Towards Parallelising IQ-TREE

Profiling the IQ-TREE Algorithm and Devising Ways to Parallelise the Algorithm

Advanced Studies Course 2021

Zaran Zahid (U6347342)

Supervisor: Dr. Minh Bui and Dr. Giuseppe Barca

Abstract

The IQ-TREE software has been extensively used to find the phylogeny of viruses, in particular the fast-spreading SARS-CoV-2 variants that currently has more than 1.7 million sequences sampled. However, due to the number of samples increasing every day, these phylogenies need to be continually updated, resulting in a greater need for more efficient and optimised phylogenetic reconstruction software. In this paper, we run profiling on the IQ-TREE software to determine the most time-consuming functions and loops present. We found that the most computationally expensive part of the software is calculating the maximum likelihood of a tree topology, with approximately 50% of the running time spent in these parts. Upon analysing these loops, parallelizing options were discussed and presented in the paper, which can significantly speed up the computation time of the IQ-TREE software.

1.0 Introduction

Phylogenetic reconstruction plays a crucial role in studying the evolutionary relationships among organisms, enriching our understanding of how they evolve and enabling us to predict how organisms might change in the future. As Dobzhansky remarked in his famous paper [9], “nothing in biology makes sense in the light of evolution”. This is particularly important for fast-spreading viruses such as the SARS-CoV-2 variants to identify and track emergent strains. The information can also provide useful insights on how transmission occurs compared to typical contact tracing [1].

Several fast phylogenetic reconstruction algorithms exist for large phylogenomic data and of the most popular algorithm is included in IQ-TREE [2], a widely used, open-source software package that infers the best phylogenetic tree based on its maximum likelihood [3]. However, as the sequences obtained for SARS-COVID-2 numbers to around 1.7 million and 10,000 sequences are being added each day, researchers need to update their analyses daily during the global pandemic. With such large datasets, there is an urgent need for more efficient phylogenetic reconstruction algorithms and further optimising existing algorithms. In this report, we profile the IQ-TREE algorithm to locate the key bottlenecks of the algorithm presented in the IQ-TREE software. With this information, the algorithm can be accelerated using parallelisation, thereby enabling researchers to reconstruct larger and more complex trees within record execution times. We perform the profiling on the GADI supercomputer, one of the most powerful supercomputers in the world to ensure that the obtained results are not limited by the hardware.

Project Aims

The overarching aim of this research is to profile the IQ-TREE software and determine what parts can be optimized to make the phylogenetic reconstruction run faster. This can be categorized into three specific objectives:

(i) To develop and write scripts in Bash to run the IQ-TREE software on a supercomputer. The script should load all the modules needed to run the software with a single command. This would allow the testing to be easier and have the algorithm running on a system designed for large-scale calculations.

(ii) Run profiling on the IQ-TREE software to identify the bottlenecks in the reconstruction algorithm. This should be performed on the GADI supercomputer using Intel Advisor, a profiling software that can also provide inform. The profiling will be run on two separate

datasets, one a lot larger than the other. There will also be different number of threads used to determine the potential bottleneck shifts determined by shared-memory parallelism.

(iii) Understand the bottleneck of the code and analyse why this is taking the most time. Furthermore, devise ways to speed up the bottleneck using parallelization.

1.1 Phylogenetics

Phylogenetics is the study of evolutionary relationships among biological entities, which can be species, individuals, or genes. The main aim of phylogenetic reconstruction is to describe evolutionary relationships in terms of relative recency of common ancestry [4]. They are important for addressing various biological questions such as relationships among species or genes, the origin and spread of viral infection and the demographic changes and migration patterns of species [5]. They are represented as evolutionary trees which record the similarity between different organisms. The similarity of organisms can be based upon several different factors, including the appearance of organisms, shared traits and more recently the genomic data of the organism. As the DNA sequence of an organism contains this genetic information, organisms with similar DNA sequences are more related to each other and have a shared ancestor that is more recent in time compared to other organisms. The process of converting a set of sequences into a phylogenetic tree is shown below in Figure 1. The internal nodes in the tree represent the shared ancestors of the current organisms.

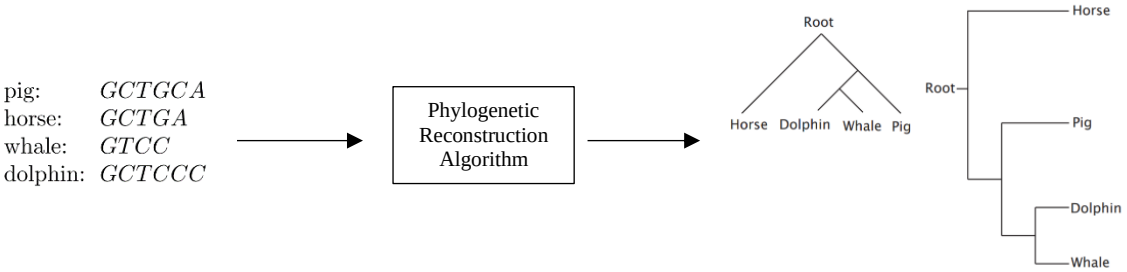

Figure 1: The process of converting a set of DNA sequences (left) into a phylogenetic tree (right). The DNA sequences for different species are first aligned, and alignments with the most similarity are more related to each other. Note: DNA sequences are comprised of four nucleotides represented by the letters A,C,T,G

The purpose for phylogenetic reconstruction is to take in a set of sequences that encode the genetic information of organisms as its input and return the best tree topology depicting the evolutionary relationship between the organisms. To achieve this, there first needs to be a model for the mutation rates that occur in ancestors resulting in the divergent species. These are known as substitution model or models of DNA sequence evolution.

1.2 Models of DNA Sequence Evolution

Evolutionary models describe the substitution process in genetic sequences over time. Once DNA sequences of several organisms are aligned, there will be sites in the sequences that differ. This is due to random mutations occurring in the ancestral sequence, causing the current sequences to diverge. The mutations include insertions, deletions and substitutions of nucleotides present at a specific site. Here we will only deal with substitutions. The rate at which these random substitutions occur are modelled using a continuous-time Markov model, depending on some time parameter t and the states being the letters of the DNA alphabet $\Sigma = \{A, C, G, T\}$. The transition probabilities, that is, the probabilities of converting from one nucleotide to another, are then stored in a matrix such as shown below:

$$P(t) = \begin{pmatrix} p_{AA}(t) & p_{AC}(t) & p_{AG}(t) & p_{AT}(t) \\ p_{CA}(t) & p_{CC}(t) & p_{CG}(t) & p_{CT}(t) \\ p_{GA}(t) & p_{GC}(t) & p_{GG}(t) & p_{GT}(t) \\ p_{TA}(t) & p_{TC}(t) & p_{TG}(t) & p_{TT}(t) \end{pmatrix}.$$

In regard to a phylogenetic tree, the tips of the tree represent the genomic data of an organism, and the branch lengths is essentially the average of the parameter t needed to convert from the ancestor organism to the current one.

