Stockholms universitet

Phylogenetic Tree Extraction Through the Application of Pixel Neighborhoods

Axel Brandberg

Supervisor: Lars Arvestad
Examiner: Marc Hellmuth

Kandidatuppsats i datalogi
Bachelor Thesis in Computer Science

August 2021

Abstract

There exists many phylogenetic trees in the literature, but in many cases these are only accessible in the form of an image. The reader of a scientific paper may wish to perform some tests of their own, or want to perform a control study to confirm or contradict the results. For these types of problems, a way to easily extract information from trees is useful. There exist one program TreeSnatcher which works semi-automatically by requiring users to click branches. There exists another program PhyloPhaser which sought to fully automate this through machine learning. This paper studied a method for extracting the tree from a raster image containing a fully resolved phylogenetic tree. The results shows a flawless execution under ideal circumstances, where the quality of images is guaranteed. The tests on graphs from scientific papers shows difficulties, when applied to circular trees. The best of these results came from the rectangular phylogenetic trees where a large majority worked flawlessly after some small modifications. The method was tested also on non-binary trees, in which case trees could still be extracted, but naturally with some errors. We found that the algorithm is feasible but the usefulness depends on the quality of images used and the amount of manual labor allowed in the users specific project.

Sammanfattning

Det finns många fylogenetiska träd i litteraturen, men i många fall är dessa endast tillgängliga i form av en bild. Läsaren av en vetenskaplig artikel kan vilja utföra egna tester eller uppföra en kontrollstudie för att bekräfta eller bestrida resultaten. För dessa typer av problem är ett sätt att enkelt extrahera information från träd användbart. Det finns ett program TreeSnatcher som fungerar halvautomatiskt och kräver att användare klickar på grenarna i grafen. Det finns ett annat program PhyloPhaser som försökte automatisera detta fullt ut genom maskininlärning. Detta dokument studerade en metod för att extrahera trädet från en rasterbild som innehåller ett binärt fylogenetiskt träd. Resultaten visar ett utmärkt resultat under idealiska omständigheter, där bildkvaliteten är garanterad. Testerna på grafer från vetenskapliga artiklar visar på svårigheter för applicering på cirkulära träd. Testerna på rektangulära fylogenetiska träden visade istället mycket bättre resultat, där en stor majoritet fungerade felfritt efter några små modifieringar. Metoden testades också på icke-binära träd, i vilket fall träd fortfarande kunde extraheras, men naturligtvis med vissa fel. Vi fann att algoritmen är funktionell men bärkraften beror på kvaliteten på bilderna som används och mängden manuellt arbete som tillåts i användarens specifika projekt.

Acknowledgements

I would like to thank my supervisor Woosok Moon for supporting my chosen project idea, and for advising on how to work with and plan this kind of project. I also wish to thank my family and friends for all support throughout the work.

I would like to thank my supervisor Lars Arvestad for supporting my project and advice on improvements and proofreading. I would also like to thank my family for helping with support and additional proofreading.

Lastly I would like to thank the companies Elsevier, Springer and John Wiley Sons for accepting my requests for including their copyrighted figures in this thesis aswell as Rajangam et al and Nalbant et al for having some of their work published in Open Access.

Contents

1 Introduction

When finding a phylogenetic tree in a scientific paper, you may want to focus on different parts of the hierarchy or analyze the tree with your own code. For example the paper may handle mutations in genes, and you want to change some of the parameters for your own experimental purpose. There is also a possibility and interest to build databases of Phylogenetic trees with a uniform presentation.

One relatively labor intense way is to extract the data from the phylogenetic tree, and type the hierarchical structure and vertex names by hand in a systematic way. This thesis tries to automate extraction of data by giving the image of the tree to a program.

1.1 Problem Statement

Investigate the feasibility of extracting the hierarchical structure of a phylogenetic tree through pixel traversal. Create a program that, given an image of a phylogenetic tree, can extract the root, intersections and leaves of the tree and output this data in a mathematical form.

1.2 Limitations

The phylogenetic tree must be a binary tree. Emergence of errors using non-binary trees will not happen, but the output will be incorrect. The phylogenetic tree should be rooted to not be ambiguous, as the output will be rooted.

To select the root, distance from a specific edge of the image is used. The pixel of the graph closest to this edge is used as the root and modifications of the image may be necessary because of this. A more advanced selection process is left for another paper to limit the scope.

This algorithm will not extract the names of leaves and intersections. It will instead extract the pixel coordinates of these as the decision was made to only extract the structure of the tree. This can be further developed by looking at text closest to each leaf and intersection and couple it to the coordinates.

2 Background

This section covers background needed to understand parts of the methodology and to better understand the results. It contains related work, a short summery of relevant parts of the Newick format (output of the code) and what NumPys masked arrays are. There are definition of a neighbor, 8-simple, problem with adjacent neighbors which are pixel level concepts used in the method. It also covers thresholds which are relevant when making the image monochrome binary, synonyms of phylogenetic trees and why one of the limitations demands binary trees.

2.1 Related Work

There exists a program called TreeSnatcher (Laubach et al. 2012) which uses a graphical interface to interact with phylogenetic trees. This requires the user to interact by clicking branches and is semi-automatic but requires tracing tree contours and have problems with quality of the image (Lee et al. 2017). In 2017, Lee et al. proposed PhyloParser, a program described as "a fully automated end-to-end system for extracting species relationships from phylogenetic tree figures in scientific literature". Their method involves deep learning and convolutional networks to find the structure of the tree and presents a high success rate(Lee et al. 2017).

In a paper from 2012, Gallego et al. tried to encouraged modeling of images using graphs by proposing three different techniques (Gallego et al. 2012). Relevant for this paper is their third technique called Skeleton Graphs where the objects of in the image is thinned while retaining their general shape. This results in 1-pixel thick lines and represents the skeleton. A root is chosen, and the image is traversed using some criterion's and enables extraction of leaves, intersections and the root.

Gallego et al suggested using the thinning algorithm proposed by Cychosz et. al. (Cychosz 1994). The idea of the method is to make the image binary (black and white, no grayscale), then through several passes, removing border pixels (bordering a specific, orthogonal direction) that satisfy 8-simple but not endpoint (The 8-simple concept is defined in the background section of this thesis).

2.2 Phylogenetic tree and synonyms

A phylogenetic tree is a graph tree (see figure 2.1). It models evolutionary relationships between different species, genes or other types of populations. They are often rooted and binary (bifurcating) but there also exist those that may be unrooted or non-binary (Nakhleh 2013).

In the literature, there exist different terms for graph trees representing evolutionary relationships between genes or species. These names are used interchangeably and among them are dendrogram, cladogram and phylogenetic trees (Choudhuri 2014).

When categorizing the phylogenetic tree, two categories would be circular and rectangular phylogenetic tree. Other names for the circular phylogenetic trees are polar, fan or circular dendrograms. The rectangular phylogenetic tree is sometimes just called phylogenetic trees or dendrograms.

In this paper the term phylogenetic trees will be used, including both above categories, and with focus on rooted binary trees. We may also use the terms trees and graphs.

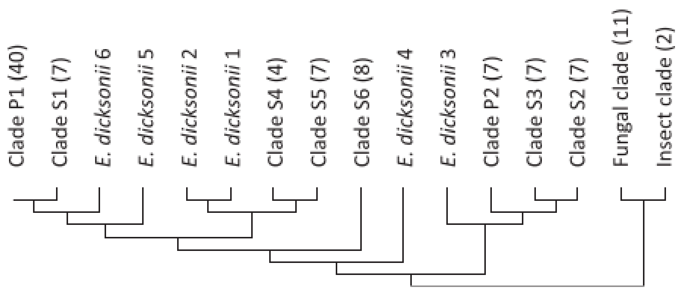


Figure 2.1: *Example of a phylogenetic tree, this specific tree is Graph 10 (Klinter et al. 2019).* ***This*** *figure was published in Molecular Phylogenetics and Evolution, 139, Klinter et al., Diversity and evolution of chitin synthases in oomycetes (Straminipila: Oomycota), 106558, Copyright @Elsevier 2019.*

2.3 Newick format

The output is a tree in Newick format, see figure 2.2. It is a string representation used by tree plotting programs to represent a tree as text. Intersections in this format are called interior nodes. Children of interior nodes are represented as comma-separated lists. The interior nodes themselves are represented as the parenthesis around the list of the children. The names of interior nodes are written to the right of the parenthesis. The names of leaves are written at their locations in the list. Both these types of names are optional. The string must end in a semi-colon ";" (Felsenstein 1990; Olsen 1990; Laubach et al. 2012).

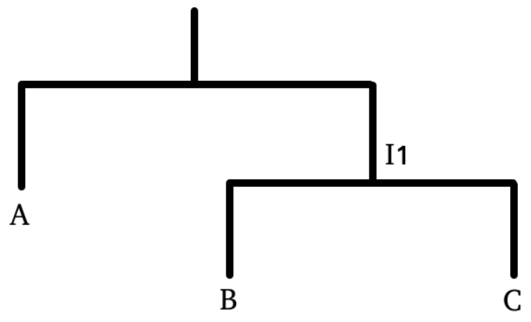


Figure 2.2: *This binary tree graph can be written in Newick format as "(A,(B,C)I1);" or "(,(,));" where root is the north most pixel.*

As this format is well established, easy to use and used by other applications , it was found to be suitable as output format. In this paper the term intersection and interior nodes will be used interchangeably.

2.4 Binary Image and the threshold problem

A binary image is an image of two colors, usually black and white. A color image can be transformed into grayscale which means each pixels value is set to an 8-bit number, this means it has a value of between 0 and 255. The transformation from color to grayscale used ITU-R 601-2 luma transform which follows the formula:

$$L = R \cdot \frac{299}{1000} + G \cdot \frac{587}{1000} + B \cdot \frac{114}{1000}. \tag{2.1}$$

R, G and B are 8-bit numbers representing the amount of red, green and blue contained in the original, colored pixel. The fractions, 299/1000 represent percentages of how much influence red will have on the final grayscale pixel (Clark 2009).

To make the image binary monochrome, a threshold is used, which is an 8-bit number. In a grayscale image, this threshold marks the cutoff point between pixels set to black or white by the code, usually colors below threshold is set to 0 and above to 1 but in the code the reverse happens, as a necessity of using NumPy (Cychosz 1994). See examples of effects of thresholds in figures (2.3, 2.4, 2.5)

2.5 NumPys masked array

NumPy is an open-source project that provides libraries for numerical computations in Python. For this project, the most important functionalities were their arrays and masked arrays. When iterating over large arrays of data such as pixels of images, NumPy array are quicker than Python list.

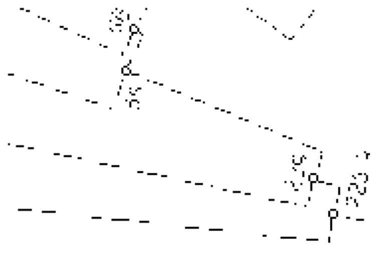


Figure 2.3: *Example of using too small threshold, holes appearing in graph 4 after thinning, (Frygelius et al. 2010).* ***Republished*** *with permission of Copyright @John Wiley Sons – Books 2010, from Evolution and human tissue expression of the Cres/Testatin subgroup genes, a reproductive tissue specific subgroup of the type 2 cystatins, Frygelius et al., 12, 3, 2010; permission conveyed through Copyright Clearance Center, Inc.*

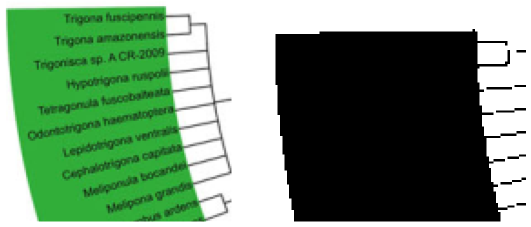


Figure 2.4: *Example using too high threshold resulting in areas not part of the graph being colored the same way as the graph in graph 11, left is the original image and right is the image made binary (Konrad et al. 2014).****Reprinted*** *by permission from Springer: Nature, Journal of molecular evolution, (The Phylogenetic Distribution and Evolution of Enzymes Within the Thymidine Kinase 2-like Gene Family in Metazoa, Konrad et al.), Copyright @Springer Nature 2014.*

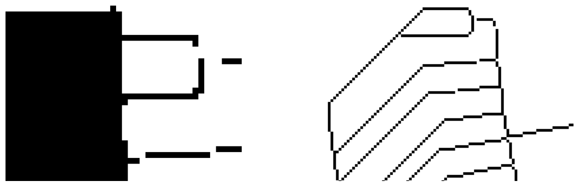


Figure 2.5: *Example where no threshold works, left shows holes and blocks in the same image, right shows the same image after thinning. (Konrad et al. 2014).****Reprinted*** *by permission from Springer: Nature, Journal of molecular evolution, (The Phylogenetic Distribution and Evolution of Enzymes Within the Thymidine Kinase 2-like Gene Family in Metazoa, Konrad et al.), Copyright @Springer Nature 2014.*

Masked arrays allows us to use this speed while ignoring elements (pixels) of the array which are finished being modified by masking them (The NumPy Community 2021).

