*4.0b10, 2003. available from http://www.tierzucht.tum.de:8080/WWW/Homepages/Bininda-Emonds.

[5] Olaf R. P. Bininda-Emonds, Denise M. Decker-Flum, and John L. Gittleman. The utility of chemical signals as phylogenetic characters: an example from the felidae. *Biological Journal of the Linnean Society*, 72(1):1–15, 2001.

[6] Olaf R. P. Bininda-Emonds, John L. Gittleman, and Andy Purvis. Building large trees by combining phylogenetic information: a complete phylogeny of the extant carnivora (mammalia). *Biological Reviews*, 74(2):143–175, 1999.

[7] Christopher J. Birch, Rhonda F. McCaw, Dieter M. Bulach, Peter A. Revill, J. Tom Carter, Jane Tomnay, Beth Hatch, Tracey V. Middleton, Doris Chibo, Michael G. Catton, Jacinta L. Pankhurst, Alan M. Breschkin, Stephen A.

Locarnini, and Scott Bowden. Molecular analysis of human immunodeficiency virus strains associated with a case of criminal transmission of the virus. *The Journal of Infectious Diseases*, 182(3):941–944, 2000.

[8] Nicolas Bortolussi, Eric Durand, Michael Blum, and Olivier François. aptreeshape: statistical analysis of phylogenetic tree shape. *Bioinformatics*, 22(3):363–364, 2006.

[9] K. Bremer. Branch support and tree stability. *Cladistics*, 10(3):295–304, 1994.

[10] David Bryant, John Tsang, Paul Kearney, and Ming Li. Computing the quartet distance between evolutionary trees. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 285–286. Society for Industrial and Applied Mathematics, 2000.

[11] Jon K. Chambers, Lynn E. Macdonald, Henry M. Sarau, Robert S. Ames, Katie Freeman, James J. Foley, Yuan Zhu, Megan M. McLaughlin, Paul Murdock, Lynette McMillan, John Trill, Ann Swift, Nambi Aiyar, Paul Taylor, Lisa Vawter, Sajda Naheed, Philip Szekeres, Guillaume Hervieu, Claire Scott, Jeanette M. Watson, Andrew J. Murphy, Emir Duzic, Christine Klein, Derk J. Bergsma, Shelagh Wilson, and George P. Livi. AG protein-coupled receptor for UDP-glucose. *Journal of Biological Chemistry*, 275(15):10767–10771, 2000.

[12] Duhong Chen, Gordon J. Burleigh, Mukul S. Bansal, and David Fernandez-Baca. PhyloFinder: an intelligent search engine for phylogenetic tree databases. *BMC Evolutionary Biology*, 8:90+, 2008.

[13] Peter J. A. Cock, Tiago Antao, Jeffrey T. Chang, Brad A. Chapman, Cymon J. Cox, Andrew Dalke, Iddo Friedberg, Thomas Hamelryck, Frank Kauff, Bartek Wilczynski, et al. Biopython: freely available python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422–1423, 2009.

[14] D. H. Colless. Review of phylogenetics: the theory and practice of phylogenetic systematics. *Systematic Zoology*, 31(1):100–104, 1982.

[15] Ralph W. Crosby and Tiffani L. Williams. A fast algorithm for computing the quartet distance for large sets of evolutionary trees. In *Bioinformatics Research and Applications*, pages 60–71. Springer Berlin, Heidelberg, 2012.

[16] Brian W. Davis, Gang Li, and William J. Murphy. Supermatrix and species tree methods resolve phylogenetic relationships within the big cats, panthera (carnivora: Felidae). *Molecular Phylogenetics and Evolution*, 56(1):64 – 76, 2010.

[17] Akshay Deepak. *SearchTree: Mining robust phylogenetic trees.* PhD thesis, Iowa State University, Ames, 2010.

[18] Armand Denis. *Cats of the world.* World wildlife series. no. 1. Houghton Mifflin, Boston, 1964.

