Approximating Amino Acid Replacement Rates Efficiently through Weighted Data Aggregation

Ruben Ridderström

Bachelor's Thesis in Computer Science
at Stockholm University, Sweden, 2019

Stockholm University

Approximating Amino Acid Replacement Rates Efficiently through Weighted Data Aggregation

Ruben Ridderström

Bachelor's Thesis in Computer Science (15 ECTS credits)
Bachelor's Programme in Computer Science
Stockholm University year 2019
Supervisor at the Department of Mathematics was Lars Arvestad
Examiner was Chun-Biu Li

Department of Mathematics
Stockholm University
SE-106 91 Stockholm, Sweden

Abstract

In trying to describe the evolutionary process involved in the creation of proteins with a common ancestry, Markov processes are often used. These Markov processes describe evolution by mutations of the amino acids, which are the building blocks of proteins. Mathematically this process, given a set of proteins in the form of multiple sequence alignments, is represented using a rate matrix Q. The elements of Q signify the rate at which the amino acids mutate among each other.

In this paper we will implement a program using the BW (Bayesian weights)-method [3] which, given one or multiple sequence alignments, estimates the rate matrix Q. The program is tested and shown to be working correctly. In addition a bootstrap extension is implemented that resamples the proteins to produce a metric by which the stability of the produced rate matrix can be evaluated. Tests are done which indicate the extension works for this purpose.

The program is implemented in Python using NumPy with a test suite to facilitate future use and modifications. It is released as open source and can also be downloaded and installed from the Python package index (PyPi).

Sammanfattning

Approximation av aminosyrors utbytesfrekvens genom viktad aggregering av data

För att beskriva den evolutionära processen som ger upphov till proteiner med gemensamt ursprung används ofta Markovprocessser. Dessa Markovprocessser beskriver evolution via mutationer i aminosyrorna som är proteinernas byggstenar. Matematiskt beskrivs den här processen, för en mängd givna proteiner i form av multilinjeringar, med en övergångsmatris Q. Elementen i Q anger med vilken intensitet aminosyrorna ersätter varandra genom mutationer.

I det här projektet kommer vi implementera ett program med hjälp av BW (Bayesian weights)-metoden [3] som, givet en eller flera multipellinjeringar, uppskattar övergångsmatrisen Q. Programmet utsätts för tester som visar på att det fungerar korrekt. En utökning med bootstrapping implementeras som resamplar multilinjeringarna för att producera ett värde med hjälp av vilket stabiliteten av den producerade övergångsmatrisen kan utvärderas. Tester utförs som indikerar att implementationen fungerar.

Programmet är implementerat i Python med hjälp av NumPy. Tester finns implementerade för att facilitera framtida användning och utökningar av programmet. Programmet finns tillgängligt under open source och kan också installeras från Python package index (PyPi).

Acknowledgements

I want to thank the supervisor Lars Arvestad, Senior Lecturer at Stockholm University, for providing the thesis subject. And for the guidance and help he provided during this project. I also want to thank Chun-Biu Li for examining this project. As well as thank Caroline Nordquist, educational counselor, for always being of great assistance during my studies.

Contents

1 Introduction

Proteins are of great importance for organisms who use them for many different functions. A single protein is made up of a sequence of amino acids, differing in length from 20 to 30 amino acids up to several thousand. This sequence decides the protein's three-dimensional shape as well as its function. There exists 20 unique amino acids e.g. alanine, isoleucine and histidine. They are encoded by the letters 'ACDEFGHIKLMNOPQRSTUVWY'. Using this encoding a protein can be encoded as a string.

For our purposes we will be dealing with proteins with a common ancestor in the form of a multialignment. A multialignment consists of a given number of proteins sequences of equal length where the related amino acid positions have been lined up. Each row of the multialignment thus consists of one protein sequence. And each column is the position on which a single amino acid have evolved.

A common area of interest when working with proteins that have a common ancestor is the evolutionary process that gave rise to these proteins. This is often described by a simple Markov process which describes the evolution in terms of mutations in the amino acid positions of the protein. Such a process is represented mathematically by a rate matrix Q. Each element in the rate matrix is the rate at which a given amino acid mutates into another amino acid.

