Project 5: NJ tree construction

Cæcilia, Lea & Peter

Introduction:

We tried to implement the Saitou and Nei algorithm, but we had some problems with it. Therefore, our implementation only contained parts of that algorithm. Furthermore, the implementation we did was not very efficient, as we could see when we were calculating the running time of it and compared it to the RapidNJ and Quicktree algorithm. As our implementation was not very efficient we could not run all the distance matrices as we needed, because it took too much time. We are a bit surprised that the RF-distance between our program and the two other implementations are significantly different since we get the exact same tree as on the slides when we used the distance matrix from the slides.

Implementation:

We have implemented NJ using the Python language and the Saitou and Nei algorithm from the slides. First we made a function to recursively build a Newick-format string from an adjacency list and then a function to write a tree as a Newick-format string. Then we also made a function to find the coordinates of the minimum value in the N matrix.

Then we have our nj function, which takes two arguments (distance matrix and taxon labels). Here we begin by calculating the N matrix. Hereafter, we produce the tree from the distance matrix. This we do by first finding the internal nodes and then finding the leaves that are closest and need to be clustered. Then we calculate the branch lengths (weights) and add the edges and branch lengths to the tree. When we get to a node without parents, we stop calculating. Then we make a new distance matrix with matrix length - 1 dimension. We add the internal nodes to the matrix and then skip the rows and columns already merged. Then we update the distance matrix to be the new distance matrix and we update the length of the matrix to be one less. At the end we save the final tree in newick format.

Machine and running time:

We have used a Macbook Air with 8 GB RAM and a dual core with a speed of 1,6 GHz.

To measure the running time of the quicktree program and the rapidNJ program, we used the 'time' command in the terminal and noted the real time.

To measure the running time of our program, we used the following code:

```python
start = time.time()

nj(dist_mat, tax_lab)

end = time.time()

print("The time of execution of above program is :", round(end-start, 3))
```

Results:

Dist mat	Running time QT	Running time RNJ	Running time our program	Speed-up relative to QT	Speed-up relative to RNJ	RF-distance QT and our program	RF-distance RNJ and our program	RF-distance RNJ and QT
89	0.013s	0.010s	1.279s	0,0102	0,0078	130.0	142.0	30.0
214	0.043s	0.021s	19.203s	0,0022	0,0011	376.0	366.0	56.0
304	0.071s	0.036s	58.574s	0,0012	0,0006	558.0	548.0	78.0
401	0.111s	0.045s	145.03s	0,0008	0,0003	744.0	734.0	98.0
494	0.143s	0.057s	287.56s	0,0005	0,0002	958.0	958.0	532.0
608	0.184s	0.084s	553.48s	0,0003	0,0002	1174.0	1172.0	20.0
777	0.356s	0.105s	-	-	-	-	-	494.0
877	0.780s	0.150s	-	-	-	-	-	54.0
1347	1.842s	0.361s	-	-	-	-	-	2.0
1493	2.136s	0.425s	-	-	-	-	-	92.0
1560	2.319s	0.404s	-	-	-	-	-	152.0
1689	2.770s	0.499s	-	-	-	-	-	102.0
1756	3.193s	0.646s	-	-	-	-	-	76.0
1849	3.651s	0.594s	-	-	-	-	-	280.0