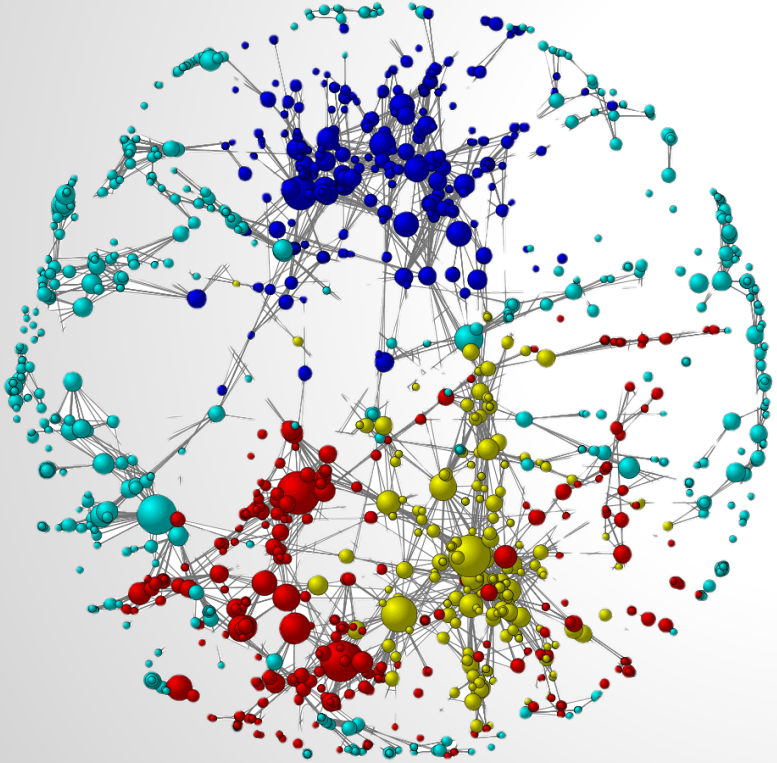
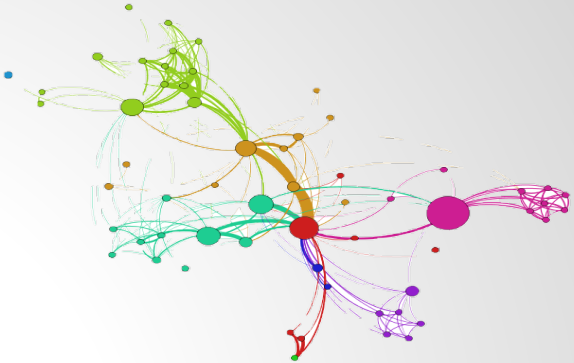
Gephi





Team: Hypergraphs

Aggelos Gickas <aggelosGks@gmail.com>

Charalampos Tsiligiannis <htsiligiannis@gmail.com>

Contributions

Modify node painter tool so color does not increase incrementally

Change method pressingNodes

```
@@ -83,12 +83,9 @@ public void unselect() {
83  83          public void pressingNodes(Node[] nodes) {
84  84              color = painterPanel.getColor().getColorComponents(color);
85  85              for (Node node : nodes) {
86      -                float r = node.getNodeData().r();
87      -                float g = node.getNodeData().g();
88      -                float b = node.getNodeData().b();
89      -                r = intensity * color[0] + (1 - intensity) * r;
90      -                g = intensity * color[1] + (1 - intensity) * g;
91      -                b = intensity * color[2] + (1 - intensity) * b;
    86  +                float r = color[0];
    87  +                float g = color[1];
    88  +                float b = color[2];
92  89                  node.getNodeData().setR(r);
93  90                  node.getNodeData().setG(g);
94  91                  node.getNodeData().setB(b);
```

When computing shortest paths save paths (as String) in column

Add method createShortestPathLabel

```
223 +    public void createShortestPathLabel(Node target, AbstractShortestPathAlgorithm algorithm) {
224 +        ArrayList<Node> path = algorithm.getShortestPathForTargetNode(target);
225 +        //Reverse the path to iterate from the source node to the target node
226 +        Collections.reverse(path);
227 +        StringBuilder sb = new StringBuilder(1024);
228 +
229 +        //Iterate from source node to target node
230 +        for (Node node : path) {
231 +
232 +            //If the node doesn't have a label use the unique id
233 +            if ((node.getNodeData().getLabel() == null) ||
234 +                (node.getNodeData().getLabel().trim().isEmpty())){
235 +                //Append the id of the node
236 +                sb.append(node.getNodeData().getId());
237 +            }else {
238 +                //Append the label of the node
239 +                sb.append(node.getNodeData().getLabel());
240 +            }
241 +
242 +            //If not the last node
243 +            if (node != target) {
244 +                //Add arrow symbol to the sequence
245 +                sb.append("\u21D2");
246 +            }
247 +        }
248 +
249 +        String shortestPathLabel = sb.toString();
250 +        //Update the shortest path attribute column
251 +        updateShortestPathColumn(shortestPathLabel);
252 +    }
```