2.6 Pixel concepts in the algorithm

2.6.1 Neighbors

A pixel has 8 adjacent pixels, unless it's located on the edge of the image. Any pixel value is either on or off, these are interpreted as same or similar color as the graph and the opposite respectively. Using the term black and white may cause confusion unless clarified at each use as the image colors are inverted during part of the algorithm.

Each pixel adjacent to another pixel of the same color is called a neighbor and already visited pixels are called marked or masked pixels (Cychosz 1994). This is performed using NumPys Masked array .

2.6.2 8-simple

The 8-simple is a procedure used in the thinning part of the algorithm. If a pixel can be removed without destroying the connectedness between neighbors, then it is 8-simple, see figure 2.6. (Cychosz 1994)

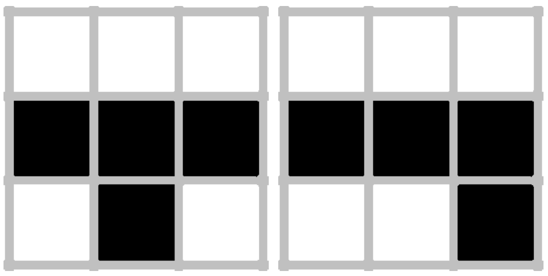


Figure 2.6: *Left image where the center pixel is 8-simple, right image is the negative example.*

Any pixel with 8 adjacent pixels can be represented as an 8-bit number using a configuration like the figure 2.7. Note that the quantity of 8-bit numbers is 256. Writing a function and checking by hand, the quantity of 8-simple 8-bit numbers is 133.

These are all the 8-bit numbers that satisfy the condition of being an 8-simple using the configuration of figure 2.7. Some of the examples from the list below can be seen in figure 2.8: 0, 1, 2, 3, 4, 5, 6, 7, 8, 12, 13,

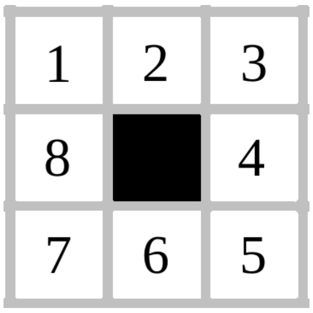


Figure 2.7: *The ordering of adjacent pixels when interpreting the neighborhood as an 8-bit number. The 1 represent 128, the 2 represent 64 and the 8 represent 1.*

14, 15, 16, 20, 21, 22, 23, 24, 28, 29, 30, 31, 32, 48, 52, 53, 54, 55, 56, 60, 61, 62, 63, 64, 65, 67, 69, 71, 77, 79, 80, 81, 83, 84, 85, 86, 87, 88, 89, 91, 92, 93, 94, 95, 96, 97, 99, 101, 103, 109, 111, 112, 113, 115, 116, 117, 118, 119, 120, 121, 123, 124, 125, 126, 127, 128, 129, 131, 133, 135, 141, 143, 149, 151, 157, 159, 181, 183, 189, 191, 192, 193, 195, 197, 199, 205, 207, 208, 209, 211, 212, 213, 214, 215, 216, 217, 219, 220, 221, 222, 223, 224, 225, 227, 229, 231, 237, 239, 240, 241, 243, 244, 245, 246, 247, 248, 249, 251, 252, 253, 254, 255.

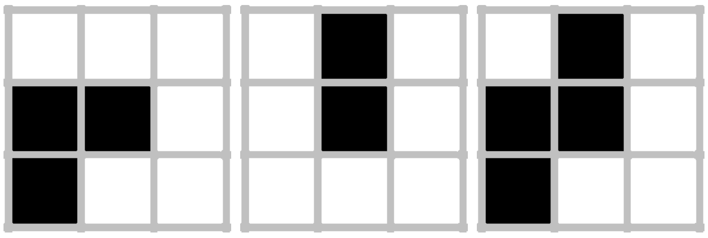


Figure 2.8: *Examples of configurations of neighborhoods, left image represents 3, middle 64, right 67.*

2.6.3 The adjacency problem, intersection and adjacent neighbors

The algorithm finds leaves and interior nodes (intersections) by traversing neighboring pixels of the phylogenetic tree (described in more detail in the implementation section). During this traversal, a pixel is an interior node if it contains two or more unmarked neighbors. The I in the images is the interior node and the red crosses are the unmarked neighbors (figure 2.9). If any of these red crosses borders another red cross (unmarked neighbors), then these will falsely be interpreted as interior nodes (false intersections), which is a problem.

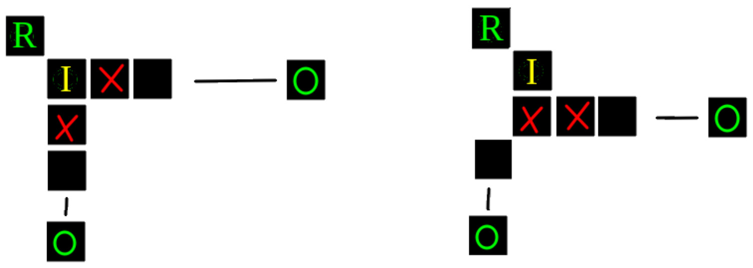


Figure 2.9: *Two examples of the same scenario, any black box represent a pixel. The roots are the top left R:s where the traversal originated from, the red crosses are the bordering neighbors and the yellow I:s are the interior nodes. Green circles are endpoints and the thin lines represent arbitrary lines of pixels without interior nodes.*

There are two ways to solve this problem:

- The first, which was not used, is to mark all neighbors before you start traversing any of them. This was not used as it was thought of after the code were finished, this could be a future development work.
- The second way is inspired by Gallego (2012), this involves joining problematic neighbors with the corresponding interior node.

The red crosses that were falsely interpreted as internal nodes (false intersections) can be found by comparing the distance between any two unmarked neighbors of internal node I. If the distance is less than or equal to 2, they are adjacent to each other. This is only true for the eight neighboring pixels.

You solve this by joining the internal node and the false intersections into a set called pixel cluster. The unmarked neighbors to any of the pixels in the clustered is compared in search of false intersection and the operation is repeated until no false intersections remain. The code is found in graph_finder.py in joinAdjacentNeighbors(). (Brandberg 2021)

You can see that the solution is more general and works for 4 and 5 false intersections too in figure 2.10 and 2.11.

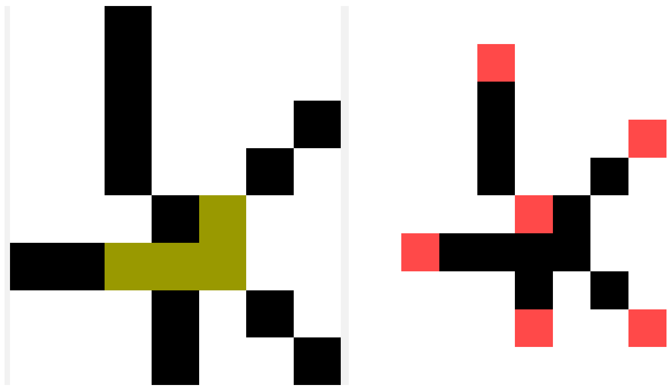


Figure 2.10: *Examples of an interior node with 4 false intersections. In left image, brown squares are false intersections. In right image red squares are the internal nodes, leaves and root after the algorithm is run. Notice that none of the false intersections are red and no red pixel borders another red pixel.*

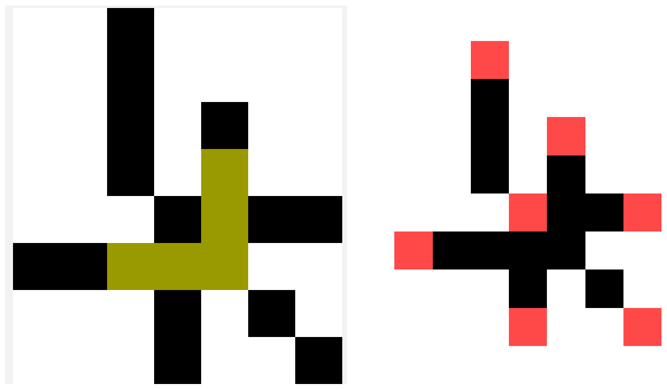


Figure 2.11: *Same as for (figure 2.10) but with 5 false vertices.*

2.7 Visualizing phylogenetic trees

When saving the generated Newick format as graphs in images or displaying them, ETE3 was used. ETE Toolkit is a Python framework that is used for displaying and analyzing phylogenetic trees (Huerta-Cepas et al. 2016).

2.8 Why binary trees were necessary

An interior node (intersection) is defined, during traversal, as a pixel with two or more unmarked neighbors. When traversing pixels in a graph, it is hard to know if an intersection should be interpreted as a child of the previous interior node or part of the previous interior node.

For the case of rectangular phylogenetic trees the interpretation was possible, but in practice problems occurred. The theoretical solution for calculating non-binary trees correctly would be to join adjacent intersections sharing same pixel height, from the perspective of the root direction (meaning the same y-coordinate).

In figure 2.12, this theoretical solution would entail merging the blue squares into one intersection, as shown in the right image, and make all children of these three interior nodes the children of this new interior node. In the image the root is the north most pixel, but if it were the east or west most pixel, the pixel height should be interpreted in terms of x-coordinate instead of y-coordinate.

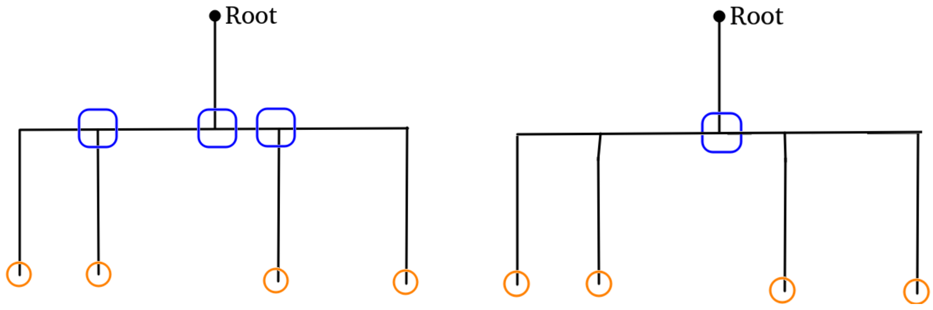


Figure 2.12: *Example where root is the north most pixel, blue squares represent three interior nodes sharing the same pixel height and the orange circles represent children to the interior nodes.*

As seen in figure 2.13, this solution necessitates a tolerance in pixel height resulting from the process of thinning. In the image, the root is the northern red pixel. The left and right intersections represents intersections which in the original image occupy the same pixel height as the middle intersection. This means that this is a non-binary part of the graph.

According to the theoretical solution, these interior nodes should be joined together but in the image, they have a height difference of 2 pixels. So a minimum tolerance difference of 2 pixels is required.

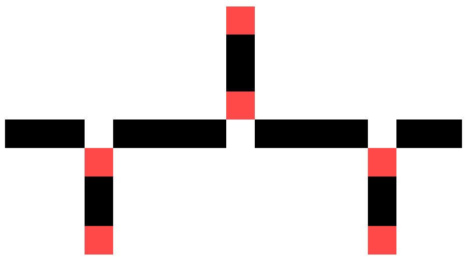


Figure 2.13: *This image is already thinned, the root is the north most red pixel, the other red pixels represent intersections and leaves found by the algorithm. The left, right and middle intersections abstractly occupy the same pixel height. The most left and right pixels are black as the image is cropped.*

In figure 2.14, we see that a tolerance of 2 pixels would make this impossible to use. At the arrow the difference in pixel height of the two interior nodes is 2 which means they merge according to the tolerance of 2. There is also figure 2.15 which shows that the concept of pixelsheight would need a new definition due to the circular graphs.