There are several substitution models used in phylogenetic reconstruction. The simplest model is the Jukes-Cantor model [5], which assumes that the substitution rate between two nucleotides is the same. More complicated and generalised substitution models exist, such as the Felsenstein model [6] and the Kimura Model [7], with the most complex model being the general time-reversible (GTR) model. These models differ based on the number of parameters in the model. For example, the Jukes-Cantor model uses one parameter representing the substitution rate α that is the same for all pairs of nucleotides The GTR model assumes different rates of substitutions for each pair of nucleotides as well as assuming different frequencies at which the nucleotides occur. The IQ-TREE software uses ModelFinder [7], a model-selection method to find the best substitution model that fits with the tree and data

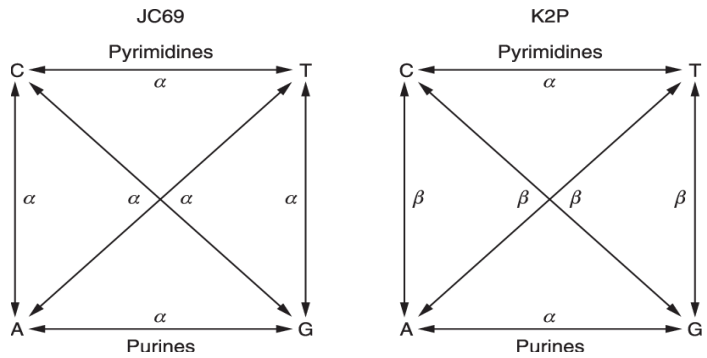

Figure 2: The Jukes-Cantor Model (left) and the Kimura Model (Right) shown with the different parameters used to depict the substitution rates [10]

1.3 Maximum Likelihood

Once a model has been chosen and sequences corresponding to different organisms are aligned, a phylogenetic tree can be reconstructed. To determine the tree topology that best explains the multiple sequence alignment, the maximum likelihood principle is utilised. Likelihoods provide the conditional probability of observing the multiple sequence alignment data given a tree topology. The higher the likelihood, the more preferred a tree topology is compared to other topologies. The likelihood function can be written as:

$$P(D|T)$$

Where D is the phylogenetic data and T is tree topology. The aim is to maximise this probability by selecting the best T, that is, selecting the best tree topology.

Two important assumptions are needed to compute the likelihood of a given tree topology:

1. Evolution in different sites is independent.
2. The evolution of each site along each branch of the tree does depend on its evolution along any other branch.

The first assumption allows us to find the likelihood for each site and multiply these together to find the overall likelihood of the data. That is, we can write:

$$\Pr(D \mid T) = \prod_{i=1}^{m} \Pr\left(D^{(i)} \,\middle|\, T\right) = \Pr\left(D^{(1)} \,\middle|\, T\right) \Pr\left(D^{(2)} \,\middle|\, T\right) \cdots \Pr\left(D^{(m)} \,\middle|\, T\right)$$

where $D(i)$ is the data at the ith site, as shown in the diagram below.

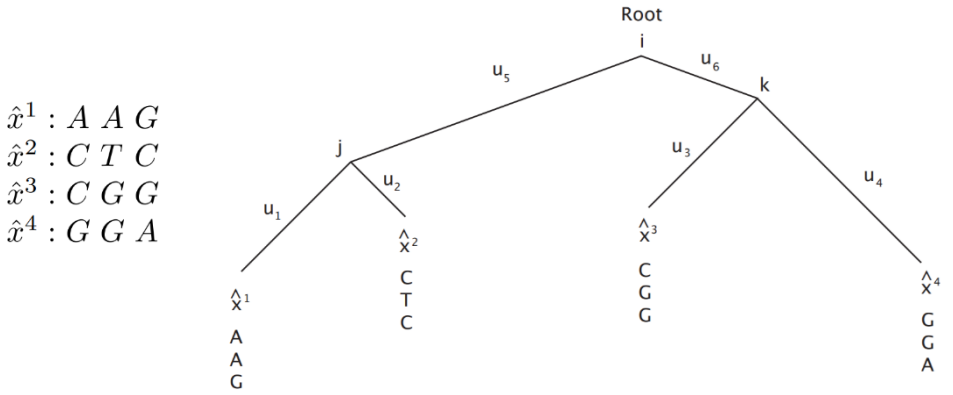

Figure 3: Am example to illustrate the Maximum Likelihood principle. Three sequences x1,..x3 are shown on the left and a tree has been initialised with branch lengths u1,.. u4. The internal nodes have also been labelled with variables i,j,k.

Next, to calculate the likelihood of a site, we simply sum over all the possible internal nodes that give the tips of the tree:

$$\Pr(D^1|T) = \sum_i \sum_j \sum_k \Pr(A, C, C, G, i, j, k)$$

The second assumption that evolution in different lineages are independent allows us to decompose the right hand side of the above equation into a product of terms, where only the relevant nodes down a branch are used to calculate the events happening in that branch.

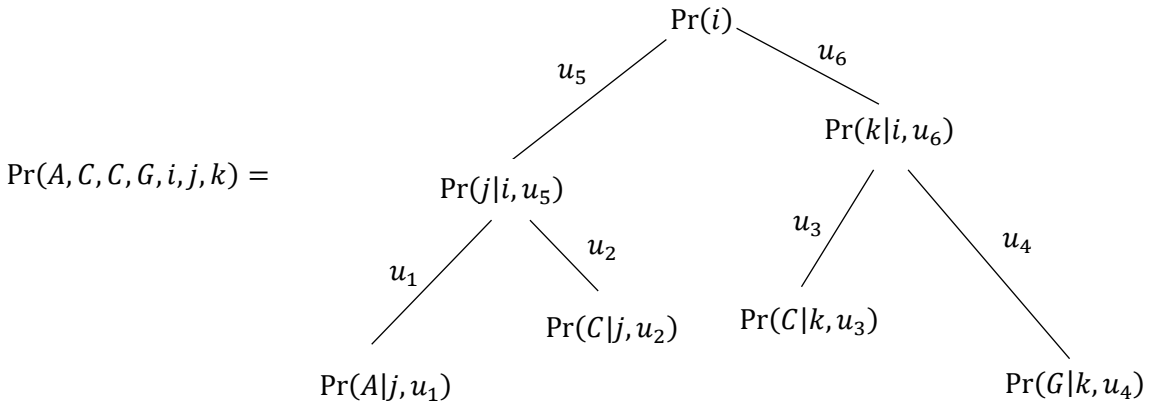

Figure 4: Calculating the probability is only dependent on the branch lengths leading to the node and the ancestor node. Thus we find that $Pr(A, C, C, G, i, j, k) = Pr(i)\ Pr(j|i, u_5)\ Pr(k|i, u_6)\ Pr(A|j, u_1)\ Pr(C|j, u_2)\ Pr(C|k, u_3)\ Pr(G|k, u_4)$

Calculating this over all the internal nodes will give the likelihood of a specific site and once all the site likelihoods are calculated, the overall likelihood can be calculated by multiplying each of the site likelihoods together.