General models like the PAM (Point Accepted Mutation) [5] and BLOSUM (BLOcks SUbstitution Matrix) [10] models have been developed for this purpose. They where created by observing the replacements of many homologous protein pairs and while they are good choices for many purposes they have the drawback that they cannot always fit the data as well as one might wish.

To get a model that better fits the data one can instead choose to use data adapted rate matrices. We will do this by using the BW-method presented in 'Efficient Methods for Estimating Amino Acid Replacement Rates' [3]. We will implement a program utilizing

the BW-method which given a multialignment will produce a rate matrix Q. This gives an easy way to create an adapted model for a given multialignment to be used in any application in which it might be needed.

2 Problem statement

In this project there will be two main focus points. Implementing a program using the BW-method [3] with high usability and maintainability. And implementing an extension to the program using bootstrapping.

2.1 Increasing usability and maintainability

The previous program had two parts. Reading and processing the input was done in Perl while the computational part was implemented in Octave. Although the previous implementation is functional the code is somewhat complicated and hard to understand. This can be attributed both to the fact that the codebase is distributed over two separate program languages, but also due to the fact that Perl has a hard to follow syntax.

As input the program takes multialignments in one of the three multialignment text formats Fasta, Phylip or Stockholm. The functionality of reading a text file of one of these formats can now be found implemented in Python under the open source project BioPython [6]. By getting this functionality from an external package less code has to be implemented and maintained.

The computational part of the program can be implemented in Python using the NumPy library. This allows for the entire program to be implemented in Python which makes the codebase easier to understand and maintain.

In its current implementation any modifications to the code require manual tests to verify that the interface and functionality of the program is maintained. By adding a test suite these tests can be automated which makes the program more stable in the face of future changes.

2.2 Bootstrap extension

A bootstrap extension will also be implemented and evaluated. The goal of this is to be able to give a metric for how stable the estimated rate matrix is for a given multialignment.

3 Markovian model of evolution

The protein sequences we study all start out as a single protein in a common ancestor. As part of evolution multiple copies of this protein can start evolving independently of each other. This can for example happen in a speciation event where a single species become two different species. Our protein then becomes part of two different species. This can keep on occuring until we end up with multiple descendents of our protein in several separate species.

During this time all the proteins keep on evolving independently of each other. To describe this we will assume a Markovian model of evolution. One step in this evolution chain happens when an amino acid in one position of our protein mutates to another amino acid.

3.1 IID and Reversibility

Mutations are assumed to be independent and identically distributed (IID), meaning that the only thing affecting the probability of what will happen in a specific position, is the amino acid currently occupying that position. And that the probability an amino acid will mutate into another, is the same at all times for all proteins.

In practice when a mutation occurs we cannot distinguish which protein sequence mutated into which. Therefore the assumption of reversibility is made. Reversibility says that A has the same odds of mutating into B, as B has to mutate into A. With this assumption we are able to make estimations as to how the proteins evolved, without knowing the direction of evolution.

3.2 Probability matrix and PAM-distance

Mathematically we describe mutation probabilities with a 20×20 probability matrix $P(t)$, such that

$$Pr\{\text{amino acid r} \rightarrow \text{amino acid c}\} = P(t)_{r,c} \tag{1}$$

Each position describes the probability that amino acid r, will mutate into amino acid c, over time t.

The unit of t is PAM-distance. One PAM is defined as the amount of time it takes on average for 1 mutation to occur per 100 amino acids. Because the same position can mutate several times, one PAM doesn't necessarily correspond to one percent of the positions having mutated.

The PAM-distance between two protein sequences A and B with a common ancestor C is defined as the distance from C to A, plus the distance from C to the B.

3.3 The rate matrix and its mathematical properties

The Markov process is described by the rate matrix Q which is closely connected to the probability matrix $P(t)$.

The rate matrix Q is a 20×20 matrix where, with exception for the diagonal, every element is positive and describes the rate at which one amino acid mutates into another. The diagonal elements are equal to the negative sum of the other elements of the row. Thereby the row sums and also total sum of Q is zero.