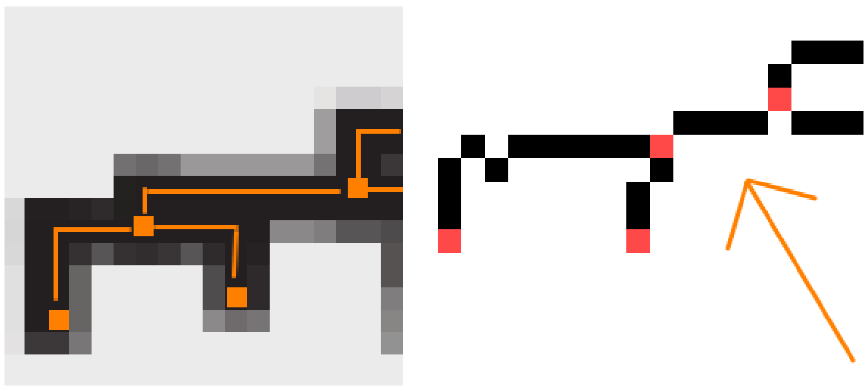


Figure 2.14: *Examples where a tolerance of 2 is problematic, left image is the original version with orange guidelines showing the true interpretation and right is the thinned version. The red pixels represent intersections and leaves found by the algorithm using graph 5 (Mazet et al. 2003).* ***This*** *figure was published in Gene, 316, Mazet et al., Phylogenetic relationships of the Fox (Forkhead) gene family in the Bilateria, 79–89, Copyright @Elsevier 2003.*

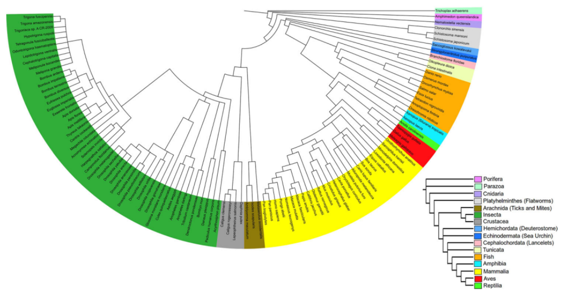

Porifera
Parazoa
Cnidaria
Platyhelminthes (Flatworms)
Arachnida (Ticks and Mites)
Insecta
Crustacea
Hemichordata (Deuterostome)
Echinodermata (Sea Urchin)
Cephalochordata (Lancelets)
Tunicata
Fish
Amphibia
Mammalia
Aves
Reptilia

Figure 2.15: *Example of graph 11 found in results, a circular phylogenetic tree were pixel height would need to be redefined using for example polar coordinates (Konrad et al. 2014).* ***Reprinted*** *by permission from Springer: Nature, Journal of molecular evolution, (The Phylogenetic Distribution and Evolution of Enzymes Within the Thymidine Kinase 2-like Gene Family in Metazoa, Konrad et al.), Copyright @Springer Nature 2014.*

For these reasons the choice was made not to implement this solution to allow non-binary trees.

3 Methodology

This section covers the hardware used to test graphs, source of the graphs used when testing algorithm and the general code implementation of the algorithm and generation of graphs. It also covers methodology of testing and common errors handled in result.

3.1 Hardware

Tests were run using Ubuntu 20.04.2 LTS, on a 64-bit machine with an Intel® Celeron(R) CPU N3060 at 1.60GHz, 3.8 GiB RAM, Intel® HD Graphics 400 (BSW) GPU. This is an inexpensive, slow and three years old laptop computer.

3.2 Images for testing

The images came from two different sources, the first source is 16 graphs, sent by the supervisor of this paper through 10 different scientific papers, the second source were two functions that generates binary trees found in the file graph_generation.py in (Brandberg 2021).

The function generateAllTrees(leaves, keepSmallerTrees) generates all binary trees up to a specified number of leaves, the boolean parameter "keepSmallerTrees" specify keeping trees with smaller quantity of leaves.

The second function generateXRandomTrees(leaves,trees,seed) works similar to generateAllTrees(), but only keeps the trees with the most leaves and limits the quantity of trees between each iteration.

The graphs analyzed were grouped into two subsets: circular (figure 3.1) and rectangular phylogenetic trees (figure 3.1).

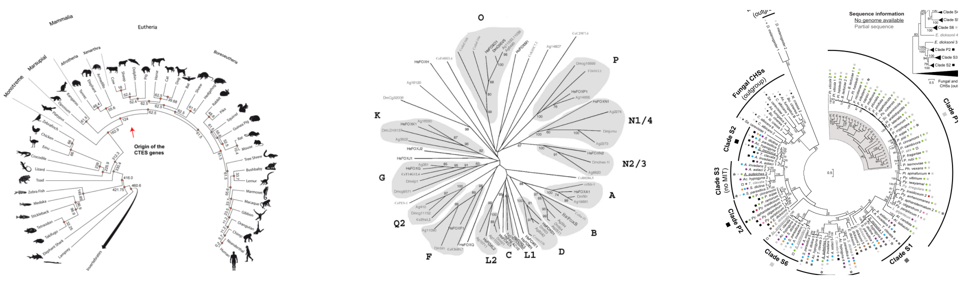


Figure 3.1: *Examples of circular phylogenetic trees where the root is located in the center of a disc and branches extends at different angles. Left to right is graph 4, 6 and 8 (Frygelius et al. 2010; Mazet et al. 2003; Klinter et al. 2019). **Republished** with permission of Copyright ©John Wiley Sons – Books 2010, from Evolution and human tissue expression of the Cres/Testatin subgroup genes, a reproductive tissue specific subgroup of the type 2 cystatins, Frygelius et al., 12, 3, 2010; permission conveyed through Copyright Clearance Center, Inc. **This** figure was published in Gene, 316, Mazet et al., Phylogenetic relationships of the Fox (Forkhead) gene family in the Bilateria, 79–89, Copyright ©Elsevier 2003. **This** figure was published in Molecular Phylogenetics and Evolution, 139, Klinter et al., Diversity and evolution of chitin synthases in oomycetes (Straminipila: Oomycota), 106558, Copyright ©Elsevier 2019.*

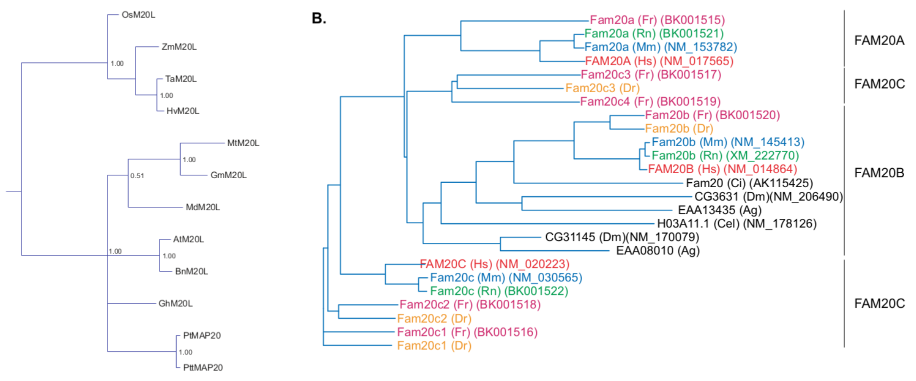


Figure 3.2: *Examples of the rectangular phylogenetic trees where root is the furthest pixel to one side and the leaves all point in the opposite direction. Left graphs 15, and right is 12 found (Rajangam et al. 2008b; Nalbant et al. 2005).*

3.3 Implementation

3.3.1 General overview of the implementation

The code first calls the thinning algorithm. For practicality reasons, this first adds a thin border of one pixel to the image and inverts it. The traversal algorithm is then called, the root is chosen, and the graph given to the algorithm, is traversed from the root. When all intersections and leaves have been located, a modified image and the Newick code is returned.

For more information see the run() function in main.py (Brandberg 2021).

3.3.2 Thinning implementation

The algorithm first adds a one-pixel border around the image to ensure no pixel outside the image is accessed. The image is made binary, inverted and NumPys masked arrays is used to quickly iterate across the pixels of the image and masking pixel that does not need more modifications.

The thinning is performed by iterating over the pixels of the image during several passes. When no pixels are removed during a pass, the algorithm is complete. Each pass is divided into four sub passes, each representing an orthogonal direction (N, S, E, W). Pixels are marked for removal if it's located on an edge in the direction of the sub pass and is an 8-simple while not being an end point. After any sub pass, all marked pixels are removed and masked (Cychosz 1994).

For more information see thining_algorithm.py (Brandberg 2021).

3.3.3 Graph finding implementation

The root is chosen by giving a direction (N, S, E, W) to the algorithm. The pixel of the same color as the graph and closest to that specified directional edge of the image, is the choice for root. Intersections and leaves of the graph are found by traversing neighboring pixels of the same color.

Each pixel adjacent to another pixel of the same color is a neighbor. When traversing the image, already visited pixels are marked using NumPys Masked array. Unless the root has no neighbors, a pixel with no unmarked neighbors is an endpoint, equivalent of a leaf. A pixel with one unmarked neighbor is an arcpixel, the same as a point on a branch. If there are two or more unmarked neighbors, the pixel is an interior node(intersection).

When the algorithm encounters an arcpixel the current pixel is discarded and the non-visited neighbor is chosen as the current pixel instead. Encountering a leaf just returns the Newick format of a leaf and a list of vertices since the last intersection.

Finding intersections uses the adjacency problem approach discussed in background. The unmarked neighbors found are called recursively and Newick code is generated from each, the list of vertices and Newick format is then returned.

For more information see graph_finder.py (Brandberg 2021).

3.3.4 Tree generation implementation

The function generateAllTrees(L, K) starts with a tree of two leaves and adds the tree to a set of new trees.

At each iteration step over the trees in the set, the algorithm substitutes one leaf at the time with a basic tree of two leaves to create a new tree. This tree is added to a temporary set containing all new trees in this iteration. The procedure is repeated L-2 times. The parameter K decides if the original trees with smaller number of leaves will be kept.

The three images of figure 3.3 shows the set after the first iteration when keepSmallerTrees is set to true. In the next pass, the algorithm will create 8 new trees. This is the same as the number of leaves in the current trees, but 3 of these will be duplicates and therefore ignored.

The function generateXRandomTrees(L,T,R) works like generateAllTrees(L,K)

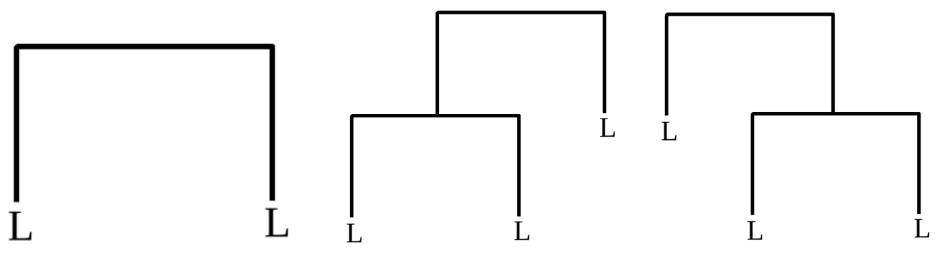


Figure 3.3: *The three trees stored in the set after one iteration.*

but where K is false and T decides how many trees to keep during each iteration. The trees to keep are chosen randomly.

A problem with generateAllTrees() is that Python list have a limitation which prevents the use of more than 15 leaves, this is due to a memory limitations using Python sets.

Because of this limitation, the only option were to use generateXRandomTrees(), when encountering trees with larger number of leaves.

When generating the graphs, the package

For more information see graph_generation.py (Brandberg 2021).

3.4 Testing of the graphs

3.4.1 Testing the 16 graphs

To evaluate the results there are a few aspects we focus on.

These include, whether the Newick Format were correctly extracted. Successful test without modifying the image. If modifications were necessary, was the problem only related to a few small problems or were there major errors in the output.

To be correct the entire phylogenetic tree, from root to intersections and finally to leaves must be found. A tree which requires modifications may be deemed as successful if there are maximum 3 small errors which can be fixed quickly. After these modifications, the code is required to correctly identify all interior nodes and leaves to not be deemed a failure. An exception to this rule were the non-binary graphs, which did not specifically require the hierarchical structure of non-binary interior nodes to be correct. But all other aspects still needed to be correct. Motivations for non-binary aspect is found in the discussion section.