1.4 Felsenstein Pruning Algorithm

To economise the computation for calculating the likelihood of a tree, dynamic programming can be utilised, where the conditional likelihood of a subtree of the whole tree is calculated. The conditional likelihood $L_k^{(i)}(s)$ represents the probability of everything observed from a node k down to its descendants, conditional of node k having state s [6]. The process is explained using the example data and tree shown in Figure 3. First the tips of the tree are initialised so that $L(s) = 1$ at a tip node if s is the state that is observed at that tip, and 0 otherwise. For example, in the left-most branch, the nucleotide A is observed for the first site. Thus:

$$\langle L^1(A), L^1(C), L^1(G), L^1(T)\rangle = \langle 1,0,0,0\rangle$$

This is the initialisation step in the algorithm. The recursive step involves calculating the conditional likelihood of the immediate ancestor of the tip nodes. In the above example, j and k are such nodes. The

conditional likelihood for any node a is given with descendants b and c (connected to node a by t_b, t_c branch lengths respectively) is given by:

$$L_a^i(s) = \left(\sum_x \Pr(x|s, t_b)\, L_b^i(x)\right)\left(\sum_y \Pr(y|s, t_c)\, L_c^i(y)\right)$$

That is, we simply take the product of the events taking place in the descendants of the current node a. For example, to calculate $L_j^i(s)$ where j is the left internal node in Figure 3, it will result in:

$$L_j^i(s) = \Pr(A|j, u_1)\Pr(A|C, u_2)$$

The algorithm keeps recursing up the tree in this fashion until the root node is reached with the likelihood $L_0^i(s)$ for every base s. To complete the likelihood calculation, we find the weighted average of every base s, weighted by the prior probability.

The pruning algorithm is the most computationally expensive part of finding the best tree topology using maximum likelihood, requiring several recursive calls down each branch of the algorithm and several calculations that can be done using dot products.

1.5 Finding the Best Tree Topology: Approaches

With the maximum likelihood principle, we have a way to infer whether one tree is better than another. Using this principle, the best tree topology for a dataset can be found by looping through all possible trees, calculating the total likelihood of each. Then a list can be constructed, storing the best trees currently found and updating the list as more trees are evaluated. However, this brute-force approach to find the best tree can only be done when the set of tree space is small. The problem of finding the maximum likelihood tree is known to be an NP-hard, and so as the number of species in the dataset increases, the tree space becomes too large to enumerate completely. Therefore, the "best" tree topology has to be determined using heuristics and searching through a subset of the possible trees.

1.6 Algorithm in IQ-TREE

The flowchart of the algorithm used in the IQ-TREE software is shown below in Figure 6. A candidate set is initialised, holding the top ten trees found using an efficient and fast search (Figure 6a). A tree is randomly chosen from this set, and Nearest Neighbour Interchange (NNI) is applied to it (Figure 6b), which exchanges the connectivity of subtrees within the tree, as demonstrated in Figure 5. Other random perturbations, which

slightly change the current tree, are also performed. This results in a new tree which can either be better or worse than the original tree.

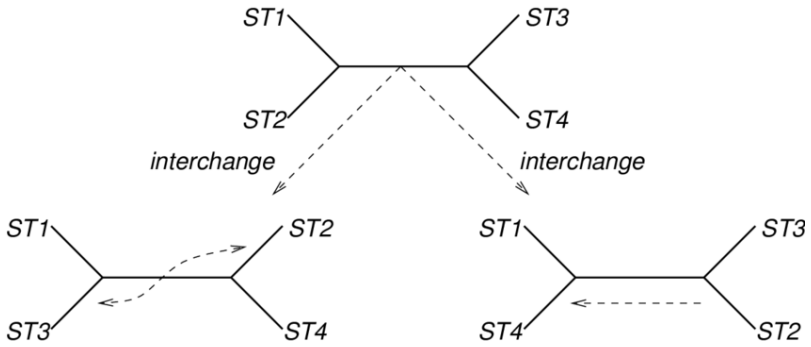

Figure 5: Nearest Neighbour Interchange [8]. Two subtrees are swapped resulting in a slightly different tree. This introduces stochasticity, which can be used to find a find a better tree topology when we already have a tree.

If the new tree has a higher likelihood than any of the trees in the candidate set, then this tree is added and the worst tree is removed from the candidate set (Figure 6d). This process is repeated until the candidate set remains unchanged for 100 iterations, after which the best tree is returned in the algorithm.

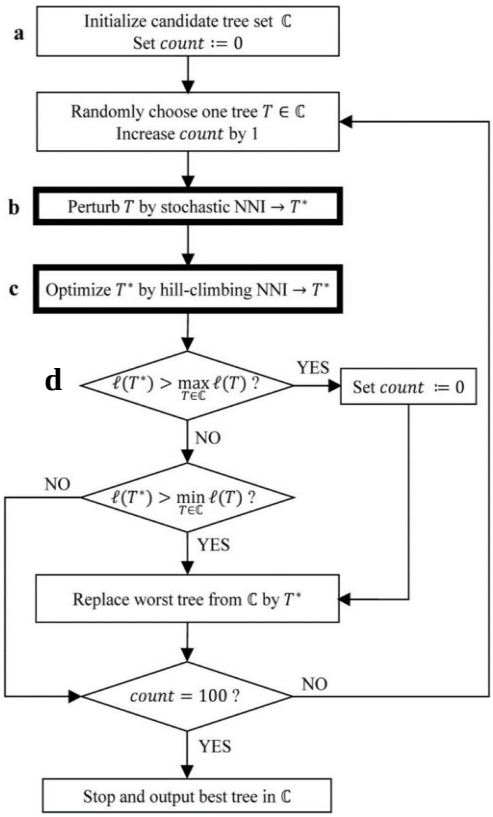

Figure 6: The phylogenetic reconstruction algorithm used in the IQ-TREE software [2].