[19] George F. Estabrook, F. R. McMorris, and Christopher A. Meacham. Comparison of undirected phylogenetic trees based on subtrees of four evolutionary units. *Systematic Biology*, 34(2):193–200, 1985.

[20] Wikimedia User Falense. Lioness at okonjima lodge, namibia. Availible from http://en.wikipedia.org/wiki/File:Okonjima_Lioness.jpg, September 2006. This Digital image is availible from Wikimedia Commons and is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license.

[21] Joseph Felsenstein. Confidence limits on phylogenies: An approach using the bootstrap. *Evolution*, 39(4):pp. 783–791, 1985.

[22] Joseph Felsenstein. *Inferring Phylogenies.* Sinauer Associates, Hoboken, New Jersey, 2003.

[23] Joseph Felsenstein. The Newick tree format. Internet Website, url: http://evolution.genetics.washington.edu/phylip/newicktree.html, last accessed, March 2014.

[24] Pablo A. Goloboff, James S. Farris, and Kevin C. Nixon. TNT, a free program for phylogenetic analysis. *Cladistics*, 24(5):774–786, 2008.

[25] Naohisa Goto, Pjotr Prins, Mitsuteru Nakao, Raoul Bonnal, Jan Aerts, and Toshiaki Katayama. Bioruby: bioinformatics software for the ruby programming language. *Bioinformatics*, 26(20):2617–2619, 2010.

[26] T. Grant and A. G. Kluge. Clade support measures and their adequacy. *Cladistics*, 24(6):1051–1064, 2008.

[27] E. Haeckel. *Generelle Morphologie der Organismen: Allgemeine grundzge der organischen formen-wissenschaft, mechanisch begrndet durch die von Charles Darwin reformirte Descendenz-Theorie.* Verlag von Georg Reimer, Berlin., 1866.

[28] M. H. Hast. The larynx of roaring and non-roaring cats. *Journal of Anatomy*, 163:117, 1989.

[29] H. Hemmer. The evolutionary systematics of living felidae: present status and current problems. *Carnivore*, 1:7179, 1978.

[30] S. J. Herrington. *Phylogenetic relationships of the wild cats of the world.* PhD thesis, University of Kansas, Lawrence, 1986.

[31] David M. Hillis, Tracy A. Heath, and Katherine St. John. Analysis and visualization of tree space. *Systematic Biology*, 54(3):471–482, 2005.

[32] Eddie C. Holmes, Sean Nee, Andrew Rambaut, Geoff P. Garnett, and Paul H. Harvey. Revealing the history of infectious disease epidemics through phylo-

genetic trees. *Philosophical Transactions: Biological Sciences*, 349(1327):pp. 33–40, 1995.

[33] S. Hossain, M. Islam, Jesmin, and H. M. Jamil. Phyql: A web-based phylogenetic visual query engine. In *IEEE International Conference on Bioinformatics and Biomedicine, 2008.*, pages 295–298, 2008.

[34] John P. Huelsenbeck, B. Larget, R. E. Miller, and F. Ronquist. Potential applications and pitfalls of bayesian inference of phylogeny. *Systematic Biology*, 51:673, 2002.

[35] John P. Huelsenbeck and Frederick Ronquist. MRBAYES: Bayesian inference of phylogenetic trees. *Bioinformatics*, 17(8):754–755, 2001.

[36] Nick J.B. Isaac, Samuel T. Turvey, Ben Collen, Carly Waterman, and Jonathan E.M. Baillie. Mammals on the edge: Conservation priorities based on threat and phylogeny. *PLoS ONE*, 2(3):e296, 03 2007.

[37] Kim Jae-Heup, E. Eizirik, S. J. O'Brien, and W. E. Johnson. Structure and patterns of sequence variation in the mitochondrial dna control region of the great cats. *Mitochondrion*, 1(3):279–292, 2001.

