Reconstructing Phylogenies with Variable Evolution Rates Among Sites

Garrett Tetrault, grt43

Introduction

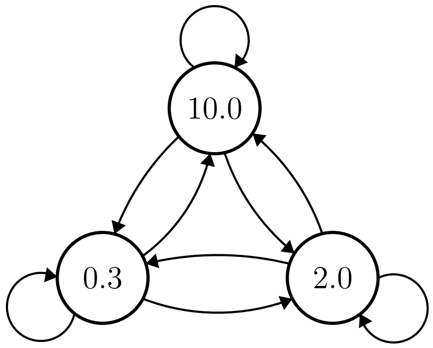
In the phylogenetic models we have developed in class, there have been drastic simplifications to allow for easy understanding and computation. One principle simplification has been the assumption that each site in the genetic sequences evolves at the same rate. For example, there may be several conserved regions in the sequences with only a subset of regions displaying significant change. This is what is captured by modeling different rates of evolution. In this paper, I will focus on the following paper by Felsenstein and Churchill that presents a model to compute the likelihood of a phylogeny, allowing for unequal evolutionary rates at different sites in the molecular sequences.

J Felsenstein, G A Churchill, A Hidden Markov Model approach to variation among sites in rate of evolution., Molecular Biology and Evolution, Volume 13, Issue 1, Jan 1996, Pages 93–104, `https://doi.org/10.1093/oxfordjournals.molbev.a025575`

We will begin by describing the model and likelihood calculations, then transition to examining how different sets of evolutionary rates effect a phylogenies likelihood and its maximum likelihood assignment of evolutionary rates.

Evolution Rate Model

At the core of the model developed by Felsenstein and Churchill is a Hidden Markov Model that describes the changes in evolutionary rates at each possible site in the different DNA sequences in a given phylogeny. Each node in the model corresponds to a different rate of evolution with transitions between all possible states. For example, consider the set of evolutionary rates $\{10.0, 2.0, 0.3\}$. A Hidden Markov Model for their model must be of the form,



Below, we have a visual depiction of how the aforementioned Hidden Markov model can be used to describe site-wise differences in evolutionary rate,

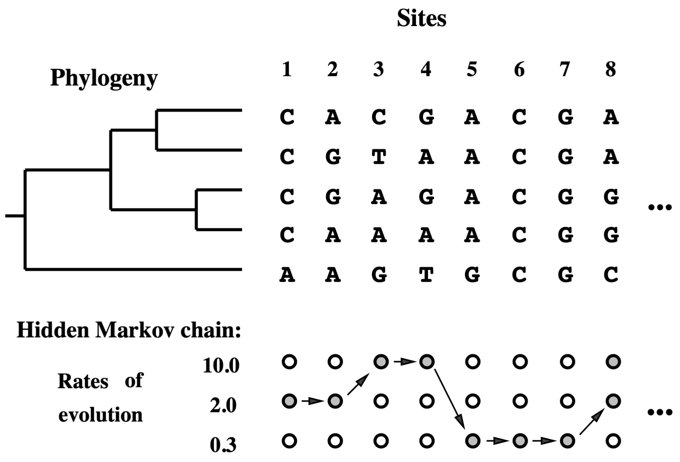


Image taken from original paper.

This model has some important implications. Chiefly, it supposes we have some preconceived notion of how a finite set of evolutionary rates relate to one another in a Hidden Markov Model. That is, as input, the model requires the different rates $\{r_i\}$ to be known and the transition probabilities P_{ij} from rate r_i to rate r_j to be known as well. Secondly, as a result of the use of a Hidden Markov Model, each site evolves independently from all other sites apart from that directly preceding it. In practice, and in our implementation, it is much easier to use the equilibrium probabilities of each rate instead of specifying each transition probability. We will denote f_i to be the equilibrium probability of rate r_i. As a further piece of notation, we will use c_k to denote the rate category (node in Hidden Markov Model) of the underlying rate r_i at site k.

Likelihood Calculations

The crux in a model like this is formulating a way to compute its likelihood given some phylogeny. We will assume we are given sequence data D and some tree topology T.