1.7 Other ML tree search algorithms

Introducing random perturbations are important parts of many ML tree searches to find a better tree from the trees already found. Other popular phylogenetic reconstruction algorithms, such as those implemented in RAxML 8 [11], only consider nearest neighbour interchanges that result in a better tree topology, known as “hill-climbing NNIs” as they reach a local optimum in the tree space. However, these algorithms are prone to be stuck in local optima [12]. The strength of the IQ-TREE algorithm is that is introduces stochastic NNI steps that allow “down-hill” moves, which decrease the likelihood of new trees. For these perturbed trees, the hill-climbing NNI is applied and due to the random perturbations done earlier, the local optimum can be avoided.

2.0 Methods and Materials

2.1 Hardware Specifications

I performed the following analyse on the GADI supercomputer, with 100 GB and 48 Intel(R) Xeon(R) Platinum 8268 CPU @ 2.90GHz CPUs. The supercomputer was accessed through a Linux Distribution which had the latest version of Intel Advisor installed.

2.2 Developing Scripts to Run

First the following modules were loaded into Gadi using the “module load” command. To load the modules efficiently, a bash file was created storing all the modules needed. The list is shown below:

```
#!/bin/sh

module load boost
module load openmpi
module load eigen
module load cmake
module load intel-advisor

export ADVIXE_EXPERIMENTAL=int_roofline
export OMP_NUM_THREADS=12
```

This script was stored as a vim file called “modules.sh” and was placed in the /.bashrc file so that it ran whenever the supercomputer was accessed.

The modules such as boost and eigen were used by the IQ-TREE algorithm for calculations. The intel-advisor module was needed to run the profiler on the software. The CMake library was used to build the IQ-TREE.

To build the IQ-TREE software, the following command was ran in a build directory:

```
cmake ../iqtree2 -DEIGEN3_INCLUDE_DIR=/apps/eigen/3.3.7
```

This command will create the binary file that runs the IQ-TREE software. Then the command:

```
make -j
```

Was ran to make the code executable. Once the preceding command is run, there is no need to run the above code again.

2.3 Running the Profiler

To run the profiler, the following command was ran where the binary file "iqtree2" is kept:

```
advixe-cl --collect survey --project-dir ./adv_results --search-dir src:=../ -
-search-dir bin:=./ -- ./iqtree2 -s example/example.phy -redo -T 4
```

This will produce the results of the profiling in a folder called

```
-- project-dir ./adv_results
```

Which will be either created or overwritten when run. Therefore, changing the directory to something else is recommended to ensure that the data is not overwritten.
To vary the results, the dataset was changed to the turtle.fa dataset accessed from the IQ-TREE

2.4 Accessing the Profiling Results

The profiling results were accessed by running the command:

```
advixe-gui
```

This would open the Graphical User Interface of the Intel Advisor installed on the home computer. The adv_results folder was then opened to view the results of the profiling in the graphical user interface.

3.0 Results and Discussion

This section reports the results obtained from the running the IQ-TREE algorithm with profiling on two different datasets. The first dataset is the "example.phy" (Example) dataset, comprised of parts of the mitochondrial DNA sequences of 17 species with approximately 2000 sites [13]. The name of the species and the first 60 sites are shown in Figure 7. This is a relatively small dataset compared to the millions of sites present in some sequences. The other dataset was the "turtle.fa" (Turtle) dataset, containing 16 species and over 20,000 sites, which is 10 times larger than the Example dataset and defines 29 genes. This dataset was used to determine the evolutionary relationship between turtles, birds, and reptiles [14]. Although still small compared to other datasets, the Turtle dataset took several minutes at a time to produce the profiling results. The name of the species and the first 60 sites are shown in Figure 8.

Figure 7: The first 60 sites of the Example dataset. In total, there are 17 species (sequences) and 2000 sites.

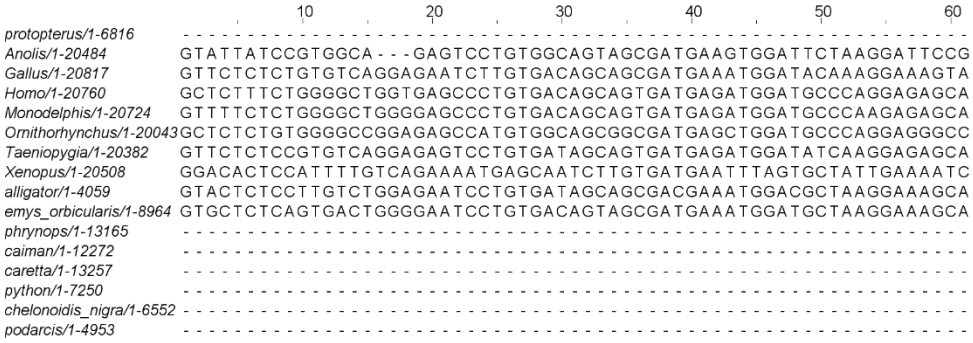

Figure 8:The Turtle dataset. In total, there are 16 species and over 20000 sites. This dataset is significantly larger than the Example dataset

3.1.1 Summary of the Total Time Spent for every Dataset and Thread Number

Figure 9 below records both the total elapsed time and the CPU time spent for the software to produce the maximum-likelihood tree. The shortest elapsed time for each dataset was observed with 4 thread. These times were 9.98 s and 265.89 s for the Example and Turtle dataset respectively.

Dataset	Number of threads	Total elapsed time (s)	Total CPU time (s)	Number of vectorized Loops	Time in Vectorized loop (s)
Example Dataset	**1**	12.95	12.77	4	1.57
	2	11.71	22.77	3	2.10
	3	11.05	30.94	3	2.86
	4	9.98	35.43	4	3.19
Turtle Dataset	**1**	611.39	610.49	4	19.04
	2	736.83	735.68	2	21.01
	3	265.89	1058.21	4	72.69
	4	265.89	1058.21	4	72.69

Figure 9: Table of results summaring the total time spent for each experiment, along with the number of vectorised loops detected by the profiler and the time spent in these vectorised loops.

3.1.2 Top 5 Places Where the Code Spent the Most Time in Example.phy (1 Thread)

The results below show the top 5 most time-consuming loops on the Example dataset with 1 thread being used. Every loop is in the file “phylokernelnew.h” and considering the total CPU time used (12.77 seconds from Figure 9), these loops contribute to 40% of the total time.

The most time-consuming is at line 684. After these loops, the next time-consuming loop drops significantly by a half. The locations are at lines 1713 and 1818 in the file.