Note that finding the correct threshold is not counted as a modification.

In short, 100 percent of the leaves and interior nodes must be found and display correct parent-child relationship, with only a few minor modifications, to be successful. The sole exception were the non-binary trees where the rules of the relationships could be bent a little bit for the non-binary parts.

The Problems that arose during testing include:

- bad positioning of root,
- non-binary sub-trees of the tree,
- non-graph parts overlapping with the tree, for example text and lines,
- holes or blocks resulting from bad thresholds,
- no possibility of choosing a correct threshold,
- branches too close to each other.

Some errors could easily be corrected.

- Bad positioning of root was fixed by drawing lines or removing nonessential pixels of the image.
- Text or numbers overlapping with the graph, were modified by changing the threshold to more appropriate value.
- When branches came too close to each other, setting a precise thresholds, within a small margin for error was often enough.
- The non-binary errors were noted in the result as discrepancy.
- No possibility of choosing a threshold because selected value was to high and low at the same time, for different parts of the graph, meant

that no useful output could be extracted and noted as failure in the result.

3.4.2 Testing of generated binary phylogenetic trees

The tests were performed by generating all possible binary trees of two to ten leaves using the function generateAllTrees(), for trees of more leaves we used generateXRandomTrees(). These generated images of trees were given to the algorithm and the algorithmically extracted code was compared to the original Newick code, finally results were returned.

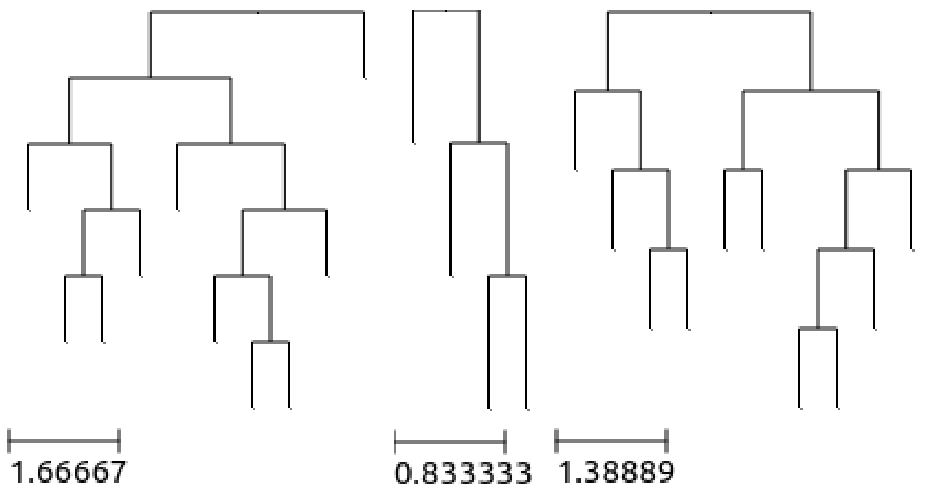


Figure 3.4: *Examples of three images tested, left to right, their quantity of leaves are 10, 4, 10.*

All trees up to 10 leaves were tested, figure 3.4 shows some of them. When testing trees of more leaves, the set of trees were limited to 200 trees. The tests, using the limited number of leaves, were performed on up to 20 leaves, 50, 60, 70, .. 110 leaves. These numbers were used due to the time it took to thin and traverse the trees, just generating the set of trees took a fraction of the time and was not a factor in the decision to limit the number of leaves or trees. The figure 3.5 shows one of the trees of 60 leaves.

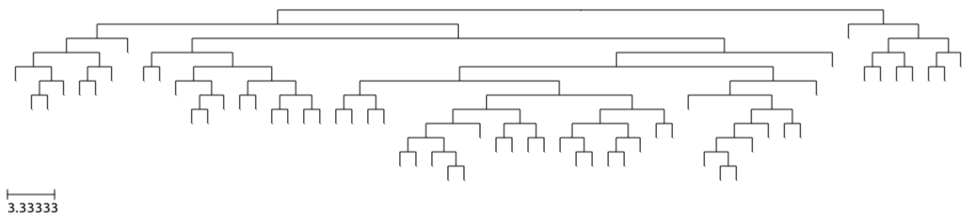


Figure 3.5: *Examples of one of the larger trees tested of 60 leaves.*

4 Result

This section covers the overview of results from testing the 16 graphs received and testing the generated graphs. The specifics of testing the 16 graphs can be found in the appendix.

4.1 Overview of results

The success of a test is determined by a string in Newick format being generated as an image of a phylogenetic tree, then run through the algorithm and returned as new Newick format string. If a comparison between these two strings returns and are equal, the test is considered successful. The success rate is the percentage of successful tests.

When generating trees, no matter the number of leaves tested, all trials returned a success rate of 100 percent but higher quantities of trees or leaves, took longer to run. The time limited the tests to either:

- all binary trees of up to 10 leaves,
- or selection of 200 trees, when using more leaves.

Analyzing provided phylogenetic trees, the circulars had a success rate of 25 percent due to problem of choosing valid threshold. All these graphs required modifications to find the root (Table 4.1).

The rectangular phylogenetic trees had a success rate of 83 percent, with two failures due to lines intersecting with the graph and in the case of graph 9 triangles which abstractly represented sub-trees. Two graphs needed modifications of the root as non-graph pixels were chosen. (Table 4.2). The most common problem was due to non-binary parts of the

Graph ID	Success	Failure	Root	Threshold
4	1		1	175
6		1	1	
8		1	1	
11		1	1	
Total	1	3	4	

Table 4.1: ***Graph ID*** *is the id of the graph,* ***Success*** *and* ***failure*** *shows the result,* ***Root*** *shows if root modification were necessary and* ***Threshold*** *shows a numbered that worked if successful.*

Graph ID	Success	Failure	Root	Non-binary	Overlap	Triangles	Threshold
1	1			1			130
2		1		1	1		
3	1		1				130
5	1			1	1		35
7	1		1				130
9		1			1	1	
10	1						150
12	1			1	1		130
13	1						130
14	1			1			14
15	1			1			130
16	1			1			100
Total	10	2	2	7	4	1	

Table 4.2: *For* ***Graph ID***, ***Success***, ***failure***, ***Root*** *and* ***Threshold*** *(see table 4.1).* ***Non-binary*** *shows these graphs.* ***Overlap*** *shows graphs where modifications were necessary to overlapping text or lines.* ***Triangles*** *shows graphs with large triangles at the end of leaves, representing sub trees.*

graphs. This involved 58 percent of all rectangular trees and 60 percent of the successful ones. This problem was handled as a minor error which did not affect the success rate of the algorithm. This was due to non-binary parts only affecting parent-child relationship in these few places and as the alternative would be to not include these graphs in the result. As for overlap, two of the successful trees had problems involving text overlapping the graph. In the case of graph 5, a leaf disappearing due to being too short is also included in this. The most common threshold was 130 and 150 were the highest necessary value of the successful trees.

Leaves	Trees	Time (seconds)	Time (Minutes)	Time/Tree	Time (Set)	Time (Alg)	Time/Tree (Alg)	Success
2	1	0.72	0.01	0.72	0	0.72	0.72	100%
3	2	0.31	0.01	0.16	0	0.31	0.16	100%
4	5	0.76	0.01	0.15	0	0.76	0.15	100%
5	14	2.25	0.04	0.16	0	2.25	0.16	100%
6	42	7.46	0.12	0.18	0	7.46	0.18	100%
7	132	25.07	0.42	0.19	0	25.07	0.19	100%
8	429	94.40	1.57	0.22	0	94.39	0.22	100%
9	1430	300.14	5.00	0.21	0.02	300.12	0.21	100%
10	4862	1099.60	18.33	0.23	0.05	1099.55	0.23	100%

Table 4.3: ***Leaves*** *is the number of leaves in the trees.* ***Trees*** *is the number of trees in the set.* ***Time*** *indicate seconds unless otherwise specified.* ***Time/Tree*** *is time per tree.* ***Time(Set)*** *indicate time to create the set of trees.* ***Time(Alg)*** *is the time for the algorithm to run on the set of trees. Success is the chance of success given the set of trees.*

4.2 The Generated Trees

Using a function to collect all possible binary trees of up to 10 leaves yielded the results seen in table 4.3. This tested in total 6917 trees of up to 10 leaves, all of the trials were successful.

Some conclusions could be drawn from table 4.3:

- The time to create the set of trees is negligible compared to the time running the algorithm.
- There is an increase in run time when the number of trees and leaves increase and using time per tree we see that each leaf adds to the time.

More information found by reviewing table 4.4:

- When the column leaves increase, we see a trend of convergence in the column Multiplicative Increase (Trees). This column displays the multiplier of quantity increase of trees compared to the row above. Notice that even though the column Multiplicative Increase (Trees) looks convergent as the column Increase of Multiplier decreases, this can still be divergent.

Leaves	Trees	Multiplicative Increase (Trees)	Increase of Multiplier	Time	Multiplicative Increase (time)
2	1			0.72	
3	2	2.0		0.31	0.43
4	5	2.5	0.50	0.76	2.5
5	14	2.8	0.30	2.25	2.9
6	42	3.0	0.20	7.46	3.3
7	132	3.1	0.14	25.07	3.4
8	429	3.3	0.11	94.40	3.8
9	1,430	3.3	0.08	300.14	3.2
10	4,862	3.4	0.07	1099.60	3.7
11	16,796	3.5	0.05		
12	58,786	3.5	0.05		
13	208,012	3.5	0.04		
14	742,900	3.6	0.03		
15	2,674,440	3.6	0.03		

Table 4.4: ***Leaves*** *is the number of leaves in the trees used,* ***Time*** *indicate seconds,* ***Multiplicative Increase (Trees)*** *is the multiplicative increase in quantity of trees,* ***Increase of Multiplier*** *is how much this multiplier grows when the number of leaves grow.*

- The time increases of working on sets with more leaves and larger sets of trees increases by a multiplier of approximately the same same size as the trees for the 10 first leaves but the multiplier is around 0.3 higher.

As an example, if we assume a multiplier of 3.7 for time increase, running the algorithm for all binary trees of 11 leaves would take around 70 minutes.

In table 4.5, we limiting ourselves to a smaller subset of possible binary trees, with maximum of 200 trees. This lets us increase the number of leaves while keeping the time for running the program down. The generation yielded the result seen in table 5 which tested 4196 trees of up to 110 leaves, all of these trials were also successful. Notice the jump in the number of leaves, after 20 in the result. We see that the time to create the set of trees is still negligible compared to running the algorithm.

The table 4.6 below compares the time it takes to run the two functions generateXRandomTrees() and generateAllTrees(). We see that the error

Leaves	Trees	Time (Total)	Time (Minutes)	Time/Tree	Time (Set)	Time (Alg)	Time/Tree (Alg)	Success
2	1	0.69	0.01	0.69	0	0.69	0.69	100%
3	2	0.26	0.00	0.13	0	0.26	0.13	100%
4	5	0.74	0.01	0.15	0	0.74	0.15	100%
5	14	2.23	0.04	0.16	0	2.23	0.16	100%
6	42	6.80	0.11	0.16	0	6.80	0.16	100%
7	132	23.42	0.39	0.18	0	23.42	0.18	100%
8	200	38.83	0.65	0.19	0.01	38.83	0.19	100%
9	200	42.92	0.72	0.22	0.02	42.91	0.22	100%
10	200	46.75	0.78	0.23	0.04	46.71	0.23	100%
11	200	48.46	0.81	0.24	0.04	48.42	0.24	100%
12	200	53.53	0.89	0.27	0.06	53.47	0.27	100%
13	200	62.02	1.03	0.31	0.08	61.94	0.31	100%
14	200	66.71	1.11	0.33	0.11	66.60	0.33	100%
15	200	68.00	1.13	0.34	0.12	67.88	0.34	100%
16	200	69.65	1.16	0.35	0.14	69.50	0.35	100%
17	200	75.86	1.26	0.38	0.17	75.68	0.38	100%
18	200	81.51	1.36	0.41	0.21	81.30	0.41	100%
19	200	85.09	1.42	0.43	0.24	84.85	0.42	100%
20	200	91.56	1.53	0.46	0.25	91.31	0.46	100%
50	200	280.47	4.67	1.40	1.92	278.54	1.39	100%
60	200	315.16	5.25	1.58	3.16	312.00	1.56	100%
70	200	351.95	5.87	1.76	4.25	347.70	1.74	100%
80	200	374.78	6.25	1.87	5.74	369.04	1.85	100%
90	200	443.52	7.39	2.22	6.85	436.67	2.18	100%
100	200	493.57	8.23	2.47	9.07	484.50	2.42	100%
110	200	575.45	9.59	2.88	10.50	564.95	2.83	100%

Table 4.5: ***Leaves*** *is the number of leaves in the trees.* ***Trees*** *is the number of trees in the set,* ***Time*** *indicate seconds unless otherwise specified,* ***Time/Tree*** *is time per tree,* ***Time(Set)*** *indicate time to create the set of trees,* ***Time(Alg)*** *is the time for the algorithm to run on the set of trees, Success is the chance of success given the set of trees.*
Notice there is a jump from 20 leaves to 50 leaves in the table.