Below, we state the formulation that was derived by Felsenstein and Churchill. Let L be the likelihood of the model, and $L_{c_k}^{(k)}$ be the likelihood of T for the data consisting of sites k through n given that site k has rate category c_k. Then we have,

$$L = \sum_{c_1} f_{c_1} L_{c_1}^{(1)}$$
$$L_{c_k}^{(k)} = \text{Prob}(D_k \mid T, r_{c_k}) \sum_{c_{k+1}} P_{c_k,c_{k+1}} f_{c_{k+1}} L_{c_{k+1}}^{(k+1)}$$
$$L_{c_n}^{(n)} = \text{Prob}(D_n \mid T, r_{c_n})$$

What is interesting to note is that this model is computing the likelihood of all possible combinations of rates at each site, not just some optimal sequence of rates.
Methods were also developed to recover the most likely sequence of rates. Let $R_{c_k}^{(k)}$ be the likelihood contribution for sites k through n with site k having rate category c_k, and sites $k+1$ through n having some combination of categories that maximizes the contribution of sites k through n. We then have that,

$$R_{c_k}^{(k)} = \text{Prob}(D_k \mid T, r_{c_k}) \max_{c_{k+1}} \left\{ P_{c_k,c_{k+1}} f_{c_{k+1}} L_{c_{k+1}}^{(k+1)} \right\}$$
$$R_{c_n}^{(n)} = \text{Prob}(D_n \mid T, r_{c_n})$$

Let c_1 be the rate that maximizes the value of $R_{c_1}^{(k)}$. If we store the sequence $\{c_k\}$ that is chosen in the maximizing steps, we can recover the maximal likelihood sequences backtracking over our choices in a process that is very similar to the Viterbi algorithm.
In both of these computations we must compute $\text{Prob}(D_k \mid T, r_{c_k})$. Below, we have an expression for this value assuming we are using the Jukes-Cantor model of evolution (which we will do for our implementation). First, let $\ell_{ic}^{(m)}(b)$ be the likelihood of T for all data for site m at or above node i on the tree, given that site m in node i is basis b, and given that site m has rate category c. As the calculation of this value is exactly that in Felsenstein's algorithm, we will not be stating directly its formulation. Let $M_{xy}(v, r)$ denote the site-wise evolution model (in this case Juke-Cantor) that denotes the probability of transitioning from basis x to basis y with a branch length of v and evolutionary rate r. With this, we then have that,

$$\text{Prob}(D_i \mid T, r_{c_i}) = \frac{1}{4} \sum_x \sum_y \ell_{jc_i}^{(i)}(x) M_{xy}(v, r_{c_i}) \ell_{kc_i}^{(i)}(y).$$

We are now well-equipped to implement to the model.

Implementation Details

As we have mentioned, we will be using the Juke-Cantor model of evolution in our implementation. This is a departure from the more complex Hasegawa, Kishino and Yano model used in the paper's implementation. Additionally, we have set our transition probabilities to be,

$$P_{ij} = \lambda\delta_{ij} + (1-\lambda)f_j.$$

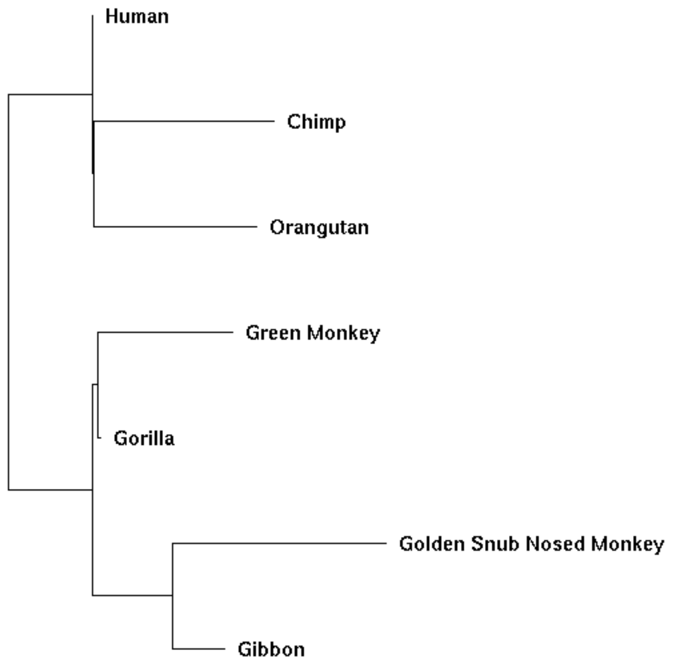
Here, λ is an 'autocorrelation' parameter such that the average expected patch length is $^1/_{(1-\lambda)}$. This is set by the user.
For data, we turn to the USCS genome browser. Specifically, we are examining the β-hemoglobin DNA sequences. We use the following species,

{human, gorilla, chimp, gibbon, green monkey, golden snub-nosed monkey,and orangutan}

To obtain a phylogeny from this, we use the PHYLIP software package implemented by Felsenstein and use this to input a tree topology and branch lengths into our model.