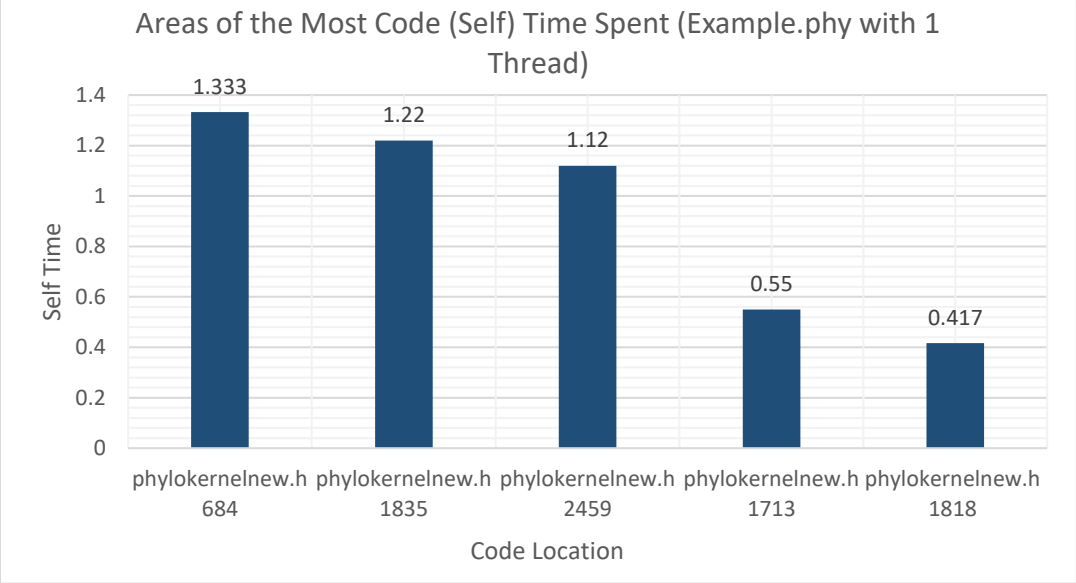

Figure 10: Top-5 time-consuming loops on the 'example.phy' dataset when 1 thread is specified. Each loop is located in the phylokernelnew.h file, which calculates the maximum likelihood of a given tree. The respective percentages are:

The loop at line 684 involves the calculation of three dot products. This loop calculates three dot products, between vectors A and D, B and D, and C and D. A helper function is used to accumulate the dot product values since first the two corresponding entries of each vector is multiplied and then these values are added together for every entry. The other loops measured all related to calculating the likelihood of the tree.

The variable ‘block’ must be significantly large and the function is called several times for this to explain why this part takes a large amount of time.

The loops at line 1713 and 1818 are used to initialize the tip cases before calculating the partial likelihood of a branch. Line 1713 handles initialisation of cherry cases in the tree, whereas the loop at line 1818 handles the internal node to tip case. The code for these sections have been omitted due to their long length. Figure 14 illustrate the parts of a tree that they affect.

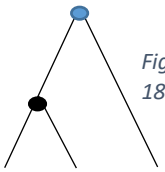

Figure 11: The loop at 1713 initialises cherry nodes (black circle). The loop at 1818 initialises internal-tip nodes (blue circle)

3.1.3 Top 5 Places Where the Code Spent the Most Time in Example.phy (2 Thread)

Figure 12 shows the amount of time spent for each location in the Example dataset with 2 threads. The results are similar to the first thread, with the first 3 loops being the same and the last two also being the same but with their orders reversed.

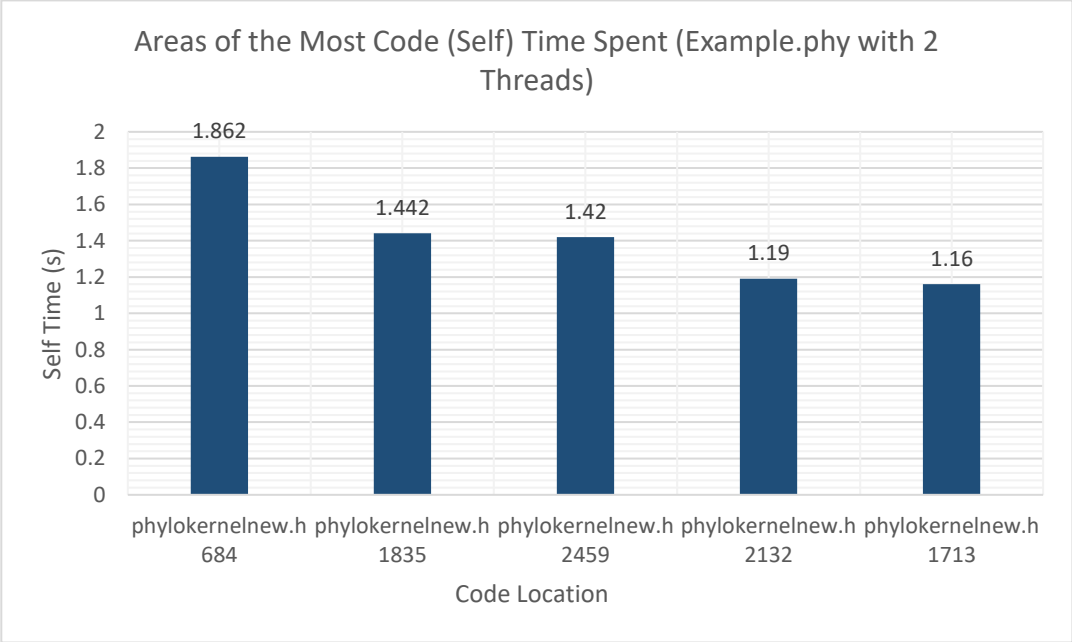

Figure 12: Time spent at each loop for Example dataset with 2 threads

3.1.4 Top 5 Places Where the Code Spent the Most Time in Example.phy (3 Threads)

Figure 12 shows the amount of time spent for each location in the Example dataset with 3 threads. Now the loop at 1713 is second most time-consuming loop. As explained earlier, this loop initializes the cherry tips of the pruning algorithm. The dot product calculations still remain the most time-consuming code.