[38] D. N. Janczewski, W. S. Modi, J. C. Stephens, and S. J. O'Brien. Molecular evolution of mitochondrial 12s rna and cytochrome b sequences in the pantherine lineage of felidae. *Molecular Biology and Evolution*, 12(4):690–707, 1995.

[39] Wikimedia User JanErkamp. African leopard in serengeti, tanzania. Availible from http://en.wikipedia.org/wiki/File:Leopard_africa.jpg, February 2007. This Digital image is availible from Wikimedia Commons and is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license.

[40] Warren Johnson and Stephen O'Brien. Phylogenetic reconstruction of the felidae using 16s rrna and nadh-5 mitochondrial genes. *Journal of Molecular Evolution*, 44:S98–S116, 1997. 10.1007/PL00000060.

[41] Warren E. Johnson, Peter A. Dratch, Janice S. Martenson, and Stephen J. O'Brien. Resolution of recent radiations within three evolutionary lineages of felidae using mitochondrial restriction fragment length polymorphism variation. *Journal of Mammalian Evolution*, 3:97–120, 1996. 10.1007/BF01454358.

[42] Warren E. Johnson, Eduardo Eizirik, Jill Pecon-Slattery, William J. Murphy, Agostinho Antunes, Emma Teeling, and Stephen J. O'Brien. The late miocene radiation of modern felidae: A genetic assessment. *Science*, 311(5757):73–77, 2006.

[43] George Karypis. Cluto: A clustering toolkit. Technical Report TR-02-017, University of Minnesota, Department of Computer Science, November 2003.

[44] Steven W. Kembel, Peter D. Cowan, Matthew R. Helmus, William K. Cornwell, Helene Morlon, David D. Ackerly, Simon P. Blomberg, and Campbell O. Webb. Picante: R tools for integrating phylogenies and ecology. *Bioinformatics*, 26(11):1463–1464, 2010.

[45] Bernard Landgraf. A snow leopard (uncia uncia). Availible from http://en.wikipedia.org/wiki/File:Uncia_uncia.jpg, January 2005. This Digital image is availible from Wikimedia Commons and is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license.

[46] Franois-Joseph Lapointe, John A. W. Kirsch, and Robert Bleiweiss. Jackknifing of weighted trees: Validation of phylogenies reconstructed from distance matrices. *Molecular Phylogenetics and Evolution*, 3(3):256–267, 1994.

[47] Susanna K. P. Lau, Patrick C. Y. Woo, Kenneth S. M. Li, Yi Huang, Hoi-Wah Tsoi, Beatrice H. L. Wong, Samson S. Y. Wong, Suet-Yi Leung, Kwok-Hung Chan, and Kwok-Yung Yuen. Severe acute respiratory syndrome coronavirus-like virus in chinese horseshoe bats. *Proceedings of the National Academy of Sciences of the United States of America*, 102(39):14040–14045, 2005.

[48] Louise A. Lewis and Paul O. Lewis. Unearthing the molecular phylodiversity of desert soil green algae (chlorophyta). *Systematic Biology*, 54(6):936–947, 2005.

[49] Wendong Li, Zhengli Shi, Meng Yu, Wuze Ren, Craig Smith, Jonathan H. Epstein, Hanzhong Wang, Gary Crameri, Zhihong Hu, Huajun Zhang, Jianhong Zhang, Jennifer McEachern, Hume Field, Peter Daszak, Bryan T. Eaton, Shuyi Zhang, and Lin-Fa Wang. Bats are natural reservoirs of sars-like coronaviruses. *Science*, 310(5748):676–679, 2005.

[50] Bing Liu, Yiyuan Xia, and Philip S. Yu. Clustering through decision tree construction. In *Proceedings of the ninth international conference on Information and knowledge management*, pages 20–29, New York, 2000. ACM.