The connection between the probability matrix $P(t)$ and rate matrix Q is given by the Kolmogorov forward equations [9]. We define $P'(t)$ as the matrix we get if we take the derivative of every element of $P(t)$. The Kolmogorov forward equations are a system of differential

equations that can be stated as

$$P'(t) = P(t)Q \tag{2}$$

which has the solution

$$P(t) = P(0)e^{Qt}. \tag{3}$$

If no time has elapsed the probability that an amino acid will be unchanged is 1. Therefore $P(0)$ is equal to the identity matrix I. And it follows that

$$P(t) = e^{Qt} \tag{4}$$

where by defintion

$$e^{Qt} = \sum_{n=0}^{\infty} Q^n \frac{t^n}{n!}. \tag{5}$$

We thus have that the probability matrix $P(t)$ relates to the rate matrix Q through the matrix exponential

$$P(t) = e^{Qt} = \sum_{n=0}^{\infty} Q^n \frac{t^n}{n!}. \tag{6}$$

If Q has 20 eigenvalues, which will be assumed to be the case, we can use eigendecomposition to factorize it as

$$Q = V\Lambda V^{-1}. \tag{7}$$

Here V contain the eigenvectors of Q in the columns. And Λ contains the eigenvalues of Q on the diagonal. From this it can be shown that $P(t)$ and Q share eigenvectors independent of t. And also that if λ is an eigenvalue of Q, then $e^{\lambda t}$ is an eigenvalue of $P(t)$. If we have the eigenvectors and eigenvalues of $P(t)$ we are thereby able to calculate Q.

4 BW method theory

The BW-method [3] is the main method of this text and will be described in this section. It takes as input a multiple sequence alignment of protein sequences with a shared ancestor. The purpose of the method is to estimate the rate matrix Q which describe the underlying Markov process giving rise to the protein sequences.

By using the eigendecomposition of Q the problem of estimating Q can be broken down into the two separate problems of estimating its eigenvectors and eigenvalues.

4.1 Estimating eigenvectors

For each of the protein sequence pairs we create a 20×20 count matrix N_i for a total of k matrices $N_1, N_2, \ldots, N_k$. Each element (r, c) of N_i correspond to the number of positions in the protein sequence pair which mutated from amino acid r into amino acid c.

The protein sequences of count matrix N_i has evolved from a common ancestral sequence for an unknown time t_i. For this time t_i there exists a probability matrix $P(t_i)$. It can be shown that not only does Q share eigenvectors with $P(t_i)$ independent of t_i, but Q also share eigenvectors with the sum of the probability matrices $\sum_{i=1}^{k} P(t_i)$.

By normalizing the rows of count matrix N_i we get a frequency matrix F_i which approximates the associated probability matrix $P(t_i)$. To use all of the available protein sequence pairs the frequency matrices are summed and normalized.

$$S = \frac{1}{k} \sum_{i=1}^{k} F_i \tag{8}$$

The resulting matrix S share eigenvectors with the probability matrices $P(t_i)$ and the rate matrix Q. This gives us the sought after eigenvectors of the rate matrix.

4.2 Estimating eigenvalues

Finding the eigenvalues of the rate matrix Q turns out to be a harder problem. If the probability matrix $P(t)$ and divergence time t was known the eigenvalues could be calculated directly. Unfortunately protein sequences are often not long enough to enable this to be done with good accuracy. Instead an approach will be used in which eigenvalues are estimated by clustering protein sequence pairs weighted for the probability of a given divergence.

First we choose a set of j divergence times τ_j. The divergences $(5, 10, \ldots, 400)$ where chosen. For each of these divergences we will make an estimation of the probability matrix $P(\tau_j)$. Having probability matrices with known divergences will let us calculate the eigenvalues of the associated rate matrix Q.

To start we will assign a weight to the probability that the protein sequence pairs have a given divergence. Lets say we are interested in finding out how likely it is that count matrix N_i coming from protein sequence pair i has divergence τ. Assuming we have a rate matrix Q we can calculate this probability as

$$\mathbb{P}(N_i|t_i = \tau, Q) = \prod_{r,c} p_{rc}(\tau)^{n_{rc}} \tag{9}$$

with elements $p_{rc}(t)$ from $P(t) = e^{Qt}$.

Using this we assign the weight

$$w_Q(N_i, \tau) = \frac{\mathbb{P}(N_i|t_i = \tau, Q)}{\mathbb{P}(N_i|Q)} \tag{10}$$

to count matrix N_i for divergence τ.