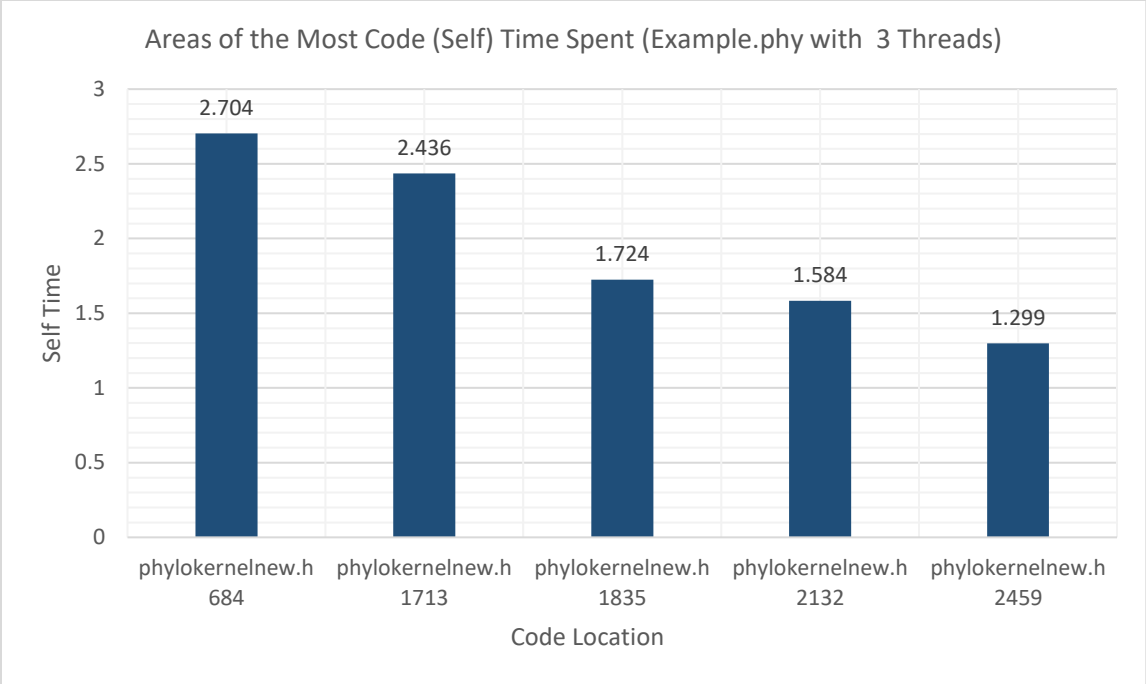

Figure 13: Top-5 time-consuming loops on the 'example.phy' dataset when 3 threads are specified.

3.1.5 Top 5 Places Where the Code Spent the Most Time in Example.phy (4 Threads)

Figure 15 shows the amount of time spent for each location in the Example dataset with 4 threads. The results are almost identical when running the experiment with 3 threads. However, the gap between the top 2 most time-consuming threads has narrowed.

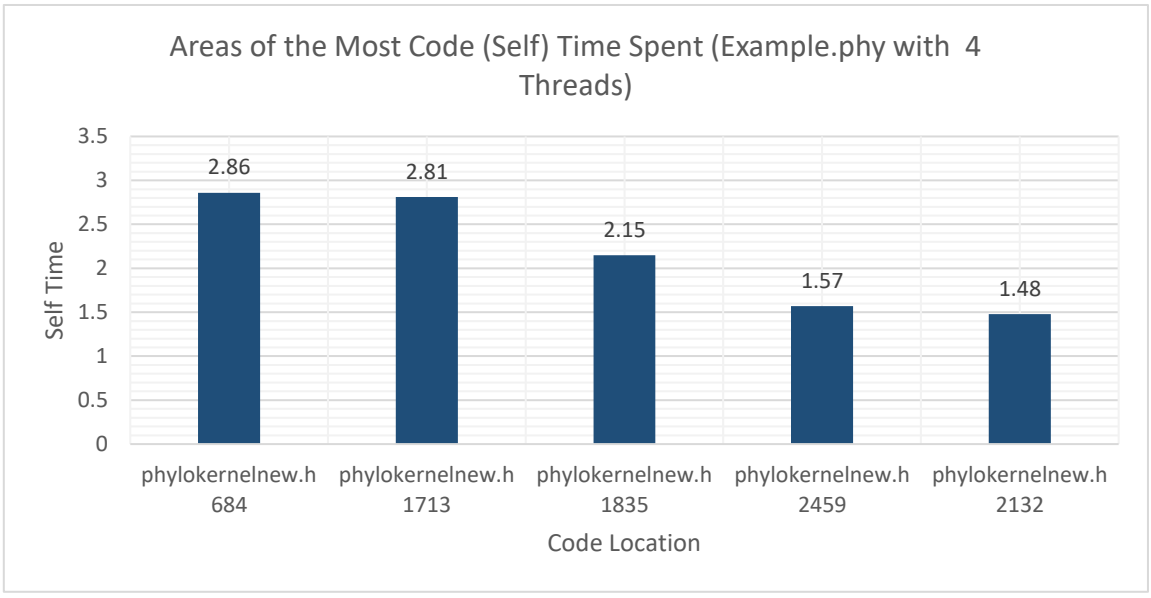

Figure 14: Top-5 time-consuming loops on the 'example.phy' dataset when 4 threads are specified.

3.1.6 Top 5 Places Where the Code Spent the Most Time in Turtle.fa (1 Thread)

Figure 15 shows the amount of time spent for each location in the Example dataset with 4 threads. The loop at 1713 has now become the most time-consuming loop and the dot product calculation function is no more present in the top 5 time consuming loops.

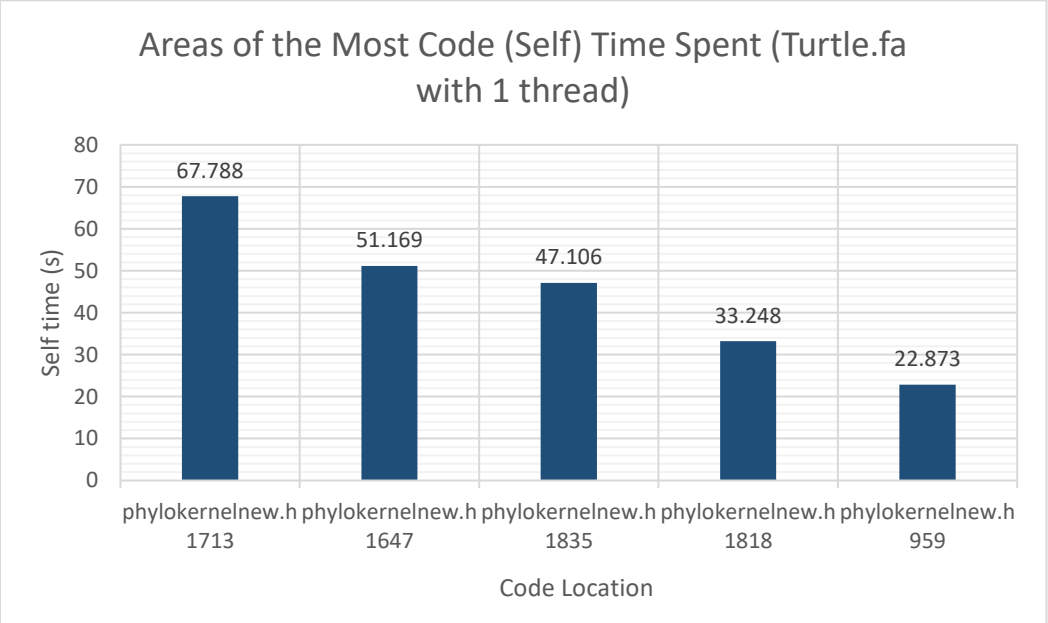

Figure 15: Top 5 time-consuming loops in Turtle dataset when 1 spread is specified

3.1.7 Top 5 Places Where the Code Spent the Most Time in Turtle.fa (2 Threads)

Figure 16 shows the amount of time spent for each location in the Example dataset with 2 threads. The results are similar to the one-thread case, with the top 3 loops remaining the same.