Leaves	All	XRandom	Error Time/Tree
2	0.72	0.69	0.029
3	0.16	0.13	0.026
4	0.15	0.15	0.006
5	0.16	0.16	0.001
6	0.18	0.16	0.016
7	0.19	0.18	0.013
8	0.22	0.19	0.026
9	0.21	0.22	0.005
10	0.23	0.23	0.008

Table 4.6: ***Leaves*** *is the number of leaves in tested trees,* ***All*** *is the time per tree when running generateAllTrees(),* ***Xrandom*** *is the time per tree when running generateXRandomTrees(),* ***Error Time/Tree*** *is the absolute time difference between them.*

between them in terms of time per tree is very small when talking about 10 leaves.

5 Discussion

This thesis investigated the feasibility of using neighboring pixels to traverse and extract hierarchical data from phylogenetic trees using Python. The purpose was to create a program which, given an image, outputs the structure of the tree graph. The paper did not seek to solve the problem of combining this hierarchical data with the names of leaves and intersections. The pixel coordinate of leaves and intersections were stored in the Newick format to make them easy to find.

We found that the method works flawlessly under ideal circumstances, when generating binary trees of arbitrary leaf size. When checking the 16 images of graphs, of the circular phylogenetic trees, only 1 out of 4 worked, but all needed modifications. Of the 12 rectangular phylogenetic trees, 2 failed and 4 needed small modifications to work. This makes a success rate of 83 percent.

We also found that 58 percent of the rectangular trees were non-binary which should be noted but not result in failure, as they do not affect the main performance of the algorithm. A non-binary intersection only affects sibling relationships, if there exist an intersection with three children, two of these children will be replaced by an intersection whose children are the two replaced children. This reasoning can be expanded to intersections of more than three children. As these non-binary graphs is almost as interesting as binary graphs from the point of exploring the feasibility, and the non-binary aspect does not affect affinity with problems of root, overlapping non-graph parts, threshold and too close branches, they were used in the analyzes.

5.1 Sources of error

A source of error is contained in the control the non-generated graphs, as a person must check the Newick code or the returned, modified image. Any mistake may lead to missed errors. These results were checked multiple times and at different days which should counteract possible errors.

Another source of error is related to the limitation of testing generated trees. As trees of 11 leaves or more were not tested extensively, smaller subsets of the possible trees were used and no trees with leaves of more than 110 leaves were run through the algorithm due to increasing runtime. The number of trees tested of up to 10 leaves were 6917 and the number of trees of 11 or more leaves tested were 4196, in total 11,113 trees. Any error occurring in a generated tree would also occur on a sub-tree, but as 100 percent of these cases tested succeed, using ideal quality images of phylogenetic trees. We are confident this method works in the ideal case.

5.2 Problematization

One problem is the limited number of real world examples of phylogenetic trees used. If done again, a large number of scientific papers with phylogenetic trees would be collected and a random selection of these graphs would be run through the algorithm. The number of graphs would have to be limited to be manageable as measure of performance if the control is performed by eye, by one person. As a side note, collecting licenses for this large quantity of graphs would also be a limiting factor to this suggestion.

A problem with the method is the reliance on threshold. For some graphs the acceptable range of threshold values was big while for example graph 5 only values between 32 and 37 could be used. The only way to find the correct range was testing and checking by eye. Another problem with the method is locating the root. This was often easy to fix but demanded manual revisions of image in applicable cases.

As described, the biggest obstacles are finding the root, text or lines overlapping the graph, bad quality of image which complicates selection of threshold or makes threshold selection impossible, or lastly branches almost overlapping each other.

The cases where this algorithm would be most useful are in large phylogenetic trees with many leaves and intersections. Typing these by hand may not be feasible and here the algorithm would be useful. The problem with these trees is that information is packed tight, numbers may not fit without encroaching on the graph using lines or themselves. Branches may need to be too close to each other to fit on the page.

5.3 Conclusion

This method shows promise if you are prepared to verify the result after the run. Looking at the data, the algorithm works better with rectangular phylogenetic trees than circular ones but as the data set is small, confirming this with certainty is not possible. If you stay within the limitations of the algorithm, or you ignore the non-binary limitation but accept incorrect hierarchically classifications in these cases, this is a possible method to use.

As the extraction of texts were ignored in this paper, there are more work to be done on this application. As the runtime was not in focus, the code was optimized a bit but not enough, a possible way forward would be to port the code to a faster language than Python, replace the thinning algorithm with a faster one. Lastly, implement a process that combines text extraction with the coordinates of inserted in the Newick code for further development of the algorithm. Another feature to implement would be in the further development would be the lifting the limitation of non-binary parts, maybe by looking at horizontal and vertical tendencies of branches.

So is the method feasible? Yes, it works in the ideal circumstance and if you allow modification, to some part of the images, it then worked for a majority of them. Is the method practical? Maybe, depends on the manual labor allowed by the specific use case. Can the algorithm handle multiple trees at the same time, meaning it be automated? Yes if you know an acceptable threshold and all images are modified prior to run or do not need modifications. The quality of each image would need to be acceptable and at the same level for all the phylogenetic trees.

References

Brandberg, A. (2021). *Dendogram Traversal*. https://github.com/bjru/dendogram-traversal.

Choudhuri, S. (2014). "Chapter 2 - Fundamentals of Molecular Evolution". In: *Bioinformatics for Beginners*. Ed. by S. Choudhuri. Oxford: Academic Press, pp. 27–53. ISBN: 978-0-12-410471-6. DOI: https://doi.org/10.1016/B978-0-12-410471-6.00002-5. URL: https://www.sciencedirect.com/science/article/pii/B9780124104716000025.

Clark, A. (2009). *PIL:s Image.convert() function*. [Accessed 14-August-2021]. URL: https://pillow.readthedocs.io/en/stable/reference/Image.html?highlight=convert#PIL.Image.Image.convert.

Cychosz, J.M. (1994). "Efficient Binary Image Thinning Using Neighborhood Maps". In: *Graphics Gems IV*. USA: Academic Press Professional, Inc., 465–473. ISBN: 0123361559. URL: https://books.google.se/books?id=CCqzMm_-WucC&lpg=PA465&dq=Efficient%20Binary%20Image%20Thinning%20Using%20Neighborhood%20Maps&hl=sv&pg=PA466#v=onepage&q&f=false.

Djerbi, S. et al. (2005). "The genome sequence of black cottonwood (Populus trichocarpa) reveals 18 conserved cellulose synthase (CesA) genes". In: *Planta* 221. Copyright ©Springer Nature 2005, pp. 739–746. DOI: 10.1007/s00425-005-1498-4.

Felsenstein, J. (1990). *The Newick tree format*. [Accessed 14-August-2021]. URL: https://evolution.genetics.washington.edu/phylip/newicktree.html.

Frygelius, J. et al. (2010). "Evolution and human tissue expression of the Cres/Testatin subgroup genes, a reproductive tissue specific subgroup of the type 2 cystatins". In: *Evolution & Development* 12. Copyright ©John Wiley Sons – Books 2010. DOI: 10.1111/j.1525-142X.2010.00418.x.

Fugelstad, Johanna et al. (2009). "Identification of the cellulose synthase genes from the Oomycete Saprolegnia monoica and effect of cellulose synthesis inhibitors on gene expression and enzyme activity." In: *Fungal genetics and biology : FG & B* 46 10. Copyright ©Elsevier 2009, pp. 759–767. DOI: 10.1016/j.fgb.2009.07.001.

Gallego, A.J., J. Calera-Rubio, and D. Lopez (Jan. 2012). "Structural Graph Extraction from Images". In: vol. 151, pp. 717–724. ISBN: 9783642287640. DOI: 10.1007/978-3-642-28765-7_86. URL: https://www.researchgate.net/publication/237265944_Structural_Graph_Extraction_from_Images.

Huerta-Cepas, J., F. Serra, and P. Bork (2016). *ETE 3: Reconstruction, analysis and visualization of phylogenomic data.* Mol Biol Evol 2006, [Accessed 14-August-2021]. DOI: 10.1093/molbev/msw046. URL: http://etetoolkit.org/.

Klinter, Stefan, Vincent Bulone, and Lars Arvestad (2019). "Diversity and evolution of chitin synthases in oomycetes (Straminipila: Oomycota)". In: *Molecular Phylogenetics and Evolution* 139. Copyright ©Elsevier 2019, p. 106558. ISSN: 1055-7903. DOI: https://doi.org/10.1016/j.ympev.2019.106558. URL: https://www.sciencedirect.com/science/article/pii/S1055790318301830.

Konrad, Anke et al. (Feb. 2014). "The Phylogenetic Distribution and Evolution of Enzymes Within the Thymidine Kinase 2-like Gene Family in Metazoa". In: *Journal of molecular evolution* 78. Copyright ©Springer Nature 2014, pp. 202–213. DOI: 10.1007/s00239-014-9611-6.

Laubach, T., A. von Haeseler, and M. Lercher (May 2012). "TreeSnatcher plus: Capturing phylogenetic trees from images". In: *BMC bioinformatics* 13, p. 110. DOI: 10.1186/1471-2105-13-110. URL: https://www.researchgate.net/publication/225042859_TreeSnatcher_plus_Capturing_phylogenetic_trees_from_images.

Lee, P.S. et al. (2017). "PhyloParser: A Hybrid Algorithm for Extracting Phylogenies from Dendrograms". In: *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)* 01, pp. 1087–1094. URL: https://www.semanticscholar.org/paper/PhyloParser%3A-A-Hybrid-Algorithm-for-Extracting-from-Lee-Yang/ae7c43ad409a7be5dc0cc59f29d3c30eee068d05.

Mazet, Francoise et al. (Nov. 2003). "Phylogenetic relationships of the Fox (Forkhead) gene family in the Bilateria". In: *Gene* 316. Copyright ©Elsevier 2003, pp. 79–89. DOI: 10.1016/S0378-1119(03)00741-8.

Nakhleh, L. (2013). "Evolutionary Trees". In: *Brenner's Encyclopedia of Genetics (Second Edition)*. Ed. by S. Maloy and K. Hughes. Second Edition. San Diego: Academic Press, pp. 549–550. ISBN: 978-0-08-096156-9. DOI: https://doi.org/10.1016/B978-0-12-374984-0.00504-0. URL: https://www.sciencedirect.com/science/article/pii/B9780123749840005040.

Nalbant, Demet et al. (Feb. 2005). "FAM20: an evolutionarily conserved family of secreted proteins expressed in hematopoietic cells". In: *BMC genomics* 6, p. 11. DOI: 10.1186/1471-2164-6-11.

Olsen, G. (1990). *Gary Olsen's Interpretation of the 'Newick's 8:45' Tree Format Standard*. [Accessed 14-August-2021]. URL: https://evolution.genetics.washington.edu/phylip/newick_doc.html.

Peltier, J.B. et al. (2002). "Central Functions of the Lumenal and Peripheral Thylakoid Proteome of Arabidopsis Determined by Experimentation and Genome-Wide Prediction". In: *The Plant cell* 14, pp. 211–236. DOI: 10.1105/tpc.010304. URL: https://pubmed.ncbi.nlm.nih.gov/11826309/.

Rajangam, Alex S. et al. (Sept. 2008a). "MAP20, a Microtubule-Associated Protein in the Secondary Cell Walls of Hybrid Aspen, Is a Target of the Cellulose Synthesis Inhibitor 2,6-Dichlorobenzonitrile ". In: *Plant Physiology* 148.3, pp. 1283–1294. ISSN: 0032-0889. DOI: 10.1104/pp.108.121913. eprint: https://academic.oup.com/plphys/article-pdf/148/3/1283/37095583/plphys_v148_3_1283.pdf. URL: https://doi.org/10.1104/pp.108.121913.