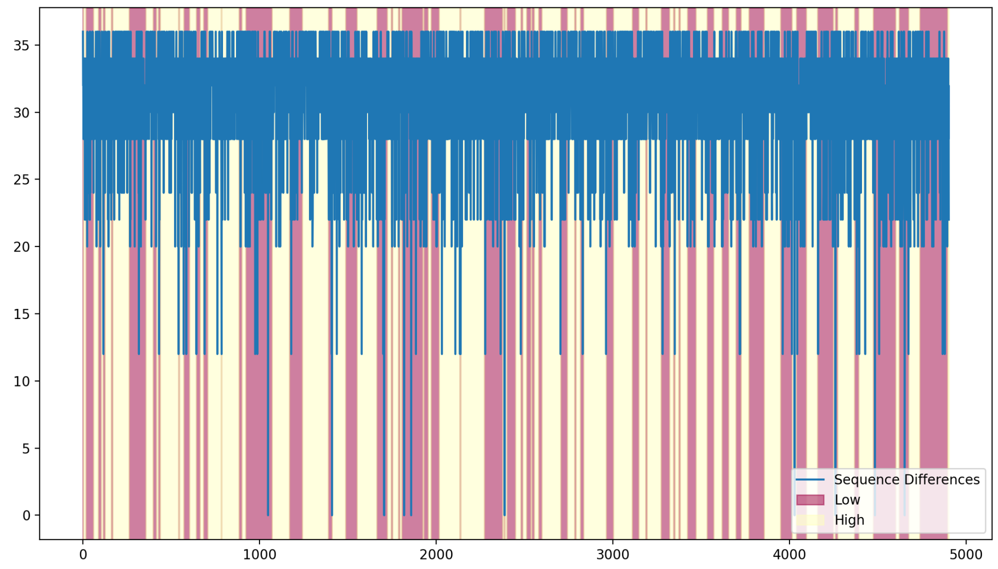
Results

Below, we have the phylogeny returned to us by the PHYLIP package,

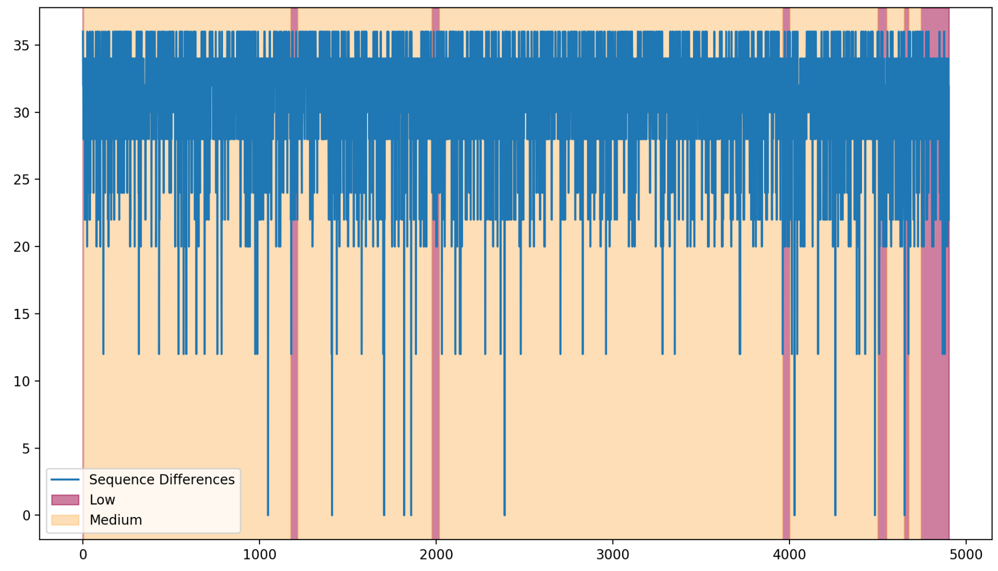


We will now begin examining different rates of evolutions and different equilibrium probabilities. Note that the evolution rates specified were attained from experimentation. There appeared to be a threshold for the evolutionary rates for this data set around 0.75 at which the chance of selecting the evolutionary rate was extremely low. This stands to reason as we are working only with primate species and would expect them to be evolutionarily similar.