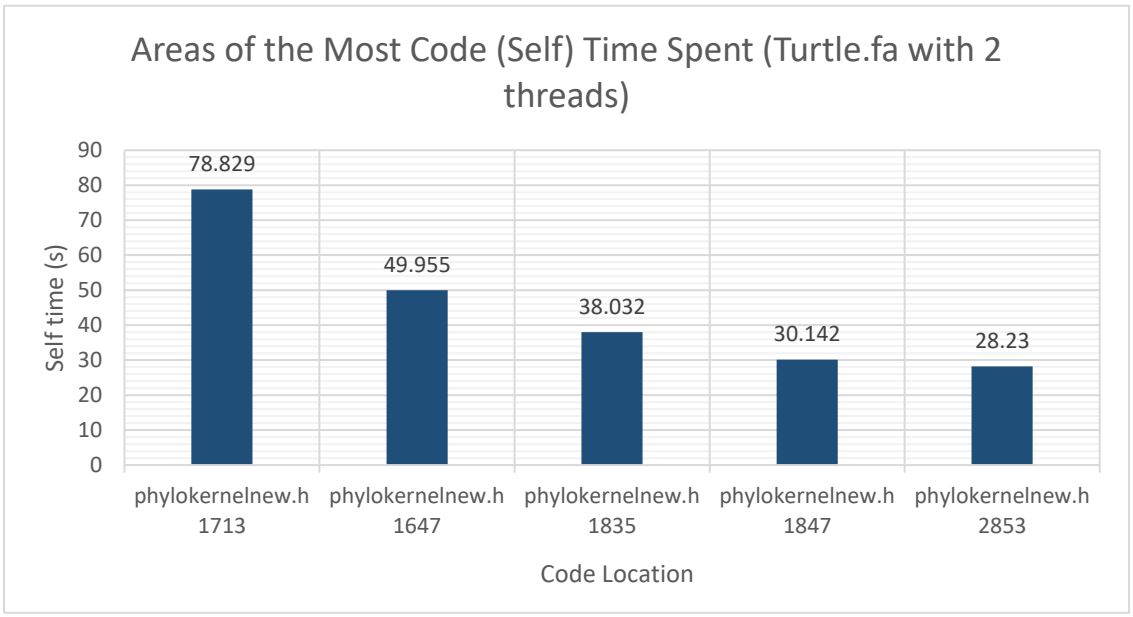

Figure 16: Top 5 time-consuming loops for Turtle dataset with 2 threads

3.1.8 Top 5 Places Where the Code Spent the Most Time in Turtle.fa (3 threads)

Figure 17 shows the amount of time spent for each location in the Example dataset with 3 threads. The top time-consuming loop remains the loop at 1713, with the difference between other loops now widening. This indicates that the loop at 1713 is most likely the bottleneck of the IQ-TREE software.

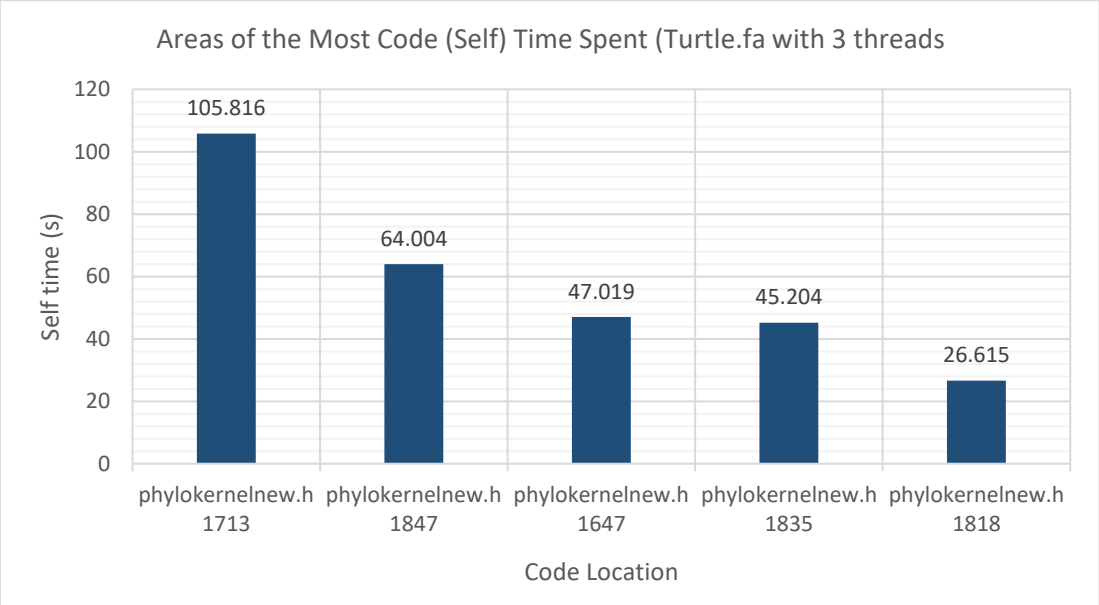

Figure 17: Top 5 time-consuming loops for Turtle Dataset with 3 threads

3.1.9 Top 5 Places Where the Code Spent the Most Time in Turtle.fa (4 Threads)

Figure 17 shows the amount of time spent for each location in the Example dataset with 4 threads. There is now a clear distinction between the most time-consuming loop, with the other loops taking a similar amount of time. The loop at 1713 takes 10% of the total CPU time, using the results in Figur