Rajangam, Alex Selvanayagam et al. (Sept. 2008b). "Evolution of a domain conserved in microtubule-associated proteins of eukaryotes". In: *Advances and applications in bioinformatics and chemistry : AABC* 1, pp. 51–69. DOI: `10.2147/AABC.S3211`.

The NumPy Community (2021). *NumPys Masked arrays*. [Accessed 14-August-2021]. URL: `https://numpy.org/doc/stable/reference/maskedarray.html`.

6 Appendix

The specifics of the results can be found here. The results from specific graphs of first circular phylogenetic trees and then rectangular phylogenetic trees are displayed in this section.

6.1 The Circular Phylogenetic Trees (Graph 4, 6, 8, 11)

This section contains the specifics of testing each of the circular phylogenetic graphs.

6.1.1 Graph 4

The graph 4, figure 6.1 required a small fix to correct root position and a part of the graph which had color 214 on the grayscale, these two modifications were necessary as using threshold of 214 resulted in problems of artifacts. Using the two modifications and a threshold of 175 worked for giving a perfect readout. When trying different values, threshold 130 missed spots in the graph while 180 made numbers overlap it, figure 6.2, (Frygelius et al. 2010).

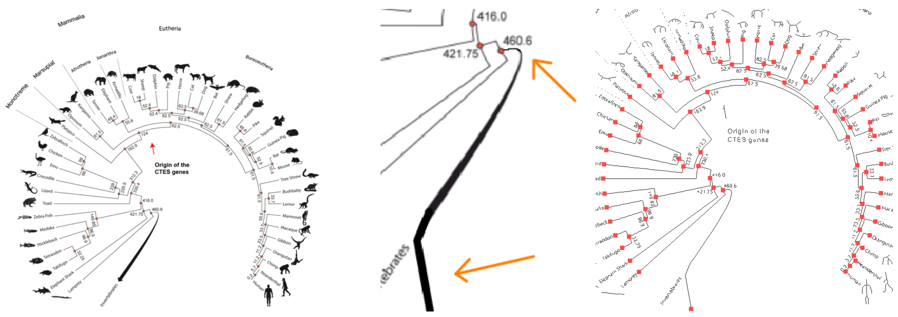
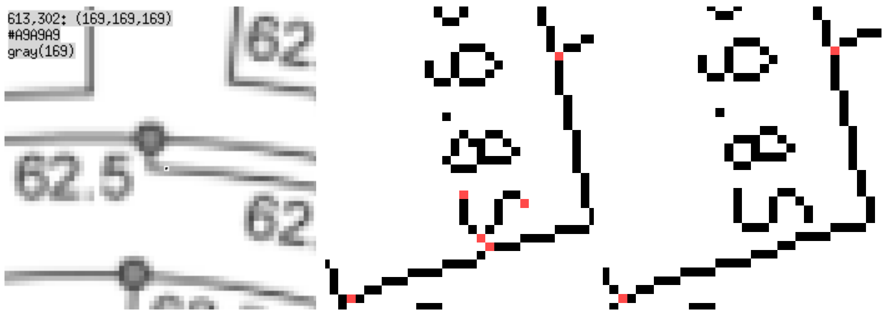


Figure 6.1: *Graph 4 and the result of running the algorithm on it. Red pixels in right image are the intersections and leaves, middle image are the two modifications performed (Frygelius et al. 2010).* ***Republished*** *with permission of Copyright ©John Wiley Sons – Books 2010, from Evolution and human tissue expression of the Cres/Testatin subgroup genes, a reproductive tissue specific subgroup of the type 2 cystatins, Frygelius et al., 12, 3, 2010; permission conveyed through Copyright Clearance Center, Inc.*



```
613,302: (169,169,169)
#A9A9A9
gray(169)
```

Figure 6.2: *Graph 4, where left image had spots missed by threshold 130, middle shows intersection at threshold 180, right shows result using threshold 175. (Frygelius et al. 2010).* ***Republished*** *with permission of Copyright ©John Wiley Sons – Books 2010, from Evolution and human tissue expression of the Cres/Testatin subgroup genes, a reproductive tissue specific subgroup of the type 2 cystatins, Frygelius et al., 12, 3, 2010; permission conveyed through Copyright Clearance Center, Inc.*

6.1.2 Graph 6

Correction to the root were necessary, note that the wrong choice of root does not affect the result as this graph proved to be unusable by the algorithm, figure 6.3.

The tree contained too many intersections where branches were too close to each other at sharp angles, that created cycles for traversal see figure 6.4 as an example. This problem can be mitigated by changing the threshold, but too high threshold yields artifacts which we want to mitigate. For this graph to work the threshold must be above 252 which is too high to yield useful results (Mazet et al. 2003).

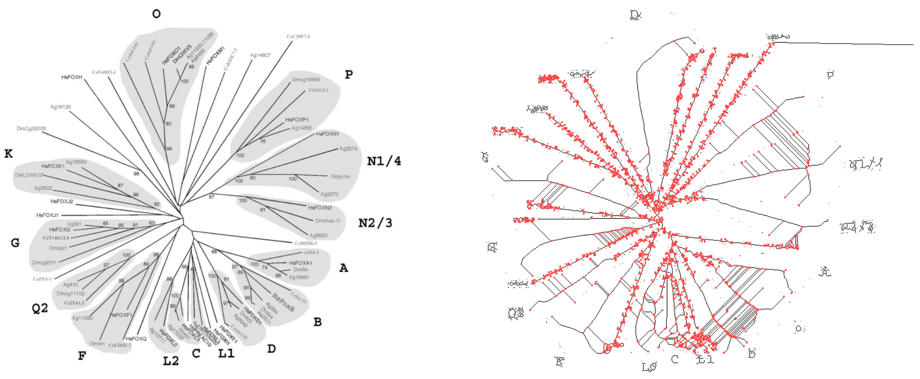


Figure 6.3: *Graph 6, Original image with a modification and the result (Mazet et al. 2003). **This** figure was published in Gene, 316, Mazet et al., Phylogenetic relationships of the Fox (Forkhead) gene family in the Bilateria, 79–89, Copyright ©Elsevier 2003.*

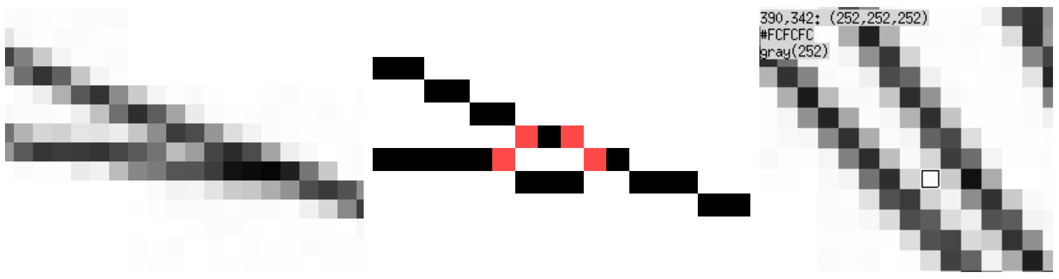


Figure 6.4: *Graph 6, showing the problems of the graph where cycles are created where they should not exist. Red pixels are leaves or intersection found by the algorithm. (Mazet et al. 2003). **This** figure was published in Gene, 316, Mazet et al., Phylogenetic relationships of the Fox (Forkhead) gene family in the Bilateria, 79–89, Copyright ©Elsevier 2003.*

6.1.3 Graph 8

The graph 8, figure 6.5 required modification of root. It had a similar problem to graph 6 where branches are intersecting. One more factor affecting the result were the lines from numbers to parts of the graph, intersecting with the branches, these lines were at least as dark as the graph which means threshold did not solve it (Klinter et al. 2019).

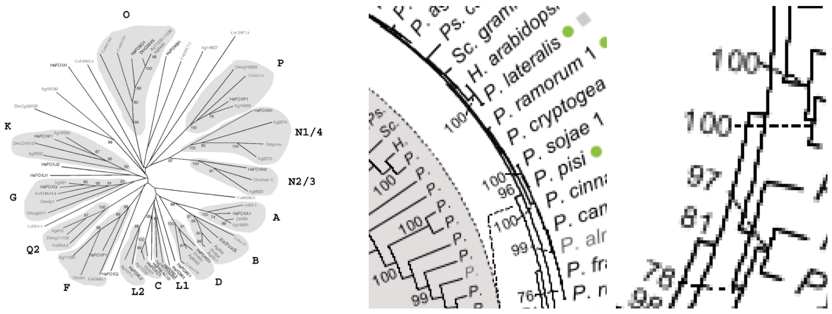


Figure 6.5: *Graph 8, Original image with modification to root and some problem parts (Klinter et al. 2019). **This** figure was published in Molecular Phylogenetics and Evolution, 139, Klinter et al., Diversity and evolution of chitin synthases in oomycetes (Straminipila: Oomycota), 106558, Copyright @Elsevier 2019.*

6.1.4 Graph 11

Some modification to the root were necessary to figure 6.6, the problem were that there existed no threshold that did not create artifacts. This graph did not work unless you performed major modifications to the colors boxes of the image. The results of thinning can be seen in the right image of figure 6.6, (Konrad et al. 2014).

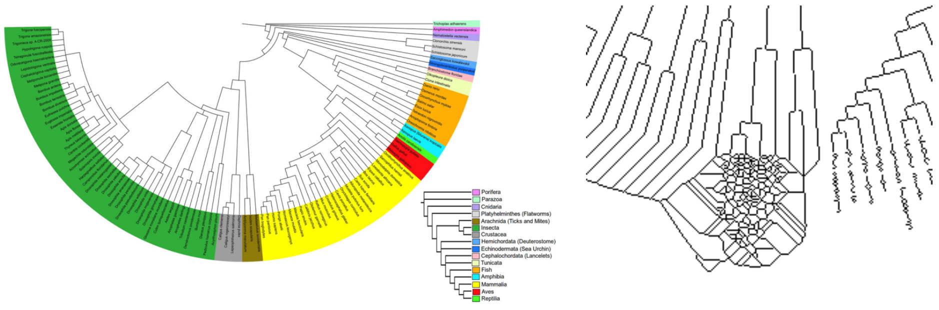


Figure 6.6: *Graph 11, showing original image and the result of the algorithm (Konrad et al. 2014). **Reprinted** by permission from Springer: Nature, Journal of molecular evolution, (The Phylogenetic Distribution and Evolution of Enzymes Within the Thymidine Kinase 2-like Gene Family in Metazoa, Konrad et al.), Copyright @Springer Nature 2014.*

Using threshold 150, holes were left in the graph, this meant an increase of the threshold were necessary but the green background were already passed the threshold which meant it needed to be lowered, figure 6.7.

Because the graph was in contact with the colored box the solution would be to remove all these connections, but that is too many modifications.

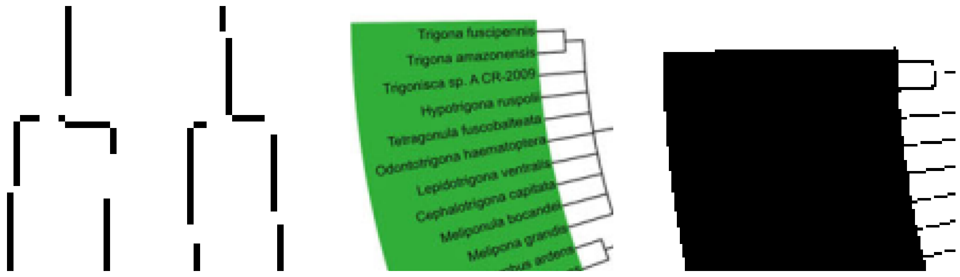


Figure 6.7: *Graph 11, why no threshold works. Holes created at the same time as non-graph parts are overlapping with it (Konrad et al. 2014).* ***Reprinted*** *by permission from Springer: Nature, Journal of molecular evolution, (The Phylogenetic Distribution and Evolution of Enzymes Within the Thymidine Kinase 2-like Gene Family in Metazoa, Konrad et al.), Copyright ©Springer Nature 2014.*

A suggestion of maybe filtering out the colors is possible? Will not be handled in this paper.