Remember that F_i is the frequency matrix of N_i. And that F_i approximates $P(t_i)$ for the unknown divergence t_i. By summing over the frequency matrices weighted for the probability that they have the given divergence τ we can find an approximation, $\tilde{P}(\tau)$, of $P(\tau)$.

$$\tilde{P}(\tau) = \frac{\sum_{i=1}^{k} F_i w_Q(N_i, \tau)}{\sum_{i=1}^{k} w_Q(N_i, \tau)} \tag{11}$$

The nominator weights each frequency matrix according to its probability to belong to divergence τ. And the denominator scales the matrix to a probability matrix.

We now have an approximated probability matrix $\tilde{P}(\tau)$ with a known divergence τ. This allows us solve for the eigenvalues of the rate matrix Q. In this manner we get an approximated probability matrix $\tilde{P}(\tau)$ for each divergence. Each with its own set of eigenvalues.

It is possible that the divergence τ chosen fits the protein sequence pairs in form of count matrices N_i very poorly. That is to say that τ deviates a lot from the actual divergence time t_i.

To account for the fact that some divergences might fit the data better than others the eigenvalues are calculated using a weighted least-square approach. The eigenvalues originating from $\tilde{P}(\tau_j)$ are assigned the weight

$$\omega_Q(\tau_j) = \sum_{i=1}^{k} w_Q(N_i, \tau_j) \tag{12}$$

The approximated probability matrix for which the divergence is a poor fit to the data will thus have a small impact on the calculated eigenvalues, while those that match the data well give a larger contribution. Having both the eigenvectors and eigenvalues of Q, Q itself can be calculated.

For the first iteration the method requires an initial rate matrix. For this purpose a simple Poission model proposed in 'Bishop and Friday' [4] was used. The process is then iterated over with the newly calculated rate matrix Q used for probability calculations. This is

repeated until the difference between two successively calculated rate matrices Q and Q' is small as measured with the Frobenius norm $|Q - Q'|$. At which point the method is finished and the resulting rate matrix is returned.

5 Bootstrap theory

An extension to allow for confidence testing of a provided mulitalignment using bootstrapping was implemented. The extension uses resampling of the multialignment to estimate how good the data is by measuring the stability of the output.

First the provided multialignment is run through the program to give an estimate of the rate matrix Q. The multialignment is then resampled k times, where more resamplings gives a more reliable result but in turn takes longer to compute. To do a resampling a new empty multialignment is created. From the original multialignment a random column is selected and copied to the new multialignment. This is repeated until the new multialignment contains the same number of columns as the original one. As no columns are removed from the original one, the same column can appear several times in the new one.

The resamplings are then used to calculate k new rate matrices Q'. From this the mean difference of all the matrices is calculated using the Frobenius norm as

$$d = \frac{1}{k} \sum_i |Q - Q'_i| \tag{13}$$

To make d easier to work with it is multiplied by a factor of 10 000. This value is the value returned from the bootstrap extension. And it is the metric by which it will be evaluated. We will henceforth refer to it simply as the bootstrap norm.

6 Correctness testing methodology

To verify that the implementation was working as intended it was put through three test cases. There were two purposes to the tests. The first was to verify that the program was working correctly by checking that the estimated rate matrix was an estimation of the actual rate matrix used to generate the multialignments. The second was to check how the properties of the data used to generate the mulitialignments effected the performance of the program.

One of the properties that define the protein sequences of a multialignment is the structure of the tree that gives rise to the multialignment. For these tests the creation of every multialignment starts out with a randomly generated protein sequence. From this sequence, mutations are introduced and protein sequences are branched off to make the final multialignment.

The simplest possible tree is a fully balanced one where each branch is equally long. A tree with these properties was manually constructed to be used as a test case. The number of leaf nodes was arbitrarily chosen to be 32.

For the second test case an unbalanced tree was manually constructed. In an unbalanced tree different leaf nodes has different amount of ancestor sequences. We chose to create this tree with 38 leaf nodes. This has the drawback of making the comparisons of the unbalanced tree and the balanced tree less clear, but with the benefit of the program being tested on more varied test cases.

The last test case was generated on a manually constructed tree with only 8 leaf nodes to see how that program worked and evaluate how it performed on trees of different sizes. The amount of leaf nodes was arbitrarily chosen. All the trees used to generate the test data can be seen in Figure 1.