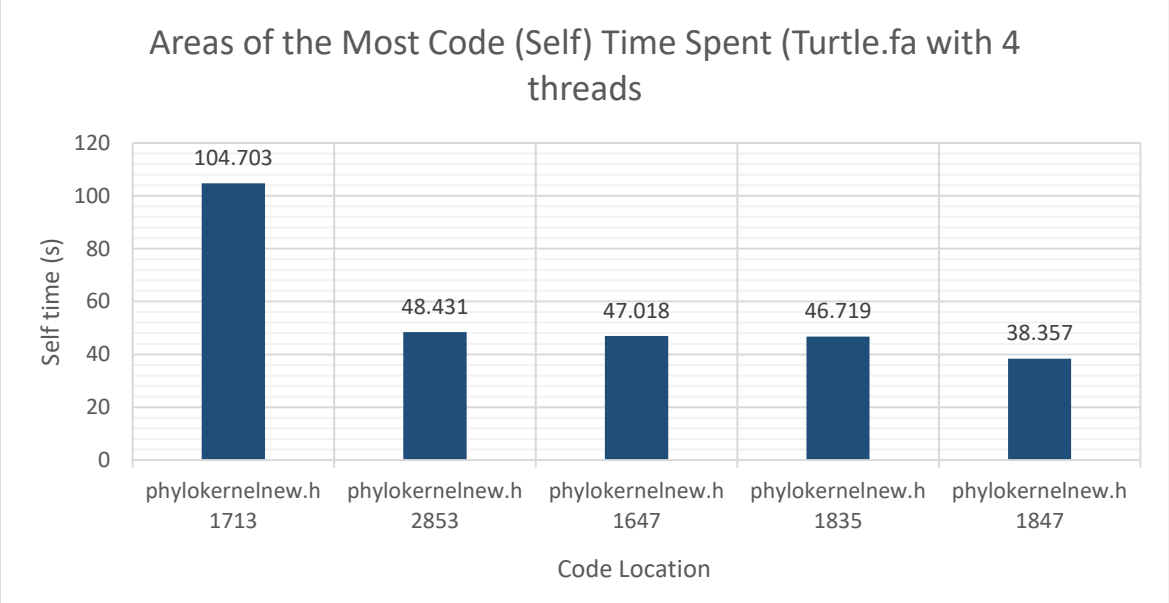

Figure 18: Top 5 time-consuming loops for Turtle Dataset with 4 threads

3.2 Discussion

From the results obtained, the overall most time-consuming loops are shown in Figure 3. Each loop is located in the “phylokernelnew.h” file, which is predominantly used to calculate the maximum likelihood of a given tree topology.

It was found that the function of the most time-consuming loops can be classified into two groups. The first group includes calculating several dot products. These loops are at lines 684 and 2853 and can be parallelized most effectively as each iteration can be done independently in a sub-thread, with the results combined in the end. The other loops seem to be used for initialisation the tips of the tree, such as the loop at 1713 and 2459. As subtrees with none of the same nodes have no relationship with each other, initializing the tips can be done independently until they start to share the same nodes.

4.0 Future Work and Conclusion

The most time-consuming loops have been found and analyzed to determine the effectiveness of optimizations that can be applied to each section. Future studies include implementing ways to optimize these loops and testing to check that the software does indeed run more efficiently. In this experiment, a small amount of threads were used. In order to fully utilize the supercomputing power, more threads should be used and the datasets that are used can be much larger to simulate the large-scale data normally given as input into the IQ-TREE software.

In conclusion, it was found that all the time-consuming loops were present in a single file, “phylokernelnew.h”, which calculates the maximum likelihood of a given tree topology. The function of the loops varied from initializing the tips of the tree to calculating relevant dot products that are needed for the maximum likelihood estimation. These loops were analyzed and it was found that the iterations of these loops are independent of each other, indicating that parallelization is highly feasible and should speed up the software. Future studies have been discussed, including implementing features to optimize the bottleneck, and testing the improved software on larger datasets.

5.0 Acknowledgements

I would like to thank Dr. Minh Bui and Dr. Guiseppe Barca for offering this enjoyable project in Semester 1 and their guidance throughout the project. I would like to also thank Fazeleh Kazemian for assisting me with using the GADI supercomputer.

6.0 References

1. Hodcroft, E.B., De Maio, N., Lanfear, R., MacCannell, D.R., Minh, B.Q., Schmidt, H.A., Stamatakis, A., Goldman, N. and Dessimoz, C., 2021. Want to track pandemic variants faster? Fix the bioinformatics bottleneck.
2. Nguyen, L.T., Schmidt, H.A., Von Haeseler, A. and Minh, B.Q., 2015. IQ-TREE: a fast and effective stochastic algorithm for estimating maximum-likelihood phylogenies. Molecular biology and evolution, 32(1), pp.268-274.
3. Minh, B.Q., Schmidt, H.A., Chernomor, O., Schrempf, D., Woodhams, M.D., Von Haeseler, A. and Lanfear, R., 2020. IQ-TREE 2: New models and efficient methods for phylogenetic inference in the genomic era. *Molecular biology and evolution*, *37*(5), pp.1530-1534.
4. Jill Harrison, C. and Langdale, J.A., 2006. A step by step guide to phylogeny reconstruction. The Plant Journal, 45(4), pp.561-572.
5. Yang, Z. and Rannala, B., 2012. Molecular phylogenetics: principles and practice. Nature reviews genetics, 13(5), pp.303-314.
6. Felsenstein, J. Evolutionary trees from DNA sequences: A maximum likelihood approach. J Mol Evol 17, 368–376 (1981). https://doi.org/10.1007/BF01734359
7. Kimura, M., 1981. Estimation of evolutionary distances between homologous nucleotide sequences. Proceedings of the National Academy of Sciences, 78(1), pp.454-458.
8. Stamatakis, A., 2004. Distributed and parallel algorithms and systems for inference of huge phylogenetic trees based on the maximum likelihood method (Doctoral dissertation, Technische Universität München).
9. Dobzhansky, T., 2013. Nothing in biology makes sense except in the light of evolution. The american biology teacher, 75(2), pp.87-91.
10. Egan, A.N. and Crandall, K.A., 2006. Theory of phylogenetic estimation. Evolutionary Genetics: Concepts and Case Studies, 1, pp.426-436.
11. Stamatakis, A., 2006. RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. Bioinformatics, 22(21), pp.2688-2690.

12. Swofford DL, Olsen GJ. 1990. Phylogeny reconstruction. In: Hillis DM, Moritz C, editors. Molecular systematics. Sinauer Associates:Sunderland (MA). p. 411–501.
13. Salemi, M., Vandamme, A.M. and Lemey, P. eds., 2009. The phylogenetic handbook: a practical approach to phylogenetic analysis and hypothesis testing. Cambridge University Press.
14. Chiari, Y., Cahais, V., Galtier, N. and Delsuc, F., 2012. Phylogenomic analyses support the position of turtles as the sister group of birds and crocodiles (Archosauria). Bmc Biology, 10(1), pp.1-15.