Cytoscape Color Coding App

G Sanjay, Sriram Goutam P

Indian Institute of Technology Madras

26/04/2023

Introduction - Color coding algorithm

- To perform basic functions cells need to sense their surrounding environment.
- This is done via signalling pathways, hence knowing the routes via which cells relay information is an important biological question.
- **Computational problem**: Given a set of source nodes and destination nodes, Find the best cost simple paths of length k in a given protein-protein interaction network.
- In a protein-protein interaction network nodes represent the proteins and value of the edges represent the probability that the two nodes interact
- Color coding algorithm is used to find maximum cost k length paths. In the algorithm we assign colors randomly to nodes and find a path with different colors.
- This process is repeated many times and the best path is taken.

Introduction to Cytoscape and Motivation

- Cytoscape is a software used for visualization of graphs.
- Using plugins like STRING we can import and analyze large biological graphs in Cytoscape.
- Color coding algorithm works for large datasets where traditional DP algorithms take a long time.
- Hence Color coding algorithm would be very useful to get pathways in the PPI datasets in Cytoscape.
- Hence we thought that a Color Coding plugin would be perfect to implement.

Related literature

- We used two papers as references for our mini-project:
  - *Efficient Algorithms for Detecting Signaling Pathways in Protein Interaction Networks*
  - *Algorithm Engineering for Color-Coding with Applications to Signaling Pathway Detection*
- The first paper uses color coding algorithm for finding signalling pathways by taking the number of colors equal to the path length.
- The algorithm is tested on the Yeast data set and the time taken is measured for different path lengths.
- The second paper improvises on the first by changing the number of colors from k to $1.3k$

Problem statement

- To create a Color coding plugin for Cytoscape
- The user will give a PPI graph with edge weights, source and destination vertices to the app.
- The app will then highlight the source and destination vertices in the graph.
- The value of the path and the path itself is returned for all destination vertices.
- The app also returns the time taken for the computation by the algorithm.

Approach to problem

- The code for the app is written in *Java*. There are two parts in this, the algorithm itself and the interface for Cytoscape.

```
// Fetching current CytoScape network
CyNetwork net = cyApplicationManager.getCurrentNetwork();
CyNetworkView networkView = cyApplicationManager.getCurrentNetworkView();
if (net == null)
    throw new NullPointerException(s:"network is null");

//Fetching all rows of network.
Collection<CyRow> rows = net.getDefaultNodeTable().getAllRows();
for (CyRow row : rows) {
    CyNode node = net.getNode(row.get(CyTable.SUID, type:Long.class));

    String myNodeName = net.getRow(node).get(CyNetwork.NAME, type:String.class);
    View<CyNode> nodeView = networkView.getNodeView(node);

    //coloring source nodes to green and destination nodes to red.
    if (sourcenodes.contains(myNodeName)) {
        nodeView.setVisualProperty(BasicVisualLexicon.NODE_FILL_COLOR, Color.green);
    } else if (destinationnodes.contains(myNodeName)) {
        nodeView.setVisualProperty(BasicVisualLexicon.NODE_FILL_COLOR, Color.red);
    }
    // else{
    // nodeView.setVisualProperty(BasicVisualLexicon.NODE_FILL_COLOR, Color.white);
    // }
    nodeView.setVisualProperty(BasicVisualLexicon.NODE_BORDER_LINE_TYPE, LineTypeVisualProperty.SOLID);
    nodeView.setVisualProperty(BasicVisualLexicon.NODE_BORDER_PAINT, Color.BLACK);
    nodeView.setVisualProperty(BasicVisualLexicon.NODE_BORDER_WIDTH, value:10.0);
    nodeView.setVisualProperty(BasicVisualLexicon.NODE_LABEL, myNodeName);
    networkView.updateView();
}
```

Figure 1: Fetching the network from Cytoscape

Approach to problem – continued

```java
//for building graph as adjacency list
ArrayList<ArrayList<Integer>> graph_adjlist = new ArrayList<ArrayList<Integer>>(numvertices);
for (int i = 0; i < numvertices; i++) {
    graph_adjlist.add(new ArrayList<Integer>());
}

for (CyRow row : rows) {
    CyNode node = net.getNode(row.get(CyTable.SUID, type:Long.class));
    List<CyEdge> adj = net.getAdjacentEdgeList(node, CyEdge.Type.ANY);
    for (CyEdge edge : adj) {

        CyNode source = edge.getSource();
        CyNode target = edge.getTarget();

        String sourceNodeName = net.getRow(source).get(CyNetwork.NAME, type:String.class);
        String targetNodeName = net.getRow(target).get(CyNetwork.NAME, type:String.class);

        Integer sourcenum = stringtoint.get(sourceNodeName);
        Integer targetnum = stringtoint.get(targetNodeName);
        if (!graph_adjlist.get(sourcenum).contains(targetnum)) {
            graph_adjlist.get(sourcenum).add(targetnum);
        }
        if (!graph_adjlist.get(targetnum).contains(sourcenum)) {
            graph_adjlist.get(targetnum).add(sourcenum);
        }
    }
}
```

Figure 2: Creating the graph as an adjacency list

Results obtained

- The network that we tested on has 2407 nodes and 6511 edges. The results are shown below

Path length	Time taken(ms)
4	1119.47
5	5589.49
6	46402.59
7	101199.81
8	502951.27

Results obtained – of the first paper

- The graph that the first paper used has 4500 nodes and 14319 edges

Path length	Time taken(s)
6	32
7	97
8	498

Future work

- The number of colors assigned can be changed to 1.3 times pathlength according to more recent papers on colorcoding to reduce the time taken for execution.
- Proposing this to the App community of Cytoscape.