[51] D. R. Maddison. The discovery and importance of multiple islands of most parsimonous trees. *Systematic Biology*, 42(2):200–210, 1991.

[52] Thomas Mailund and Christian N. S. Pedersen. QDist–quartet distance between evolutionary trees. *Bioinformatics*, 20(10):1636–1637, 2004.

[53] Sylvie Marcacci and Jean-Paul Schwitzgubel. Using plant phylogeny to predict detoxification of triazine herbicides. In *Phytoremediation*, volume 23 of *Methods in Biotechnology*, pages 233–249. Humana Press, New York, 2007.

[54] Michelle Y. Mattern and Deborah A. McLennan. Phylogeny and speciation of felids. *Cladistics*, 16(2):232–253, 2000.

[55] Suzanne J. Matthews, Seung-Jin Sul, and Tiffani L. Williams. Treezip: A new algorithm for compressing large collections of evolutionary trees. In *Data Compression Conference (DCC), 2010*, pages 544–544. IEEE, 2010.

[56] Suzanne J. Matthews and Tiffani L. Williams. Mrsrf: an efficient mapreduce algorithm for analyzing large collections of evolutionary trees. *BMC Bioinformatics*, 11(Suppl 1):S15, 2010.

[57] Suzanne Jude Matthews. *Efficient algorithms for comparing, storing, and sharing large collections of evolutionary trees*. PhD thesis, Texas A&M University, College Station, 2012.

[58] Michael L. Metzker, David P. Mindell, Xiao-Mei Liu, Roger G. Ptak, Richard A. Gibbs, and David M. Hillis. Molecular evidence of hiv-1 transmission in a criminal case. *Proceedings of the National Academy of Sciences*, 99(22):14292–14297, 2002.

[59] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[60] Sumeet Moghe. A young adult from the terai in india. Availible from http://en.wikipedia.org/wiki/File:Tigress_at_Jim_Corbett_National_Park_.jpg, August 2013. This Digital image is availible from Wikimedia Commons and is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license.

[61] Virginia Morell. TreeBASE: The Roots of Phylogeny. *Science*, 273(5275):569–0, 1996.

[62] Craig Moritz. Uses of molecular phylogenies for conservation. *Philosophical Transactions: Biological Sciences*, 349(1327):pp. 113–118, 1995.

[63] Luay Nakhleh, Daniel Miranker, Francois Barbancon, William H. Piel, and Michael Donoghue. Requirements of Phylogenetic Databases. *IEEE International Symposium on Bioinformatic and Bioengineering*, 0:141+, 2003.

[64] K. C. Nixon. The parsimony ratchet, a new method for rapid parsimony analysis. *Cladistics*, 15:407–414, 1999.

[65] Kristin Nowell. Revision of the felidae red list of threatened species. *Cat News*, 37:4–6, 2002.

[66] Kristin Nowell and Peter Jackson. *Wild cats: status survey and conservation action plan.* IUCN, 1996.

[67] Johan A. A. Nylander, Fredrik Ronquist, John P. Huelsenbeck, and Joseluis Nieves-Aldrey. Bayesian Phylogenetic Analysis of Combined Data. *Systematic Biology*, 53(1):47–67, 2004.

[68] Johan A. A. Nylander, James C. Wilgenbusch, Dan L. Warren, and David L. Swofford. AWTY (are we there yet?): a system for graphical exploration of mcmc convergence in bayesian phylogenetics. *Bioinformatics*, 24(4):581–583, 2008.

[69] Chin-Yih Ou, Carol A. Ciesielski, Gerald Myers, Claudiu I. Bandea, Chi-Cheng Luo, Bette T. M. Korber, James I. Mullins, Gerald Schochetman, Ruth L. Berkelman, A. Nikki Economou, John J. Witte, Lawrence J. Furman, Glen A. Satten, Kersti A. Maclnnes, James W. Curran, Harold W. Jaffe, Laboratory Investigation Group, and Epidemiologic Investigation Group. Molecular Epi-

demiology of HIV Transmission in a Dental Practice. *Science*, 256(5060):1165–1171, 1992.