To create multialignments from the trees a reference matrix Q had to be used. We chose to use the JTT [2] model. The JTT matrix

was created by estimating from large databases of protein sequences. For our purposes of testing the program any number of well known rate matrices or even a randomly generated one could have been used. The JTT matrix was chosen as its a well known and widely used rate matrix.

To generate data the open source program Seq-Gen [1] which simulates evolution was used. Seq-Gen takes as input the chosen generative model JTT and a tree. A random start sequence is then generated using the equilibrium distribution of the given model which is used to create a multialignment.

For each tree 100 multialignments were generated for protein sequence lengths between 400 and 10 000 amino acids.

When a multialignment contains too little data in the form of not enough amino acid positions it is unable to make an estimation of the rate matrix Q. It instead exits early with an error message. The minimum length of 400 amino acids was chosen after initial testing seemed to indicate that it was near the lower limit for which the program was reliable able to finish for all trees.

The program was then run on the multialignments. The estimated rate matrix Q' was used with the JTT [2] rate matrix Q to calculate $|Q - Q'|$ for each resampling. The mean and standard deviation of these norms were calculated. To make the values easier to work with they were multiplied by a factor 10 000.

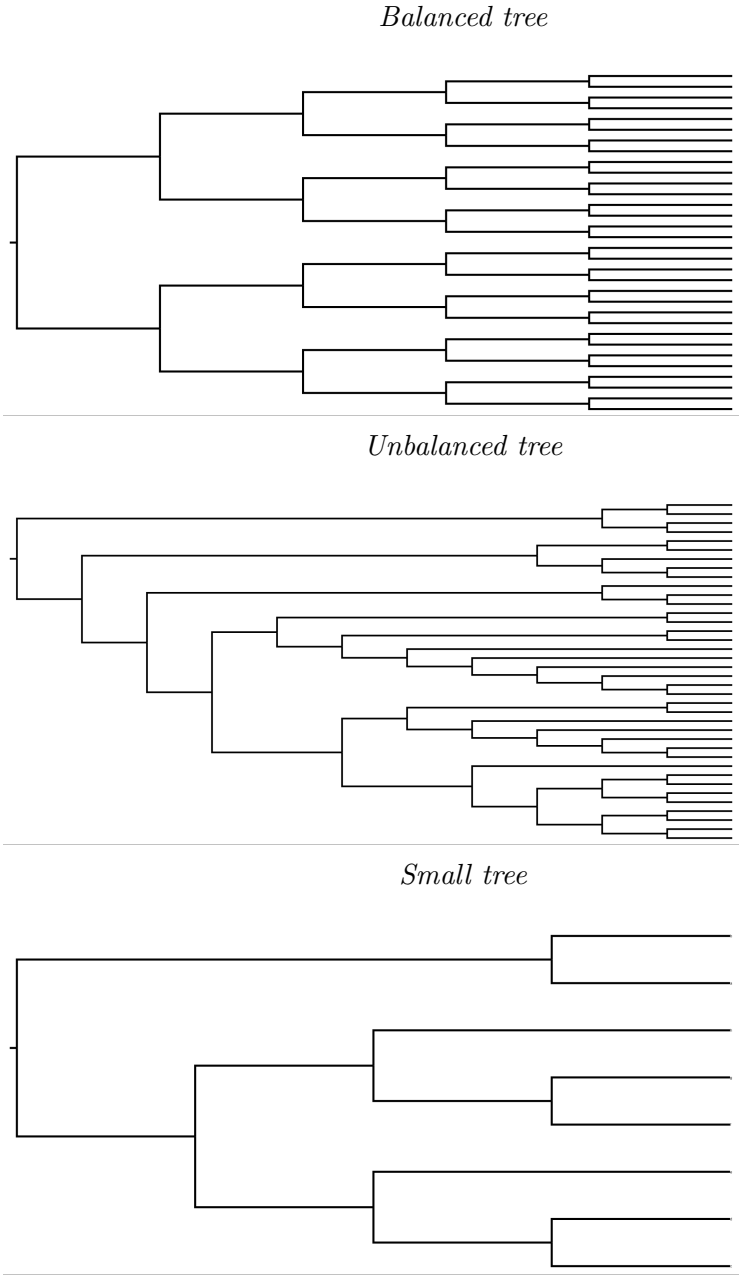

Figure 1: Trees used to generate test data. The root node to the far left represents the ancestral protein sequence. This is the protein sequence from which mutations occur to generate the leaf nodes. Each split in the tree indicates a protein sequence being duplicated into two child sequences. The leaf nodes to the far right each represent one protein sequence. The branch length here depicted as horizontal distance is not to scale.