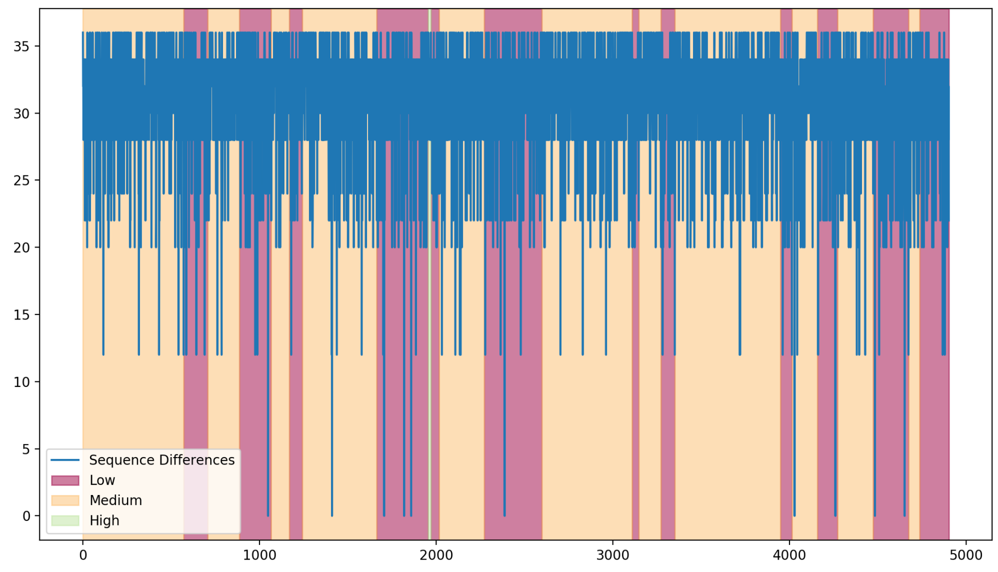
For the first analysis, we will set the auto-correlation parameter to be $\lambda = 0.9$. As a base line, we first compute the likelihood of the model with a flat evolutionary rate of 0.3. With this, we find a log likelihood of `-64420.0656448217`.
We first split the evolutionary rates into a high rate of 0.4 and a low rate of 0.2, with equal equilibrium probabilities. That is, $f_i = 0.5$ for $i \in \text{Low}, \text{High}$. With these conditions we find that the log-likelihood is `-64569.30264119623`. To visualize this, we plot the rate categories as shading regions and differences in bases among sequences below,



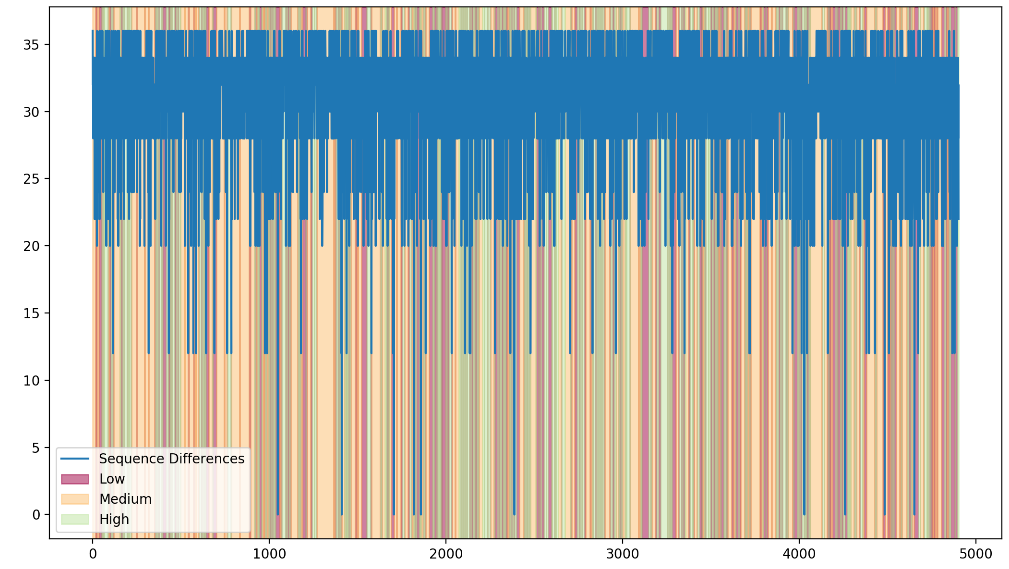
We now allow for a third evolutionary rate model of 0.3. We first test the results when the equilibrium probabilities are all equal. That is, $f_i = 0.333$ for $i \in \text{Low}, \text{Medium}, \text{High}$. With these conditions we find that the log-likelihood is `-64480.19948628911`. To visualize this, we plot the rate categories as shading regions and differences in bases among sequences below,