[70] Roderic Page. TBMap: a taxonomic perspective on the phylogenetic database TreeBASE. *BMC Bioinformatics*, 8(1):158+, 2007.

[71] Roderic Page. Towards a Taxonomically Intelligent Phylogenetic Database. *Nature Precedings*, (713), 2007.

[72] Terence Parr. *The definitive ANTLR reference: Building domain-specific languages*. Pragmatic Bookshelf, Dallas, Texas, 2007.

[73] J. Pecon-Slattery, A. J. Pearks Wilkerson, W. J. Murphy, and S. J. O'Brien. Phylogenetic assessment of introns and sines within the y chromosome using the cat family felidae as a species tree. *Molecular Biology and Evolution*, 21(12):2299–2309, 2004.

[74] Jaakko Peltonen, Jarkko Venna, and Samuel Kaski. Visualizations for assessing convergence and mixing of markov chain monte carlo simulations. *Computational Statistics & Data Analysis*, 53(12):4453–4470, 2009.

[75] Maurice H. Quenouille. Notes on bias in estimation. *Biometrika*, 43(3/4):353–360, 1956.

[76] A. Rambaut and A. J. Drummond. Tracer v1.4. Internet Website, url: http://beast.bio.ed.ac.uk/Tracer, last accessed, March 2014.

[77] Vincent Ranwez, Nicolas Clairon, Frederic Delsuc, Saeed Pourali, Nicolas Auberval, Sorel Diser, and Vincent Berry. PhyloExplorer: a web server to validate, explore and query phylogenetic trees. *BMC Evolutionary Biology*, 9(1):108+, 2009.

[78] D. F. Robinson and L. R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53:131–147, 1981.

[79] Nina Rønsted, Vincent Savolainen, Per Mølgaard, and Anna K. Jäger. Phylogenetic selection of narcissus species for drug discovery. *Biochemical Systematics and Ecology*, 36(56):417–422, 2008.

[80] L. O. Salles. *Felid phylogenetics: extant taxa and skull morphology (Felidae, Aeluroidea).* American Museum novitates. American Museum of Natural History, New York, 1992.

[81] Diane I. Scaduto, Jeremy M. Brown, Wade C. Haaland, Derrick J. Zwickl, David M. Hillis, and Michael L. Metzker. Source identification in two criminal cases using phylogenetic analysis of hiv-1 dna sequences. *Proceedings of the National Academy of Sciences*, 107(50):21242–21247, 2010.

[82] Dennis Shasha, Jason Tsong-Li Wang, Huiyuan Shan, and Kaizhong Zhang. Atreegrep: Approximate searching in unordered trees. In *Proceedings of the 14th International Conference on Scientific and Statistical Database Management*, pages 89–98. IEEE, 2002.

[83] Mark E. Siddall. Another monophyly index: revisiting the jackknife. *Cladistics*, 11(1):33–56, 1995.

[84] D. E. Soltis, P. S. Soltis, M. W. Chase, M. E. Mort, D. C. Albach, M. Zanis, V. Savolainen, W. H. Hahn, S. B. Hoot, M. F. Fay, M. Axtell, S. M. Swensen, L. M. Prince, W. J. Kress, K. C. Nixon, and J. S. Farris. Angiosperm phylogeny inferred from 18s rDNA, *rbcL*, and *atpB* sequences. *Botanical Journal of the Linnean Society*, 133:381–461, 2000.

[85] Douglas E. Soltis, Matthew A. Gitzendanner, and Pamela S. Soltis. A 567-taxon data set for angiosperms: the challenges posed by bayesian analyses of large data sets. *International Journal of Plant Sciences*, 168(2):137–157, 2007.