Add method updateShortestPathColumn

```
261 +    public void updateShortestPathColumn(String shortestPathLabel) {
262 +        AttributeModel attributeModel = Lookup.getDefault().lookup(AttributeController.class).getModel();
263 +        AttributeTable nodeTable = attributeModel.getNodeTable();
264 +        AttributeColumn shortColumn = nodeTable.getColumn(SHORTEST_PATH);
265 +
266 +        //Check if the column exists
267 +        if (shortColumn == null) {
268 +            shortColumn = nodeTable.addColumn(SHORTEST_PATH, NbBundle.getMessage(ShortestPath.class, "ShortestPath.name"),
269 +        }
270 +
271 +        //Add data to row given specific source node
272 +        AttributeRow row = (AttributeRow) this.sourceNode.getNodeData().getAttributes();
273 +        row.setValue(shortColumn, shortestPathLabel);
274 +    }
```

Support edge weight when computing centrality metrics (Closeness, Betweenness and Eccentricity)

Add method calculateWeightedMetrics

```
309     public void calculateWeightedMetrics(Graph hgraph) {
310         AbstractShortestPathAlgorithm algorithm;
311         int countPaths = 0;
312         int countProgress = 0;
313         double eccentricityValue;
314         double totalDistance = 0;
315         double minEccentricity = Double.MAX_VALUE;
316         double maxEccentricity = Double.MIN_VALUE;
317 
318         Progress.start(this.progress, hgraph.getNodeCount());
319 
320         //Iterate over the nodes of the graph
321         for (Node sourceNode : hgraph.getNodes()) {
322             if (isDirected()) {
323                 algorithm = new BellmanFordShortestPathAlgorithm(hgraph, sourceNode);
324             }else {
325                 algorithm = new DijkstraShortestPathAlgorithm(hgraph, sourceNode);
326             }
327 
328             algorithm.compute();
329 
330             //Extract all the paths for each node
331             for (Node currentNode : hgraph.getNodes()) {
332                 if (currentNode != sourceNode) {
333                     ArrayList<Node> paths = algorithm.getShortestPathForTargetNode(currentNode);
334                     this.avgDist += algorithm.getDistanceShortestPathForTargetNode(currentNode);
335                     countPaths++;
336 
337                     //Iterate the path and check if a node is inside tha path
338                     for (int i = 0; i < paths.size(); i++) {
339                         if ((i != 0) && (i != paths.size() - 1)) {
340                             //Increment the frequency
341                             this.betweennessWeighted.put(paths.get(i),  this.betweennessWeighted.get(paths.get(i)) + 1)
342                         }
343                     }
344                 }
345             }
```

```
347             //Calculate closeness
348             for (Edge edge : hgraph.getEdges(sourceNode)) {
349                 //sum up all edges weigth
350                 totalDistance += edge.getWeight();
351             }
352 
353             this.closenessWeighted.put(sourceNode, (1 /(double)totalDistance));
354 
355             //Calculate eccentricity
356             eccentricityValue = algorithm.getMaxDistance();
357             this.eccentricityWeighted.put(sourceNode,eccentricityValue);
358 
359 
360             minEccentricity = Math.min(eccentricityValue, minEccentricity);
361             maxEccentricity = Math.max(eccentricityValue, maxEccentricity);
362 
363             countProgress++;
364 
365             if (this.isCanceled) {
366                 hgraph.readUnlockAll();
367                 return;
368             }
369 
370             //Update progress
371             Progress.progress(this.progress, countProgress);
372         }
373 
374         setBetweenessValues(countPaths, hgraph);
375         assignGraphFields(minEccentricity, maxEccentricity);
376     }
377 
```

Minimum Spanning Tree Algorithm (Kruskal Algorithm)

Add methods execute & markNodes

```
93      /**
94       * Implements and executes Kruskal's algorithm for finding the minimum spanning tree.
95       */
96      public void execute(){
97          int edges_selected=0;
98          while(edges_selected<N-1){
99              EdgeWeighted edgew=edgesWeighted.remove();//remove edge with minimum weight
100             if(!createsCycle(edgew.edge)){//if addition is feasible
101                 Edge next_edge=edgew.edge;
102                 MSP.add(next_edge);//add edge to MSP permutation
103                 markNodes(next_edge);
104                 edges_selected++;
105             }
106         }
107
108     }
109
110     /**
111      * Marks the source and target node of an edge added to MSP, to prevent creation of cycles.
112      * @param edge the edge just added to the MSP
113      */
114     private void markNodes(Edge edge){
115          Node source=edge.getSource();
116          Node target=edge.getTarget();
117          int source_index=indexes.get(source);
118          int target_index=indexes.get(target);
119          marked[source_index]=true;//assign true
120          marked[target_index]=true;
121     }
```

Add private class EdgeWeighted

```
152     private static class EdgeWeighted implements Comparable<EdgeWeighted>{
153
154         public final Edge edge;
155         public final float weight;
156
157         public EdgeWeighted(Edge edge,float weight){
158             this.edge=edge;
159             this.weight=weight;
160         }
161
162
163         public int compareTo(EdgeWeighted other) {
164              //check distances first
165              if (weight < other.weight){
166                  return -1;
167              }
168              else if (weight > other.weight){
169                  return 1;
170              }
171              //if distances are the same break the tie with edge id
172              else if(edge.getId()<other.edge.getId()){
173                  return 1;
174              }else if(edge.getId()>other.edge.getId()){
175                  return -1;
176              }
177              else{
178                  return 0;
179              }
180          }
181
182     }
```

Hypergraphs

Thank you!Questions?