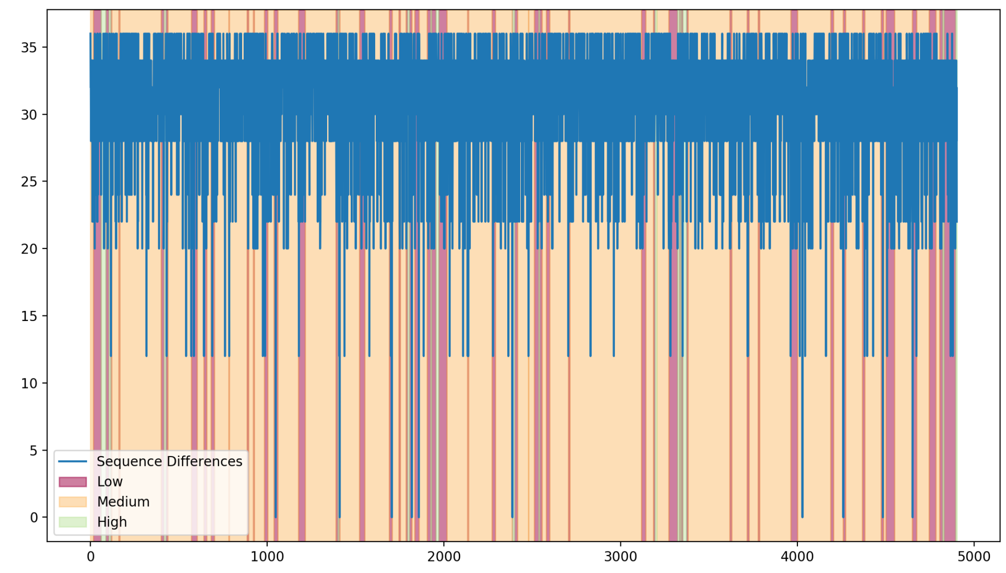
We now examine what happens when we wildly change the equilibrium probabilities. We will let $f_{\text{Low}} = 0.8$ and $f_{\text{Medium}} = f_{\text{High}} = 0.1$. With these conditions we find that the log-likelihood is `-64571.29739397782`. To visualize this, we plot the rate categories as shading regions and differences in bases among sequences below,



From this we can see that, even if we weight the preference greatly towards a specified evolutionary rate, the dominant rate, in this case the Medium rate, will still be the most apparent. Interestingly, these results indicate that the model was most likely (in our tests) with a static evolution rate of 0.3. This could be potentially due to, again, the fact that these species are fairly closely related and that we are working on the same protein.
Returning to the model in which we have a high, medium, and low rates of 0.4, 0.3, and 0.2 respectively with equal equilibrium probabilities. Now, however, we will examine different autocorrelation parameters.
We begin with $\lambda = 0.2$. With these conditions we find that the log-likelihood is `-64482.77727355491`. To visualize this, we plot the rate categories as shading regions and differences in bases among sequences below,



We now consider $\lambda 0.5$. With these conditions we find that the log-likelihood is `-64481.185185916955`. To visualize this, we plot the rate categories as shading regions and differences in bases among sequences below,



As would be expected, we can see that varying λ determines the probability we can transition from one state to another. In the case of a very low λ, we are seeing the evolution rates over fit the data, switching at nearly every position.

Extensions

A significant extension to this project would be to use the likelihood calculations to determine the maximum likelihood phylogeny.

```
(1) Begin with one DNA sequence.

(2) For each subsequent sequence:

    (3) For each possible edge to join on tree:

        (4) Compute the maximum likelihood branch length
        by Newton's Method.

        (5) Compute the likelihood of the tree with the
        new sequence and branch.

    (6) Keep the maximum likelihood tree topology and
    branch lengths.

(7) Return the final tree topology.
```

To compute the maximum likelihood branch length, first recall the basic structure of Newton's Method. Let (j, k) be the new edge we are adding. Consider likelihood as a function of the branch length $L(v)$. To maximize, we find v such that $\frac{dL}{dv} = 0$. This is done by a recursive formula that approaches a value where this is true. That recursive rule is,

$$v_{i+1} = v_i - \left(\frac{dL}{dv}(v_i) \bigg/ \frac{d^2L}{dv^2}(v_i) \right)$$

These derivatives can be computed from straight forward calculations of the previous formulas. The paper outlines how they can be computed for their model, however as we have mentioned, they used the Hasegawa, Kishino and Yano model as opposed to the Jukes-Cantor model. In the Jukes-Cantor model, the derivatives are given by,

$$\frac{dL}{dv} = \sum_{c_1} f_{c_1} \frac{dL_{c_1}^{(1)}}{dv}$$

$$\begin{aligned} \frac{dL_{c_k}^{k}}{dv} = \Bigg(& \frac{d\,\mathrm{Prob}(D_k \mid T, r_{c_k})}{dv} \sum_{c_{k+1}} P_{c_k,c_{k+1}} f_{c_{k+1}} L_{c_{k+1}}^{(k+1)} \\ & + \mathrm{Prob}(D_k \mid T, r_{c_k}) \sum_{c_{k+1}} P_{c_k,c_{k+1}} f_{c_{k+1}} \frac{dL_{c_{k+1}}^{(k+1)}}{dv} \Bigg) \end{aligned}$$