[86] Matthew Spencer, Elizabeth A. Davidson, Adrian C. Barbrook, and Christopher J. Howe. Phylogenetics of artificial manuscripts. *Journal of Theoretical Biology*, 227(4):503–511, 2004.

[87] Jason E. Stajich, David Block, Kris Boulez, Steven E. Brenner, Stephen A. Chervitz, Chris Dagdigian, Georg Fuellen, James G. R. Gilbert, Ian Korf, Hilmar Lapp, et al. The bioperl toolkit: Perl modules for the life sciences. *Genome Research*, 12(10):1611–1618, 2002.

[88] Cara Stockham, Li-San Wang, and Tandy Warnow. Statistically based post-processing of phylogenetic analysis by clustering. volume 18, pages S285–S293. Oxford Univ Press, 2002.

[89] Michael Stolz. New philology and new phylogeny: Aspects of a critical electronic edition of wolfram's parzival. *Literary and Linguistic Computing*, 18(2):139–150, 2003.

[90] Seung-Jin Sul, Grant Brammer, and Tiffani L. Williams. Efficiently computing arbitrarily-sized robinson-foulds distance matrices. In *Algorithms in Bioinformatics*, pages 123–134. Springer, 2008.

[91] Seung-Jin Sul, Suzanne Matthews, and Tiffani L. Williams. Using tree diversity to compare phylogenetic heuristics. *BMC Bioinformatics*, 10 (Suppl 4)(S3), 2009.

[92] Seung-Jin Sul and Tiffani L. Williams. A randomized algorithm for comparing sets of phylogenetic trees. In *Proceedings of the Fifth Asia Pacific Bioinfor-*

matics Conference, pages 121–130, 2007.

[93] Seung-Jin Sul and Tiffani L. Williams. An experimental analysis of consensus tree algorithms for large-scale tree collections. In *Proceedings of the 5th International Symposium on Bioinformatics Research and Applications*, pages 100–111, Berlin, Heidelberg, 2009. Springer-Verlag.

[94] D. L. Swofford. PAUP*: Phylogenetic analysis using parsimony (and other methods), 2002. Sinauer Associates, Underland, Massachusetts, Version 4.0.

[95] Philip G. Szekeres, Alison I. Muir, Lisa D. Spinage, Jane E. Miller, Sharon I. Butler, Angela Smith, Gillian I. Rennie, Paul R. Murdock, Laura R. Fitzgerald, Hsiao ling Wu, Lynette J. McMillan, Sephanie Guerrera, Lisa Vawter, Nabil A. Elshourbagy, Mooney Jeffery L., Bergsma Derk J., Shelagh Wilson, and Chambers Jon K. Neuromedin U is a potent agonist at the orphan G protein-coupled receptor FM3. *Journal of Biological Chemistry*, 275(27):20247–20250, 2000.

[96] R Development Core Team et al. R: A language and environment for statistical computing. Technical report, ISBN 3-900051-07-0. R Foundation for Statistical Computing. Vienna, Austria, 2013. url: http://www.R-project.org, 2005.

[97] Bjørn C. Tørrissen. Junior jagua belize zoo. Availible from http://en.wikipedia.org/wiki/File:Junior-Jaguar-Belize-Zoo.jpg, December 2010. This Digital image is availible from Wikimedia Commons and is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license.

[98] Nancy Vandermey. Neofelis nebulosa. Availible from http://en.wikipedia.org/wiki/File:Neofelis_nebulosa.jpg, June 2005. This Digital image is availible from Wikimedia Commons and is licensed under the Creative Commons Attribution-Share Alike 3.0 Unported license.

[99] Jarkko Venna and Samuel Kaski. Local multidimensional scaling. *Neural Networks*, 19(6):889–899, 2006.

[100] Jarkko Venna, Jaakko Peltonen, Kristian Nybo, Helena Aidos, and Samuel Kaski. Information retrieval perspective to nonlinear dimensionality reduction for data visualization. *The Journal of Machine Learning Research*, 11:451–490, 2010.