6.2 The rectangular phylogenetic Trees (12 Graphs)

This section contains the specifics of testing each of the rectangular phylogenetic graphs.

6.2.1 Graph 1

This graph cannot be shown here due to not clearing the copyright in time but it is figure 3 from the paper of Rajangam et. al. (Rajangam et al. 2008a).

The graph contained multiple non-binary intersections, which would make the graph interpreted incorrectly. Ignoring the problem with non-binary tree, using threshold 130 were enough to yield correct result.

When deciding the threshold, the brightest relevant pixel of the graph were 66 meaning the threshold must be above this value. As some numbers intersected with the graph, the darkest pixels NOT of the graph had value 151, this meant a threshold between these two values.

6.2.2 Graph 2

The algorithm did not work for this graph of figure 6.8, it had non-binary parts and multiple false positives. Some parts intersecting with graph at value 106, while brighter parts of the graph had 143. As seen in figure 6.9, using threshold 150, the graph is visible but the result is not usable without preprocessing where text close to the graph is removed (Frygelius et al. 2010).

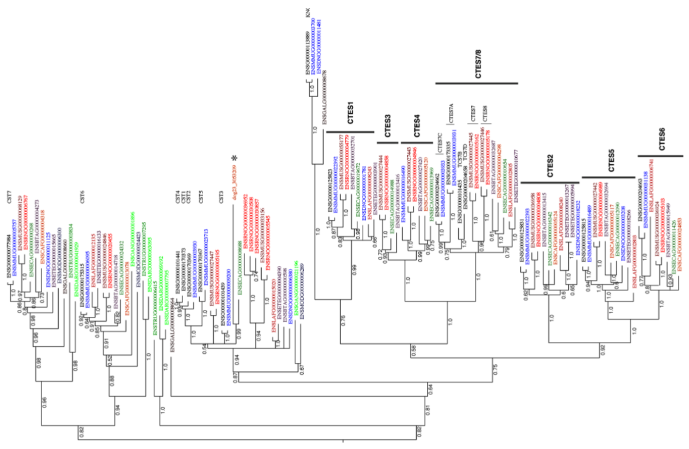


Figure 6.8: *Graph 2, showing the original graph (Frygelius et al. 2010).* ***Republished*** *with permission of Copyright @John Wiley Sons – Books 2010, from Evolution and human tissue expression of the Cres/Testatin subgroup genes, a reproductive tissue specific subgroup of the type 2 cystatins, Frygelius et al., 12, 3, 2010; permission conveyed through Copyright Clearance Center, Inc.*

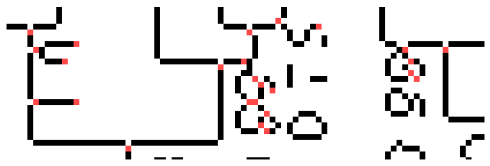


Figure 6.9: *Graph 2, one of the problem areas, red pixels are leaves and intersections found, most of these is text and not part of the graph(Frygelius et al. 2010).* ***Republished*** *with permission of Copyright @John Wiley Sons – Books 2010, from Evolution and human tissue expression of the Cres/Testatin subgroup genes, a reproductive tissue specific subgroup of the type 2 cystatins, Frygelius et al., 12, 3, 2010; permission conveyed through Copyright Clearance Center, Inc.*

6.2.3 Graph 3

This phylogenetic tree in figure 6.10 is not circular, but it also looks different from the other rectangular phylogenetic trees, as it's closer to the rectangular it will be categorized as such. A modification of the root was necessary to circumvent the word "Eutheria". The algorithm worked using threshold 130 and the modification to the root (Frygelius et al. 2010).

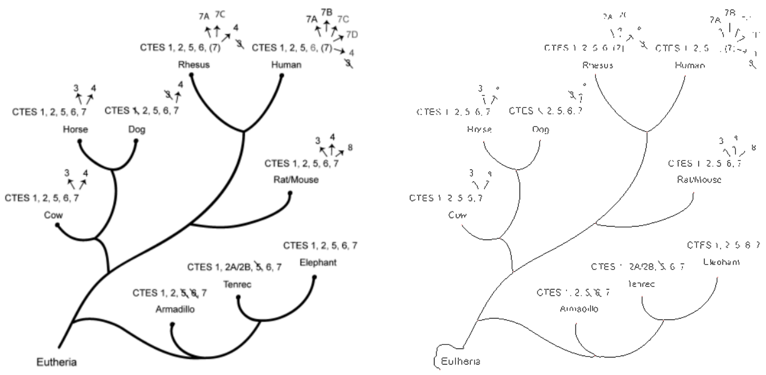


Figure 6.10: *Graph 3, original image and the thinned version (Frygelius et al. 2010).* ***Republished*** *with permission of Copyright @John Wiley Sons – Books 2010, from Evolution and human tissue expression of the Cres/Testatin subgroup genes, a reproductive tissue specific subgroup of the type 2 cystatins, Frygelius et al., 12, 3, 2010; permission conveyed through Copyright Clearance Center, Inc.*

As a side note, in figure 6.11 when threshold 200 was originally used, the first intersection created a cycle.

6.2.4 Graph 5

The tree in figure 6.12 is not binary at root level. Ignoring the non-binary aspect, almost no modifications were necessary and using threshold 35 there were only two misses in the graph. There were 115 leaves in the tree (Mazet et al. 2003).

The first mistake was missing a leaf, figure 6.13, this was removed during thinning as the branch were too short, especially compared to the width of the line. Notice that HsFOXF1 has no leaf in the thinned output. This

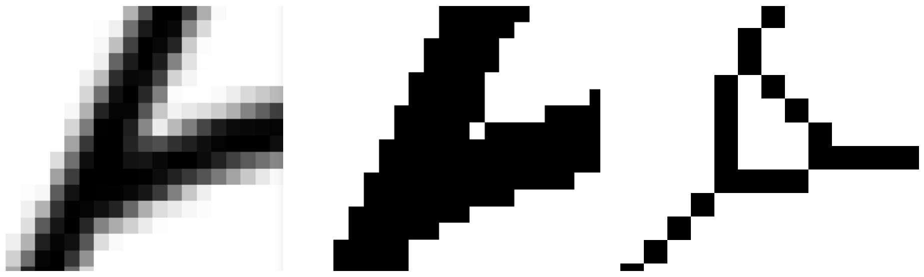


Figure 6.11: *Graph 3, examples where bad threshold may lead to cycles in the graph. From left to right, the cycle zoomed in, as well as before and after thinning (Frygelius et al. 2010).* ***Republished*** *with permission of Copyright @John Wiley Sons – Books 2010, from Evolution and human tissue expression of the Cres/Testatin subgroup genes, a reproductive tissue specific subgroup of the type 2 cystatins, Frygelius et al., 12, 3, 2010; permission conveyed through Copyright Clearance Center, Inc.*

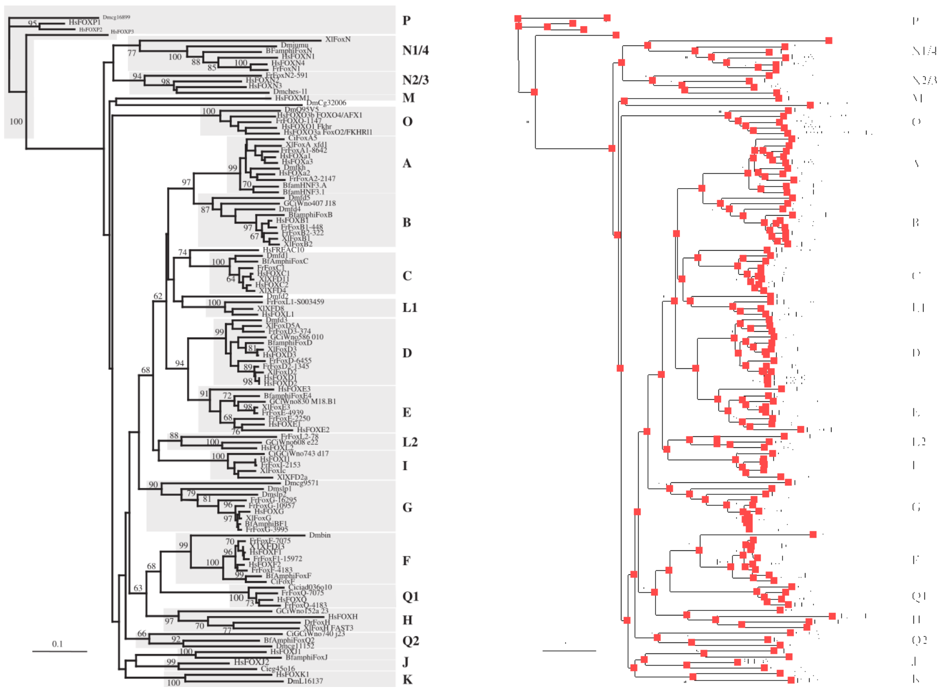


Figure 6.12: *Graph 5, original image and the result of the algorithm, the red pixels are enlarged to display on screen and they represent leaves and intersections (Mazet et al. 2003).* ***This*** *figure was published in Gene, 316, Mazet et al., Phylogenetic relationships of the Fox (Forkhead) gene family in the Bilateria, 79–89, Copyright @Elsevier 2003.*

could also have been solved by the two vertical lines having space between them.

The second mistake, figure 6.14, being the number 1 at branch GCi-

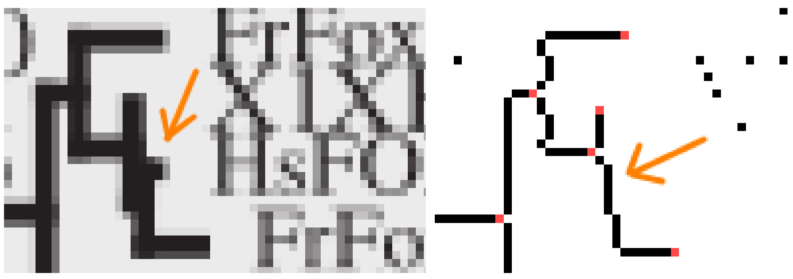


Figure 6.13: *Graph 5, the problem part before and after the algorithm (Mazet et al. 2003).* ***This*** *figure was published in Gene, 316, Mazet et al., Phylogenetic relationships of the Fox (Forkhead) gene family in the Bilateria, 79–89, Copyright @Elsevier 2003.*

Wno608 intersecting with the graph, as the value of the 1 were the same as the graph, modifying the threshold does not help.

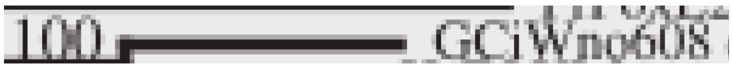


Figure 6.14: *Graph 5, the one in the image were the same color as the graph and required modifications (Mazet et al. 2003).* ***This*** *figure was published in Gene, 316, Mazet et al., Phylogenetic relationships of the Fox (Forkhead) gene family in the Bilateria, 79–89, Copyright @Elsevier 2003.*

When searching for the threshold, branches of the graph were extremely close to each other, making pixels in-between the two branches as dark as 38 while the graph had value 32, see figure 6.15.

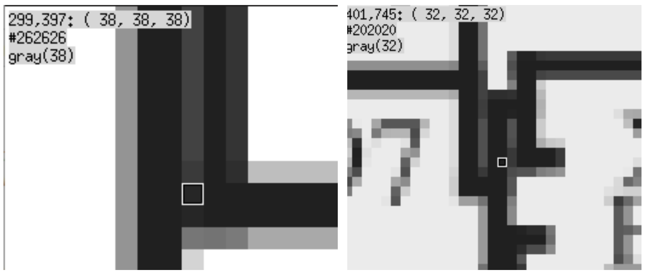


Figure 6.15: *Graph 5, Showing the small difference between some pixels not of the graph and some of the graph (Mazet et al. 2003).* ***This*** *figure was published in Gene, 316, Mazet et al., Phylogenetic relationships of the Fox (Forkhead) gene family in the Bilateria, 79–89, Copyright @Elsevier 2003.*

When using threshold 35, almost all vertices were displayed correctly, except for the two mistakes mentioned earlier.