$$\frac{dL_{c_n}^{(n)}}{dv} = \frac{d\,\mathrm{Prob}(D_n \mid T, r_{c_n})}{dv}$$

and,

$$\frac{d^2L}{dv^2} = \sum_{c_1} f_{c_1} \frac{d^2L_{c_1}^{(1)}}{dv^2}$$

$$\begin{aligned} \frac{d^2L_{c_k}^{k}}{dv^2} = \Bigg(& \frac{d^2\,\mathrm{Prob}(D_k \mid T, r_{c_k})}{dv^2} \sum_{c_{k+1}} P_{c_k,c_{k+1}} f_{c_{k+1}} L_{c_{k+1}}^{(k+1)} \\ & + 2\frac{d\,\mathrm{Prob}(D_k \mid T, r_{c_k})}{dv} \sum_{c_{k+1}} P_{c_k,c_{k+1}} f_{c_{k+1}} \frac{dL_{c_{k+1}}^{(k+1)}}{dv} \\ & + \mathrm{Prob}(D_k \mid T, r_{c_k}) \sum_{c_{k+1}} P_{c_k,c_{k+1}} f_{c_{k+1}} \frac{d^2L_{c_{k+1}}^{(k+1)}}{dv^2} \Bigg) \end{aligned}$$

$$\frac{d^2L_{c_n}^{(n)}}{dv^2} = \frac{d^2\,\mathrm{Prob}(D_n \mid T, r_{c_n})}{dv^2}$$

To compute the site-wise likelihood derivatives, we recall that with the Jukes-Cantor model,

$$\mathrm{Prob}(D_i \mid T, r_{c_i}) = e^{-\frac{4}{3}vr_{c_i}}K_1 + K_2$$

where

$$K_1 = \frac{1}{4}\sum_x\sum_y \ell_{jc_i}^{(i)}(x)\left(\delta_{xy} - \frac{1}{4}\right)\ell_{kc_i}^{(i)}(y)$$
$$K_2 = \frac{1}{16}\sum_x\sum_y \ell_{jc_i}^{(i)}(x)\ell_{kc_i}^{(i)}(y)$$

From this, we can derive the derivatives as follows,

$$\frac{d\,\mathrm{Prob}(D_i \mid T, r_{c_i})}{dv} = -\frac{4}{3}r_{c_i}e^{-\frac{4}{3}vr_{c_i}}K_1$$
$$\frac{d^2\,\mathrm{Prob}(D_i \mid T, r_{c_i})}{dv^2} = \left(\frac{4}{3}r_{c_i}\right)^2 e^{-\frac{4}{3}vr_{c_i}}K_1$$

One would now have all the analytical pieces for computing steps (4) and (5) in the aforementioned algorithm. Due to the time constraints and the complexity of these calculation (particularly in log-space), they have not been implemented in this project. However, this extended project would provide a more robust self-contained analysis of phylogenetic trees.

References

1. J Felsenstein, G A Churchill, A Hidden Markov Model approach to variation among sites in rate of evolution., Molecular Biology and Evolution, Volume 13, Issue 1, Jan 1996, Pages 93–104, `https://doi.org/10.1093/oxfordjournals.molbev.a025575`