7 Results

The data produced from the tests allowed for the program to be evaluated with respect to correctness of both the implementation of the rate matrix approximation as well as the reliability of the bootstrap norm.

7.1 Program implementation correctness

The result of the correctness testing can be seen in Figure 2. The left graph shows the mean of the calculated norms and the right the standard deviation. In all cases the program converges as would be expected for a working implementation.

The mean and standard deviation is similar for the balanced and unbalanced tree. The unbalanced tree has more nodes with 38 nodes versus 32 nodes for the balanced tree. This results in the multialignments of the unbalanced tree having more protein sequences. Despite this the program gives a better rate matrix estimation for the balanced tree and with a lower standard deviation for the produced rate matrices.

In contrast the method performs significantly worse for the small tree, with the mean difference norm for the longest sequence length being twice as high as the balanced tree. As the smaller tree has less data in its multialignment this is a reasonable result.

The standard deviation graph show a correlation similar to that of the means, with longer sequence lengths having a lower standard deviation, and the balanced tree performing best followed by the unbalanced and small tree.

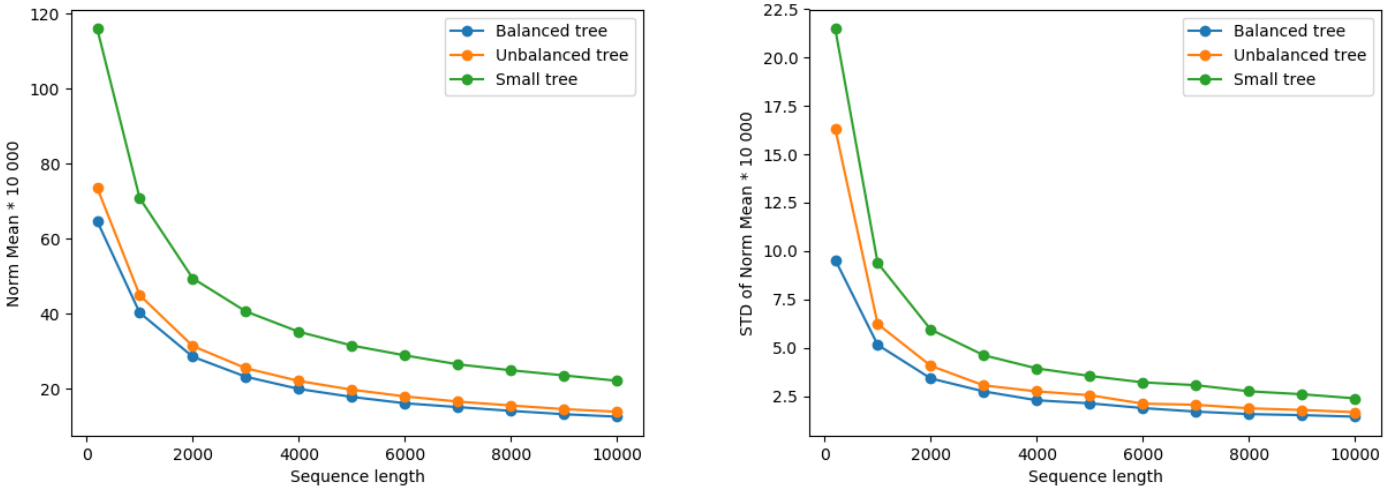

Figure 2: Mean and standard deviation of the norm $|Q - Q'|$ of 100 multialignments. Here Q is the actual rate matrix used to generate the multialignments. And Q' is the rate matrix estimation of the program. In all cases the program converges with increasing sequence length.

7.2 Bootstrap

When using bootstrapping it is always preferable to use as many resamplings as possible to get a reliable and meaningful result. The constraints are limitations in computational power and time. Therefore a compromise has to be done between better results and longer computations.