[101] Todd J. Vision. Open data and the social contract of scientific publishing. *BioScience*, 60(5):pp. 330–331, 2010.

[102] Jason TL Wang, Huiyuan Shan, Dennis Shasha, and William H Piel. Fast structural search in phylogenetic databases. *Evolutionary Bioinformatics*, 1:37–46, 2005.

[103] Jason Tsong-Li Wang, Huiyuan Shan, Dennis Shasha, and William H. Piel. Treerank: a similarity measure for nearest neighbor searching in phylogenetic databases. In *15th International Conference on Scientific and Statistical Database Management*, pages 171–180. IEEE, 2003.

[104] Lei Wei, Xiaobing Wu, and Zhigang Jiang. The complete mitochondrial genome structure of snow leopard panthera unci. *Molecular Biology Reports*, 36:871–878, 2009. 10.1007/s11033-008-9257-9.

[105] Edward Orlando Wiley and Bruce S Lieberman. *Phylogenetics: Theory and practice of phylogenetic systematics.* John Wiley & Sons, Hoboken, New Jersey, 2011.

[106] Lixin Yu, Hongwei Song, Shaozhe Lu, Zhongxin Liu, Linmei Yang, and Xianggui Kong. Luminescent properties of lapo4: Eu nanoparticles and nanowires. *ChemInform*, 36(1):no–no, 2005.

APPENDIX A

DATA SETS

- *Data set #1:* 20,000 trees obtained from a Bayesian analysis of an alignment of 150 taxa (23 desert taxa and 127 others from freshwater, marine, ands oil habitats) with 1,651 aligned sites [48]. Two independent runs consisting of 25 million generations (trees were sampled every 1,000 generations) were performed using the GTR+I+Γ model in MrBayes with four independent chains. The authors constructed a majority consensus tree in their study using the 20,000 trees from the last 10 million generations from each of the two runs. The resolution rates of the majority and strict consensus trees are 85.7% and 34.0%, respectively. The total number of clades across the 20,000 trees is 2,940,000, where 1,168 of them are unique.

Data set #1: 150 taxa Bayesian trees		
run	r_0	r_1
-	10,000	10,000

Table A.1: The number of 150 taxa Bayesian trees in each run. There are 2 total runs and 20,000 total trees.

- *Data set #2:* 33,306 trees obtained from an analysis of a three-gene, 567 taxa (560 angiosperms, seven outgroups) data set with 4,621 aligned characters, which is one of the largest Bayesian analysis done to date [85]. Twelve runs, with four chains each, using the GTR+I+Γ model in MrBayes ran for at least 10 million generations. Trees were sampled every 1,000 generations. The authors discuss the difficulties with combining trees from multiple runs. To obtain our collection of 33,306 trees, we discard the trees from the first 8 million generations. The resolution rates of the majority and strict consensus trees are 92.6% and 51.8%, respectively. The total number of bipartitions across the 33,306 trees is 18,784,584, where 2,444 of them are unique.

- *Data set #3:* 100 trees were obtained from Gordon Burleigh. Theses trees are part of a bootstrap analysis of a 950 taxa gymnosperm data set.
- *Data set #4:* 200 trees were obtained from Gordon Burleigh. Theses trees are part of a bootstrap analysis of a 950 taxa Saxifragales data set.

Data set #2: 567 taxa Bayesian trees												
run	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}	r_{11}
-	3177	3178	3178	3179	2985	2985	2986	2987	2162	2162	2163	2164

Table A.2: The number of 567 taxa Bayesian trees in each run. There are 12 total runs and 33,306 total trees.