2. Durbin, Richard et al. "Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids." (1998).

Code

phylogeny.py

```
import numpy as np

''' Phylogeny
Phylogeny class defined by evolution rate model. Note that sequences
are added to the phylogeny individually and NOT at initialization.
'''
class Phylogeny:

    # Constant value.
    BASES = ['A','C','G','T']


    #
    ---------------------------------------------------------------
    #                                              Object
    Construction

    ''' __init__
    Initialize a phylogeny from a given evolution rate model.
    '''
    def __init__(self, rates, nodes, tree, root, data, seqlen,
    branch_lengths):
        # Rates are a dictionary of the form,
        # {<rate class>: {
        #     'rate': <evolution rate>,
        #     'prob': <rate equilibrium probabiliy>}}
        self.rates = rates

        # List of nodes in tree.
        self.nodes = nodes

        # Tree topology of the form,
        # {<node>: [<children of node>]}
        self.tree = tree

        # Will be some node in tree.
        self.root = root

        # Genentic data of the form,
        # {<sequence ID>: <sequence>}
        self.data = data

        self.seqlen = seqlen

        # Distance along branches between nodes of the form,
        # {<node1>: {
        #     <node2>: <distance from node1 to node2>}}
        self.branch_lengths = branch_lengths


    #
    ----------------------------------------------------------------
    #                                                  Graph
    Algorithms

    ''' _bfs
    Return the order in which nodes are traversed in a breadth-first
    from the root node.
    '''
    def _bfs(self):
        # Begin with no nodes visited.
        visited = []
        queue = [self.root]

        # While there are still nodes to be visited.
        while queue:
            node = queue.pop(0)

            if node not in visited:
                visited.append(node)
                neighbours = self.tree[node]

                for neighbour in neighbours:
                    queue.append(neighbour)

        return visited


    #
    ----------------------------------------------------------------
    #                                                  Model
    Algorithms

    ''' _log_sum
    Calculate log(a + b).
    '''
    def _log_sum(self, a, b):
        # Avoid returning NaN.
        if a == -np.Inf and b == -np.Inf:
            return -np.Inf

        elif a > b:
            return a + np.log1p(np.exp(b - a))

        else:
            return b + np.log1p(np.exp(a - b))


    ''' _p_rate_trans
    Get the probability the evolution rate trasitions from i to j.
    '''
    def _p_rate_trans(self, i, j, auto_coef=np.log(0.7)):
        if i == j:
            return self._log_sum(auto_coef,
                np.log(1 - np.exp(auto_coef)) + self.rates[j]['prob'])
        else:
            return (np.log(1 - np.exp(auto_coef)) + self.rates[j]['prob'])


    ''' _jcm
    Evaluate M_{ij}(time, rate) under the Jukes-Cantor model. In this
    model, we are assuming that branch length is (time * rate).
    Arguments:
        i, j: DNA bases
        length: branch length
        rate: rate of evolution
    '''
    def _jcm(self, i, j, length, rate):
        if i == j:
            return np.log(0.25) + np.log(1 + 3 * np.exp((-4/3) * length * rate))
        else:
            return np.log(0.25) + np.log(1 - np.exp((-4/3) * length * rate))


    ''' _update_p_nodes
    Compute the node probabilities (ell values in paper) for each site.
    '''
    def _get_p_nodes(self, site, rate):
        p_nodes = {node:{b:-np.Inf for b in self.BASES} for node in self.nodes}

        order = self._bfs()

        for node in reversed(order):
            for basis in self.BASES:
                # If we are at a leaf.
                if not self.tree[node]:
                    # Kronecker delta function.
                    delta = 0 if (self.data[node][site] == basis) else -np.Inf
                    p_nodes[node][basis] = delta

                # If the node has children.
                else:
                    left, right = self.tree[node]
                    l_dist = self.branch_lengths[node][left]
                    r_dist = self.branch_lengths[node][right]

                    for x in self.BASES:
                        for y in self.BASES:
                            l_prob = self._jcm(basis, x, l_dist, self.rates[rate]['rate']) \
                                + p_nodes[left][x]

                            r_prob = self._jcm(basis, y, r_dist, self.rates[rate]['rate']) \
                                + p_nodes[right][y]

                            p_nodes[node][basis] = self._log_sum(p_nodes[node][basis], l_prob + r_prob)

        return p_nodes


    ''' likelihood
    Compute the likelihood of a tree.
    '''
    def likelihood(self):
        # Likelihood of the tree given data and that a site has some
        # specfic rate category. Initialized outisde of loop so we
        # can make use of recursive formula.
        ll_tree = {rate:-np.Inf for rate in self.rates}

        # Likelihood of the contribution of rates that maximizes
        # likelihood at a given site.
        ll_rates = {rate:-np.Inf for rate in self.rates}

        site_rates = [{rate:None for rate in self.rates} for site in range(self.seqlen)]

        # Note that we start at the end of the sequence.
        for site in reversed(range(self.seqlen)):
            #print(ll_tree)
            ll_site = {rate:-np.Inf for rate in self.rates}

            # Compute the likelihood at each site.
            for rate in self.rates:
                p_nodes = self._get_p_nodes(site, rate)

                tail = self.root
                head = self.tree[tail][0]
                length = self.branch_lengths[tail][head]

                for x in self.BASES:
                    for y in self.BASES:
```

```python
                        ll_site[rate] = self._log_sum(ll_site[rate],
                            np.log(0.25)
                            + p_nodes[tail][x]
                            + p_nodes[head][y]
                            + self._jcm(x, y, length, self.rates[rate]['rate']))

            # Recursively calculate next.
            # Base case.
            if site == (self.seqlen - 1):
                for rate in self.rates:
                    ll_tree[rate] = ll_site[rate]
                    ll_rates[rate] = ll_site[rate]
                    site_rates[site][rate] = rate

            else:
                ll_tree_new = {rate:-np.Inf for rate in self.rates}
                ll_rates_new = {rate:-np.Inf for rate in self.rates}

                for i in self.rates:
                    rate_coef = -np.Inf

                    max_rate_contribution = -np.Inf
                    max_rate_category = i

                    for j in self.rates:
                        ll_contribution = self._p_rate_trans(i, j) + ll_tree[j]
                        rate_contribution = self._p_rate_trans(i, j) + ll_rates[j]

                        rate_coef = self._log_sum(rate_coef, ll_contribution)

                        if rate_contribution > max_rate_contribution:
                            max_rate_contribution = rate_contribution
                            max_rate_category = j

                    ll_tree_new[i] = ll_site[i] + rate_coef
                    ll_rates_new[i] = ll_site[i] + max_rate_contribution
                    site_rates[site][i] = max_rate_category

                ll_tree = ll_tree_new
                ll_rates = ll_rates_new
            # print(ll_rates)

        # Compute final likelihood of tree.
        ll = -np.Inf

        for rate in self.rates:
            ll = self._log_sum(ll, ll_tree[rate])

        # Compute maximal sequence of rates by backtracking through
        # our maximal choices.
        final_rate_list = []
        for site in range(self.seqlen):
            if site == 0:
                final_rate_list.append(max(site_rates[site],
    key=site_rates[site].get))
            else:

    final_rate_list.append(site_rates[site][final_rate_list[site-1]])

        return final_rate_list, ll
```