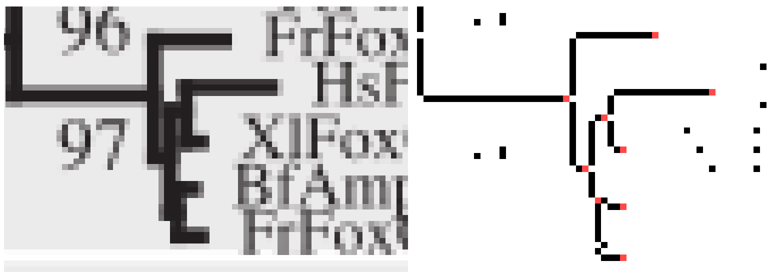


Figure 6.16: *Graph 5, a part of the graph that worked perfectly after the specific threshold. (Mazet et al. 2003).* ***This*** *figure was published in Gene, 316, Mazet et al., Phylogenetic relationships of the Fox (Forkhead) gene family in the Bilateria, 79–89, Copyright @Elsevier 2003.*

6.2.5 Graph 7

The graph 7, figure 6.17, required modification of root by drawing a line and worked flawlessly for threshold 130 (Fugelstad et al. 2009).

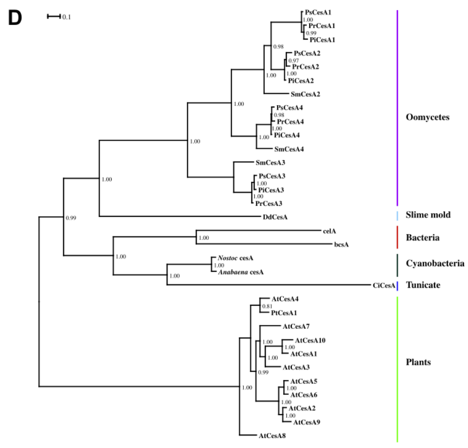


Figure 6.17: *Graph 7, the graph used (Fugelstad et al. 2009).* ***This*** *figure was published in Fungal genetics and biology : FG & B, 46 10, Fugelstad et al., Identification of the cellulose synthase genes from the Oomycete Saprolegnia monoica and effect of cellulose synthesis inhibitors on gene expression and enzyme activity. 759–767, Copyright @Elsevier 2009.*

6.2.6 Graph 9

The image of figure 6.18 required modifications of root. This was not enough as no useful result could be extracted. There were two types of problems preventing the algorithm from working on this tree. The first was the extra lines drawn across and intersecting with the branches, figure 6.18 and 6.19, the second are the black triangles in figure 6.19, (Klinter et al. 2019).

The lines caused problems as their color values were 0 when overlapping the branches, this meant that the threshold could not solve that problem. The black triangles, caused problems as they were interpreted by the algorithm as an intersection with two leaves as children. The unfortunate mistake arose from the thinning algorithm.

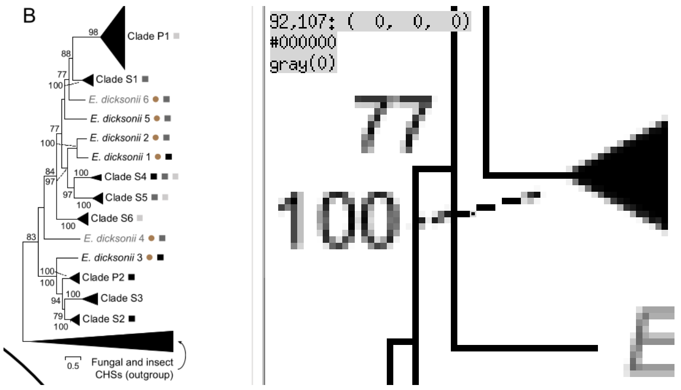


Figure 6.18: *Graph 9, the original image and a problem part, there were multiple problem parts (Klinter et al. 2019).* ***This*** *figure was published in Molecular Phylogenetics and Evolution, 139, Klinter et al., Diversity and evolution of chitin synthases in oomycetes (Straminipila: Oomycota), 106558, Copyright @Elsevier 2019.*

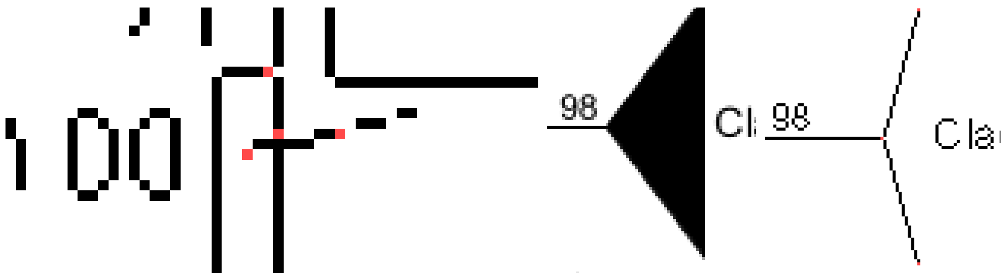


Figure 6.19: *Graph 9, more problem parts (Klinter et al. 2019).* ***This*** *figure was published in Molecular Phylogenetics and Evolution, 139, Klinter et al., Diversity and evolution of chitin synthases in oomycetes (Straminipila: Oomycota), 106558, Copyright @Elsevier 2019.*

6.2.7 Graph 10

Using threshold 150 was enough to extract the data, figure 6.20, (Klinter et al. 2019).

6.2.8 Graph 12

This graph of figure 6.21worked almost perfectly using threshold 130 but as the root were non-binary the output was incorrect, ignoring the non-binary aspect only one flaw arose, modifying this flaw were enough to fix the tree (Nalbant et al. 2005).

The false-positive hits came from the F in leaf FAM20C(Hs) intersecting

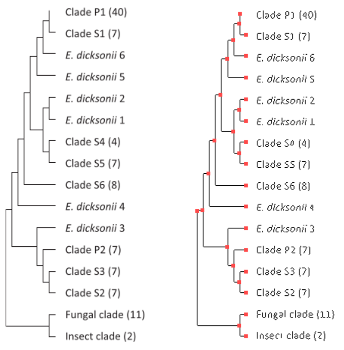


Figure 6.20: *Graph 10, original graph and result (Klinter et al. 2019). **This** figure was published in Molecular Phylogenetics and Evolution, 139, Klinter et al., Diversity and evolution of chitin synthases in oomycetes (Straminipila: Oomycota), 106558, Copyright @Elsevier 2019.*

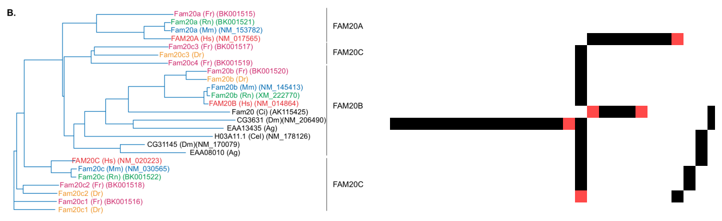
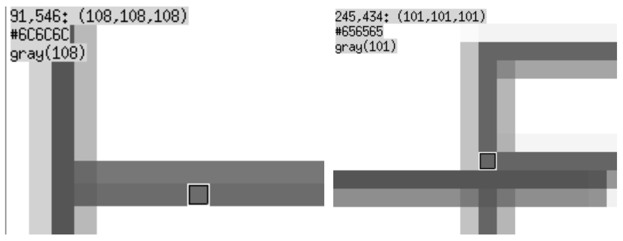


Figure 6.21: *Graph 12, original image and the part requiring modification (Nalbant et al. 2005).*

with the graph, see figure 6.21.

As some parts of the graph were as bright as 108 while the F had color value 101, no threshold will solve this problem, figure 6.22. A small modification of the pixels were enough to fix the problem.



```
91,546: (108,108,108)
#6C6C6C
gray(108)
```

```
245,434: (101,101,101)
#656565
gray(101)
```

Figure 6.22: *Graph 12, left image is branch with color value 108, right is the F that overlaps with the graph, value 101 (Nalbant et al. 2005).*

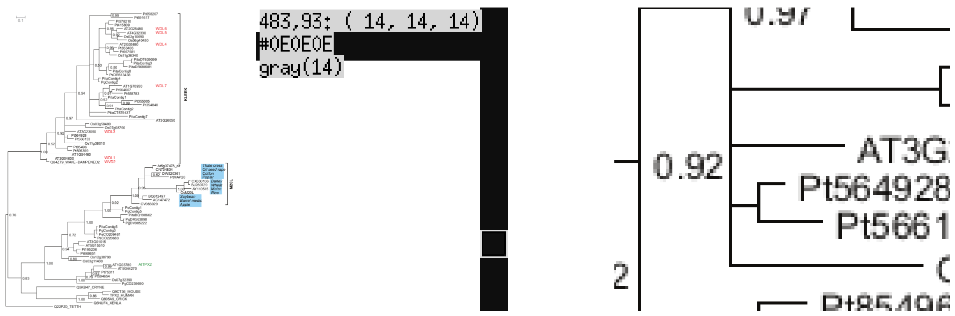
6.2.9 Graph 13

This graph cannot be shown here due to not clearing the copyright in time but it is figure 4 from the paper of Peltier et. al. (Peltier et al. 2002).

Graph 13 works flawlessly at threshold 130.

6.2.10 Graph 14

Using threshold 14 worked for figure 6.23. It was not a binary tree and disregarding that aspect all 78 leaves were found and the algorithm worked (Rajangam et al. 2008b).



```
483,93: ( 14, 14, 14)
#0E0E0E
gray(14)
```

Figure 6.23: *Graph 14, original image, part of graph of color 14 and non-binary parts of the graph (Rajangam et al. 2008b)*

6.2.11 Graph 15

This graph of figure 6.24 worked using threshold 130. The graph is non-binary but disregarding that aspect, output was correct.

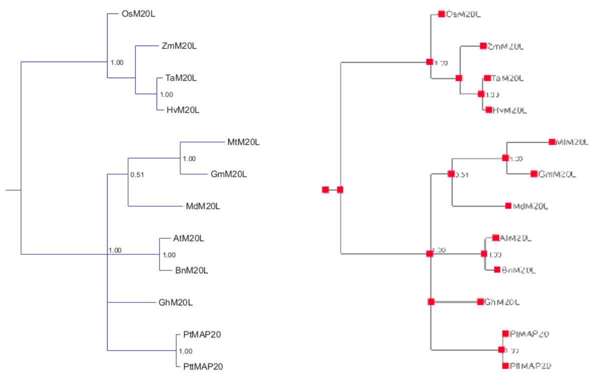


Figure 6.24: *Graph 15, original graph and the result (Rajangam et al. 2008b)*

6.2.12 Graph 16

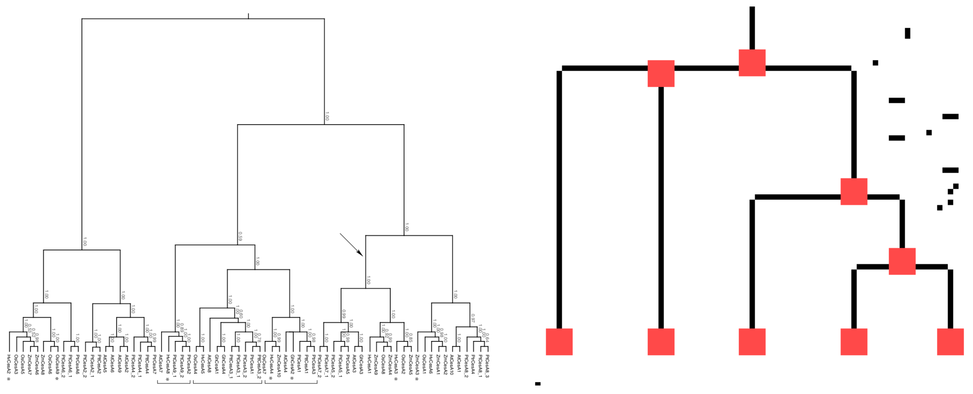


Figure 6.25: *Graph 16, original graph and the single non-binary part of the graph, red pixels are enlarged to be clearly visible (Djerbi et al. 2005).* ***Reprinted*** *by permission from Springer: Nature, Planta, (The genome sequence of black cottonwood (Populus trichocarpa) reveals 18 conserved cellulose synthase (CesA) genes, Djerbi et al.), Copyright ©Springer Nature 2005.*

A threshold of 100 were necessary in figure 6.25. There was one intersection with three children but disregarding this non-binary aspect, the output was correct and returned 70 leaves.

Datalogi
August 2021

www.math.su.se

Beräkningsmatematik
Matematiska institutionen
Stockholms universitet
106 91 Stockholm