Elements of Computational Biology
Subject 15: Distance phylogenetics: UPGMA and NJ

Agata Radys, Paweł Cejrowski, Łukasz Myśliński

December 21, 2016

1 Usage

Program was developed in Java programming language without any external libraries. All sources are available on Github and compiled using `Maven`.

Listing 1: Building project

```
git clone git@github.com:MiSSLab/BioComp15.git
mvn package
```

Created Java archive can be run using `JRE`. Sample data can be found in directory `resources/`.

Listing 2: Running project using data1.matrix

```
java -jar -Dfilename="resources/data1.matrix" \
    target/distance-phylogenetics-jar-with-dependencies.jar
```

2 Data formats

2.1 Input

Application requires `CSV` data format and quadratic matrix of distances.

Listing 3: Example data file content

```
a,b,c,d,e
0,8,8,5,3
8,0,3,8,8
8,3,0,8,8
5,8,8,0,5
3,8,8,5,0
```

Labels in header has to be lexicographically sorted, dense vector with every column matching `"[a-z]+"`.

2.2 Output

Resulting trees are printed in ASCII-art to the STDOUT.

Listing 4: UPGMA output

```
UPGMA(resources/data1.matrix)
[[8.0]]
|-[[5.0]]
|   |-[[3.0]]
|   |   |---- [[a]]
|   |   `---- [[e]]
|   `---- [[d]]
`-[[3.0a]]
    |---- [[b]]
    `---- [[c]]

      a b c d e 8.0 5.0 3.0 3.0a
a    |0 0 0 0 0 0   0   1   0
b    |0 0 0 0 0 0   0   0   1
c    |0 0 0 0 0 0   0   0   1
d    |0 0 0 0 0 0   1   0   0
e    |0 0 0 0 0 0   0   1   0
8.0  |0 0 0 0 0 0   1   0   1
5.0  |0 0 0 1 0 1   0   1   0
3.0  |1 0 0 0 1 0   1   0   0
3.00 |0 1 1 0 0 1   0   0   0

UPGMA-canonical
[[a]]
`-[[3.0]]
     |---- [[e]]
     `-[[5.0]]
         |---- [[d]]
         `-[[8.0]]
             `-[[3.0a]]
                 |---- [[b]]
                 `---- [[c]]
```

Despite the fact that NJ returns unrooted tree it is presented as a rooted one with particular node choosen as a root.

Listing 5: NJ output

```
NJ(resources/data1.matrix)
[[g]]
|---- [1.5-<-[a]]
|---- [1.5-<-[e]]
‘-[1.75-<-[h]]
    |---- [1.0-<-[d]]
    ‘-[4.0-<-[f]]
        |---- [1.5-<-[b]]
        ‘---- [1.5-<-[c]]

   a b c d e g h f
a|0 0 0 0 0 1 0 0
b|0 0 0 0 0 0 0 1
c|0 0 0 0 0 0 0 1
d|0 0 0 0 0 0 1 0
e|0 0 0 0 0 1 0 0
g|1 0 0 0 1 0 1 0
h|0 0 0 1 0 1 0 1
f|0 1 1 0 0 0 1 0

NJ-canonical
[[a]]
‘-[[g]]
    |---- [[e]]
    ‘-[[h]]
        |---- [[d]]
        ‘-[[f]]
            |---- [[b]]
            ‘---- [[c]]
```

3 Algorithms

3.1 UPGMA (ang. Unweighted Pair Group Method with Arithmetic Mean)

Data: ultrametric matrix d for set L.

Listing 6: UPGMA pseudocode

```
clusters[|L|]
while (clusters.length > 1):
    calculate distances between clusters
        (sum of distances between cluster members
            divided by product of cluster cardinalities)
    find the lowest distance
    merge the closest clusters
```

3.2 NJ

Data: ultrametric matrix d for set L.

Q - matrix: $Q(i,j) = (n-2)d(i,j) - \sum_{k=1}^{n} d(i,k) - \sum_{k=1}^{n} d(j,k)$

Distance from the pair members to the new node:

$d'(f,u) = \frac{1}{2}d(f,g) + \frac{1}{2(n-2)}(\sum_{k=1}^{n} d(f,k) - \sum_{k=1}^{n} d(g,k))$

$d(g,u) = d(f,g) - d'(f,u)$

Listing 7: NJ pseudocode

```
clusters[|L|]
while (number of clusters > 2):
    calculate Q-matrix
    find the lowest q-distance
    merge the q-closest clusters
    update distances
merge last 2 clusters
```

3.3 Creaing adjacency matrix

Both rooted and unrooted trees created via UPGMA and NJ algorithms can be transformed to adjacency matrix. The algorithm for that is as follows:

Listing 8: Tree to adjacency matrix.

```
function walk(Node rootNode):
    int x = adjacencyMatrix.establishPosition(rootNode)
    for (node : rootNode.getChildren ):
        int y = adjacencyMatrix.establishPosition(node)
        adjacencyMatrix[x][y] = 1;
        adjacencyMatrix[y][x] = 1;
        walk(node);

init adjacency matrix with 0;
walk(tree.rootNode)
```

3.4 Creating canonical tree

To compare trees, we need its canonical form. It will be a tree with the first node from input data as a root.

Listing 9: Tree to adjacency matrix.

```
function toCanonicalTree:
    List<Integer> visited = new ArrayList<>();
    Node rootNode = createNodes(0, visited);
    return new Tree(rootNode);

function Node createNodes(int i, List<Integer> visited):
    visited.add(i);
    Node node = new Node(adjacencyMatrix.header[i]);
    for (j = 0; j < adjacencyMatrix.header.length; j++):
        if (visited.notContains(j) && adjacencyMatrix[i][j] = 1):
            node.children.add(createNodes(j, visited));
    return node;
```

3.5 Tree comparison

Trees in canonical form can be easily compared using the following algorithm.

Listing 10: Comparing trees.

```
function equals(Node aNode, Node bNode):
    if (aNode.children.size = bNode.children.size):
        if (aNode.children.size == 0 && .children.size  == 0) {
            if (aNode.label = bNode.label) {
                return true
            else:
                return false
        else:
            comparisons = []
            for (aChild : aNode.children):
                comparisons2 = []
                for (bChild : bNode.children):
                    boolean equals = equals(aChild, bChild)
                    comparisons2.add(equals)
                comparisons.add(comparisons2.reduce(or))
            return comparisons.reduce(and)
    else:
        return false;
```

An important assumption in this algorithm is that we can compare labels of leaf-nodes, because they are those given as an input so they have to be equal.