main.py

```python
from phylogeny import Phylogeny
import matplotlib.pyplot as plt
from matplotlib import cm
import numpy as np
import argparse

'''
Arguments:
    filename: name of fasta file to read
Returns:
    sequences: dictionary of outputs (string (sequence id) ->
   sequence (string))
    size: length of each sequence
'''
def read_data(filename):
    with open(filename, "r") as f:
        lines = f.readlines()
        sequences = {}
        output = ''
        size = 0
        curr = ''
        flag = False

        for line in lines[1:]:
            l = line.split()
            sequences[l[0]] = l[1]
            size = len(l[1])

    return sequences, size


def main():
    # Define phylogeny.
    rates = {
        'r1': {'rate': 0.2, 'prob': np.log(0.30)},
        'r2': {'rate': 0.3, 'prob': np.log(0.30)},
        'r3': {'rate': 0.4, 'prob': np.log(0.40)}
    }
    # rates = {
    #     'Low': {'rate': 0.2, 'prob': np.log(0.333)},
    #     'Medium':{'rate': 0.3, 'prob': np.log(0.333)},
    #     'High': {'rate': 0.4, 'prob': np.log(0.333)}
    # }

    # rates = {
    #     'Medium':{'rate': 0.3, 'prob': np.log(1)}
    # }

    nodes = ['human',
        'gorilla',
        'chimp',
        'gibbon',
        'golden',
        'orangutan',
        'green',
        'root',
        'interim1',
        'interim2',
        'interim3',
        'interim4',
        'interim5']

    tree = {
        'human': [],
        'gorilla': [],
        'chimp': [],
        'gibbon': [],
        'golden':[],
        'orangutan':[],
        'green':[],
        'root':['interim1', 'interim2'],
        'interim1': ['human', 'interim5'],
        'interim2': ['interim3', 'interim4'],
        'interim3': ['green', 'gorilla'],
        'interim4': ['golden', 'gibbon'],
        'interim5': ['chimp', 'orangutan']
    }

    root = 'root'

    data, seqlen = read_data('../data/dna_data.txt')
    # seqlen = 500

    branch_lengths = {
        'human': {},
        'gorilla': {},
        'chimp': {},
        'gibbon': {},
        'golden': {},
        'orangutan':{},
        'green': {},
        'root':{'interim1': 1.197025, 'interim2': 1.197025},
        'interim1': {'human': 0.00006, 'interim5': 2.87046},
        'interim2': {'interim3': 0.12230, 'interim4': 1.82491},
        'interim3': {'green': 3.09816, 'gorilla': 3.81994},
        'interim4': {'golden': 4.93107, 'gibbon': 1.22929},
        'interim5': {'chimp': 4.14481, 'orangutan': 3.75325}
    }

    # Compute likelihood and most likely rates.
```

```python
    phylo = Phylogeny(rates, nodes, tree, root, data, seqlen,
    branch_lengths)
    rate_list, ll = phylo.likelihood()

    print('log likelihood=' + str(ll))
    print('most likely sitewise rates=' + str(rate_list))

    # Collect data and format for plotting.
    evo_rates = []
    seq_diffs = []
    rate_range = {rate:[] for rate in rates}

    prev_elem = rate_list[0]
    range_start = 0
    for site in range(seqlen):
        if not (rate_list[site] == prev_elem):
            rate_range[prev_elem].append((range_start, site))
            prev_elem = rate_list[site]
            range_start = site

        evo_rates.append(rates[rate_list[site]]['rate'])

        diffs = 0
        for species1 in data:
            for species2 in data:
                if data[species1][site] != data[species2][site]:
                    diffs += 1
        seq_diffs.append(diffs)
    rate_range[prev_elem].append((range_start, seqlen - 1))

    print(rate_range)

    # Plot results.
    fig, ax = plt.subplots()

    colormap = cm.get_cmap('Spectral')
    rate_idx = 0
    for rate in rates:
        range_idx = 0
        color = colormap(rate_idx/len(rates))
        for start, end in rate_range[rate]:
            if range_idx == 0:
                ax.axvspan(start, end, alpha=0.5, color=color,
    label=rate)
            else:
                ax.axvspan(start, end, alpha=0.5, color=color)
            range_idx += 1
        rate_idx += 1

    # ax.plot(range(seqlen), evo_rates)
    ax.plot(range(seqlen), seq_diffs, label='Sequence Differences')
    ax.legend()
    plt.show()

if __name__ == "__main__":
    main()
```