- *Data set #5:* 4,898 trees from a maximum parsimony (MP) analysis of the 567 "three-gene" data set. This is a further analysis using different methods of the data set in Data Set #2 [84]. Two MP algorithms, parsimony ratchet [64] and Rec-I-DCM33 were used to infer the phylogenies. Each MP algorithm created 5,000 trees for a total of 10,000 trees. Parsimony ratchet was used to was call Pauprat and was based on a Perl script by Bininda-Emonds [4] to generate a PAUP* [94] batch file to run the parsimony ratchet heuristic. The set was culled to remove any trees that were less and near-optimal. For our experiments, we use parsimony trees that are step$_0$, step$_1$, and step$_2$ away from the best-known maximum parsimony score for this data set, which is 44,165. Let x represent the parsimony score of a tree t_i. Then, tree t_i is $x - b$ steps away from the best score. In our experiments, step$_0$, step$_1$, and step$_2$ represents trees that are 0, 1, and 2 steps away from the best score, b, respectively. Between the two algorithms, there are 4,898 trees that fit this criteria.

Data set #5: 567 taxa maximum parsimony trees										
run	r_0	r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9
-	246	179	197	169	234	796	764	761	791	761

Table A.3: The number of maximum parsimony trees in each run. Runs r_0 to r_4 were obtained from Pauprat. The remaining trees were collected from Rec-I-DCM33.

APPENDIX B

PQL GRAMMER AND LEXER

Listing B.1: Rules for the PQL grammar

```
prog
: query+;


query
: (assignment_expression )? NEWLINE;


assignment_expression
: IDENTIFIER '=' assignment_expression
| not_expression ;


not_expression
: ('!')? difference_expression;


difference_expression
 : union_expression  ( '-' union_expression )*;


union_expression
: intersection_expression ( '+' intersection_expression )*;


intersection_expression
```

```
: equality_expression  ('^' equality_expression)*;

equality_expression
 : postfix_expression (('==' |'!=' ) postfix_expression)* ;

postfix_expression
: primary_expression (  '[' assignment_expression  ']'
      |  '(' ')'
      |  '(' argument_expression_list  ')' )*;

argument_expression_list
: assignment_expression (',' assignment_expression )*;

primary_expression
: IDENTIFIER
| atom
| '('assignment_expression  ')';

constant_expression_list
: ( assignment_expression  | RANGE_LITERAL )
  (',' (assignment_expression | RANGE_LITERAL ) )* ;

atom
: constant
| '['  ( constant_expression_list )? ']'
| '{' (constant_expression_list)?  '}' ;

constant
: ( DECIMAL_LITERAL )
|  ( CHARACTER_LITERAL )
|  (STRING_LITERAL )
|  (FLOATING_POINT_LITERAL) ;
```

Listing B.2: Rules for the PQL lexer

```
IDENTIFIER
: LETTER (LETTER|'0'..'9')*;

fragment
LETTER
:   '$'
|   'A'..'Z'
|   'a'..'z'
|   '_';

CHARACTER_LITERAL
: '\'' ( ESCAPE_SEQUENCE | ~('\''|'\\') ) '\'';

STRING_LITERAL
: '"' STRING_GUTS '"';

fragment
```

```
STRING_GUTS
: ( ESCAPE_SEQUENCE | ~('\\'|'"') )* ;

RANGE_LITERAL
  : ('-')? ('0'..'9')+ '..' ('-')?('0'..'9')+ ;

DECIMAL_LITERAL
: ('0' | ('-')? '1'..'9' '0'..'9'*)   ;

FLOATING_POINT_LITERAL
: ('-')?('0'..'9')+ '.' ('0'..'9')*
|      '.' ('0'..'9')+ ;

fragment
ESCAPE_SEQUENCE
:     '\\' ('b'|'t'|'n'|'f'|'r'|'\"'|'\''|'\\');

COMMENT
  :     '/*' ( : . )* '*/' ;

LINE_COMMENT
: '//' ~('\n'|'\r')* ;

NEWLINE: ('\r\n'|'\n'|'\r');
```

```
WS  :   (' '|'\r'|'\t'|'\u000C'|'\n');
```