A test was done to try and evaluate how the number of resamplings affects the result of the program. To save on computational time the test was only done on a single multialignment. A random multialignment chosen from the shortest sequences for the small tree was chosen. By using the multialignment consisting of the least amount of data, and thereby in some sense the worst multialignment, the hope is that the result might better reflect a worst-case scenario of the stability.

The multialignment was repeatedly resampled and the bootstrap norm was calculated and stored. The relative standard deviation of all the bootstrap norms was calculated every time a new value was added. When the relative standard deviation changed less then 1 ‰ for the last ten values added the final relative standard deviation was stored.

The test was performed with 10, 20, 40, 80, 160 and 320 resamplings. As can be seen in Figure 3, over 160 resamplings gave a relative standard deviation under 2 %. It was decided 200 resamplings gave an acceptable compromise between stability and calculation time.

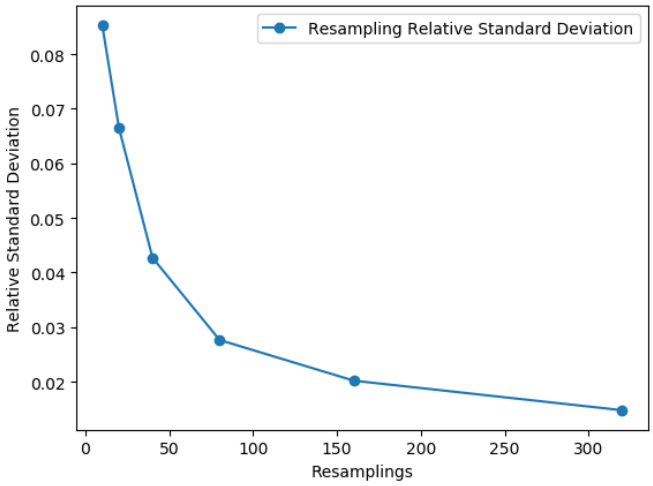

Figure 3: Relative standard deviation of the bootstrap norm for a single multialignment generated from the small tree. Results are for $10, 20, \ldots, 320$ *resamplings. Despite only having six datapoints due to heavy computational demands the pattern of decreasing relative standard deviation with increased resampling is clear.*

7.2.1 Result and reliability of bootstrap norm

After settling on using 200 resamplings the bootstrap method was tested on one random multialignment for each length of each tree.

The result can be seen in Figure 4. With only one multialignment used for each datapoint the produced graphs are uneven. The unbalanced tree even has an increase in bootstrap norm between the last two datapoints. Despite this the overall pattern of decreasing norm for the more balanced trees and long sequences is clear.

In an attempt to verify whether the bootstrap norm would give a reliable result not only for one multialignment but in general more bootstrap norms were produced and their standard deviation calculated. Due to the heavy computational demands this was only done on 4 sequence lengths for each tree with the bootstrap norm being calculated for 100 multialignments using 200 resamplings. The graph shows that the standard deviation follows the same pattern as the bootstrap norm as would be expected if the bootstrap norm decreases with increasing data.

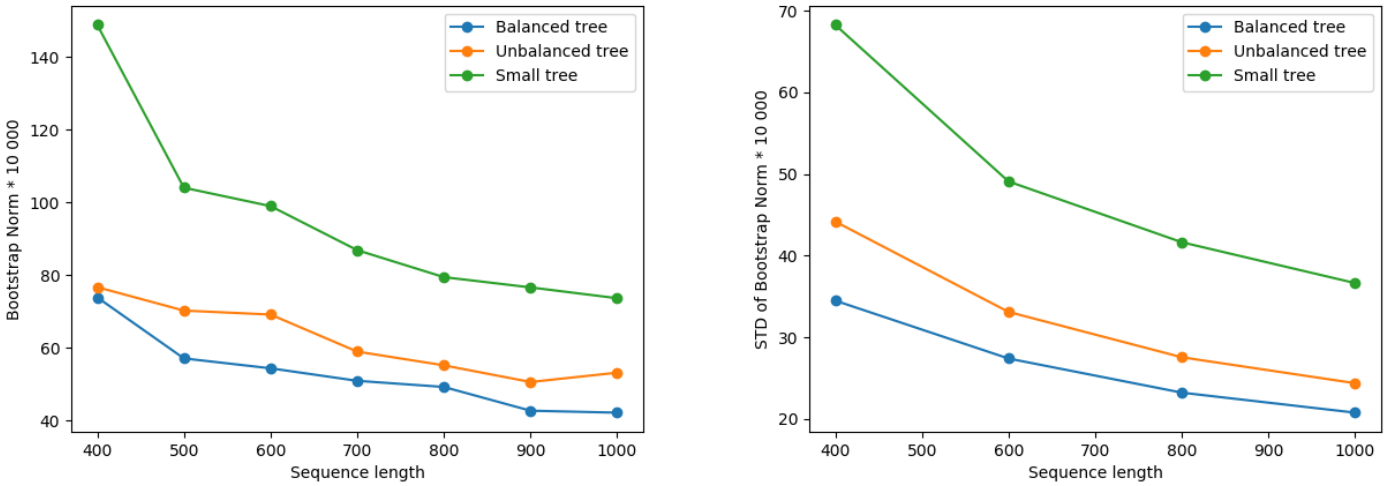

Figure 4: The left Figure shows the bootstrap norm of a single multialignment for each tree and sequence length. The right Figure shows the standard deviation of 100 multialignments for each tree and four different sequence lengths. In both cases 200 resamplings were used.

8 Conclusions

In this project the BW-method was implemented. It was tested for correctness by generating multialignments with known properties to allow for comparisons between the produced results and the desired results. These tests were used as a basis for a test suite which aims at facilitating further use and development of the software in the future.

In addition the program was given a functionality extension using bootstrapping. This aims to produce a metric that can be used to judge how stable the output of the BW-method is for a given multialignment. A lower bootstrap norm would indicate a more stable output.

It was shown that multialignments consisting of more amino acid positions did produce more stable output and that this correlated with a lower bootstrap norm. It was also shown that the multialignments from more balanced trees produced more stable output, which also correlated with a lower bootstrap norm. More research, including more computational time, would have to be done to fully verify the implementation and its results.

The program was implemented with the goal of being easy to maintain and extend in the future. To achieve this the program was implemented in modules such as the interface, input handling, bootstrap and estimation module. By having clearly defined modules with high cohesion and well defined interfaces the coupling of the program is reduced. Making future changes less likely to have unintended side effects while also making it easier to add on new functionality to the existing codebase.

The test suite was implemented with unit- and integration tests. By using unit tests the interface of individual modules can automatically be tested for correctness to verify that changes in the code maintains the integrity of the module interface.

Integration tests allow for the same kind of automated verification of the functionality of the entire program.

The program in its entirety and its source is available on GitHub [7]. It is also possible to install the program directly through the PyPi [8] database.

References

[1] Rambaut A and Grass N C. Seq-Gen: an application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees. *Bioinformatics*, 13(3):235–238, 1997.

[2] Jones D T, Taylor W R, and Thornton J M. The rapid generation of mutation data matrices from protein sequences. *Bioinformatics*, 8(3):275–282, 1992.

[3] Arvestad L. Efficient methods for estimating amino acid replacement rates. *Journal of Molecular Evolution*, 62(6):663–673, 2006.

[4] Bishop M J and Friday AE. Evolutionary trees from nucleic acid and protein sequences. *Proc. R. Soc. Lond. B*, 226(1244):271–302, 1985.

[5] Dayhoff M O. A model of evolutionary change in proteins. *Atlas of Protein Sequence and Structure*, 5:89–99, 1972.

[6] Cock P J. A., Antao T, Chang J T., Chapman B A., Cox C J., Dalke A, Friedberg I, Hamelryck T, Kauff F, Wilczynski B, and de Hoon M J. L. Biopython: freely available Python tools for computational molecular biology and bioinformatics. *Bioinformatics*, 25(11):1422–1423, 2009.

[7] Ridderström R. Modelestimator. `https://github.com/RubenRidderstrom/modelestimator-v2`, 2018.

[8] Ridderström R. Modelestimator. `https://pypi.org/project/modelestimator-v2/`, 2018.

[9] Sheldon M Ross. *Introduction to probability models*, page 393. Academic press, 2014.

[10] Henikoff S and Henikoff J G. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915–10919, 1992.