

```

1 package ch.ethz.icr.prefattach;
2
3 import uchicago.src.sim.engine.SimInit;
4 import uchicago.src.sim.engine.SimpleModel;
5 import uchicago.src.sim.gui.RectNetworkItem;
6 import uchicago.src.sim.network.*;
7 import java.util.ArrayList;
8 import java.util.Iterator;
9
10 /**
11  * Preferential Attachment model
12  *
13  * @author Luc Girardin
14  * @author Lars-Erik Cederman
15  * @author Laszlo Gulyas
16  * @version 2.0
17  */
18 public class Model extends SimpleModel {
19     int initNodes, m;
20     int numEdges, numNodes;
21
22     double pRemoving;
23     int removingStarts;
24
25     public void setup() {
26         super.setup();
27         initNodes = 3;
28         m = 2;
29         numEdges = 0;
30         numNodes = 0;
31         pRemoving = 0;
32         removingStarts = 100000;
33     }
34
35     /**
36     * Creates a new edge
37     */
38     protected void createEdgeBetween(DefaultNode n1, DefaultNode n2) {
39
40         DefaultDrawableEdge edge = new DefaultDrawableEdge(n1, n2);
41         n1.addOutEdge(edge);
42         n2.addInEdge(edge);
43
44         // We keep track of the number of edges for efficiency reasons.
45         numEdges++;
46     }
47
48     /**
49     * This method creates a new node with the given ID
50     */
51     protected DefaultNode createNewNode(int ID) {
52         numNodes++;
53
54         // We create a node (which is, in fact, prepared to be displayed).
55         DefaultDrawableNode n = new DefaultDrawableNode(Integer.toString(ID),
56             new RectNetworkItem(10, 10));
57         return n;
58     }
59
60     /**
61     * Remove a node from the list of list of agents
62     */
63     protected void removeNode(DefaultNode node) {

```

```

64     ArrayList outEdges = node.getOutEdges();
65     Iterator it = outEdges.iterator();
66     while (it.hasNext()) {
67         DefaultEdge edge = (DefaultEdge) it.next();
68         DefaultNode other = (DefaultNode) edge.getTo();
69         other.removeInEdge(edge);
70     }
71     node.clearOutEdges();
72
73     // Remove all the edges that direct to this node
74     ArrayList inEdges = node.getInEdges();
75     it = inEdges.iterator();
76     while (it.hasNext()) {
77         DefaultEdge edge = (DefaultEdge) it.next();
78         DefaultNode other = (DefaultNode) edge.getFrom();
79         other.removeOutEdge(edge);
80     }
81     node.clearInEdges();
82
83     agentList.remove(node);
84 }
85
86 public void buildModel() {
87     super.buildModel();
88
89     // We create the initial nodes
90     for (int i = 0; i < initNodes; i++) {
91         agentList.add(createNewNode(i));
92     }
93
94     // We connect all nodes to all other nodes
95     Iterator it = agentList.iterator();
96     while (it.hasNext()) {
97         DefaultNode n = (DefaultNode) it.next();
98         Iterator it2 = agentList.iterator();
99         while (it2.hasNext()) {
100             DefaultNode other = (DefaultNode) it2.next();
101
102             if ((other != n) && (!other.hasEdgeTo(n))) {
103                 createEdgeBetween(n, other);
104             }
105         }
106     }
107 }
108
109 /**
110  * This method selects an 'old' node with the appropriate probability.
111  * (See Equation (77) in Albert & Barabasi, 2001.)
112  */
113 public DefaultNode selectNode() {
114
115     // The node to be selected
116     DefaultNode n = null;
117
118     // The sum of degrees in the graph. We assign the value based on the
119     // number of edges, instead of summing it up each time.
120     int sumDegrees = numEdges * 2;
121
122     double rand = this.getNextDoubleFromTo(0.0, 1.0);
123
124     double cumProb = 0.0;
125     boolean done = false;
126     Iterator i = agentList.iterator();

```

```

127     while (!done && i.hasNext()) {
128         n = (DefaultNode) i.next();
129         // The probability of selecting the given node.
130         double prob = (double) (n.getInDegree() + n.getOutDegree()) /
131             (double) (sumDegrees);
132         cumProb = cumProb + prob;
133
134         if (rand < cumProb)
135             done = true;
136     }
137     return n;
138 }
139
140 public void step() {
141     // Removing nodes
142     if (getTickCount() > removingStarts) {
143         int numRemoved = (int) ((double) agentList.size() * pRemoving);
144         System.out.println(numRemoved);
145         for (int i = 0; i < numRemoved; i++) {
146             int index = getNextIntFromTo(0, agentList.size() - 1);
147             DefaultNode node = (DefaultNode) agentList.get(index);
148             removeNode(node);
149         }
150     }
151
152     DefaultNode newNode = createNewNode(numNodes + 1);
153
154     // We create the links for it
155     for (int i = 0; i < m; i++) {
156         DefaultNode oldNode;
157         do {
158             oldNode = selectNode();
159         } while (newNode.hasEdgeTo(oldNode));
160
161         // We create the link
162         createEdgeBetween(newNode, oldNode);
163     }
164
165     // We add the new node to the list
166     agentList.add(newNode);
167 }
168
169 /**
170  * We print out the degree distribution
171  */
172 public void postStep() {
173
174     System.out.print(getTickCount() + ": ");
175     Iterator i = agentList.iterator();
176     while (i.hasNext()) {
177         DefaultNode n = (DefaultNode) i.next();
178         System.out.print(n.getInDegree() + n.getOutDegree() + " ");
179     }
180     System.out.println();
181 }
182 }
183
184 public static void main(String[] args) {
185     SimInit init = new SimInit();
186     Model m = new Model();
187     init.loadModel(m, null, false);
188 }
189 }

```

```
1 package ch.ethz.icr.prefattach;
2
3 import uchicago.src.sim.analysis.Plot;
4 import uchicago.src.sim.analysis.NetSequenceGraph;
5 import uchicago.src.sim.engine.Controller;
6 import uchicago.src.sim.engine.SimInit;
7 import uchicago.src.sim.gui.*;
8 import uchicago.src.sim.network.DefaultNode;
9 import uchicago.src.reflector.ListPropertyDescriptor;
10
11 import java.util.Iterator;
12 import java.util.Hashtable;
13
14 /**
15  * Preferential Attachment model
16  *
17  * @author Luc Girardin
18  * @author Lars-Erik Cederman
19  * @author Laszlo Gulyas
20  * @version 2.0
21  */
22 public class ModelGUI extends Model {
23     private static final int LAYOUT_RANDOM = 0;
24     private static final int LAYOUT_CIRCULAR = 1;
25     private static final int LAYOUT_FRUCHT = 2;
26     private static final int LAYOUT_KAMADA = 3;
27
28     boolean showPowerPlot;
29     private Plot powerPlot;
30     boolean showGraph;
31     private DisplaySurface dsurf;
32     private GraphLayout graphLayout;
33     private boolean showNetSequenceGraph;
34     private NetSequenceGraph netSequenceGraph;
35     private int layout;
36
37     public ModelGUI() {
38
39         Controller.ALPHA_ORDER = false;
40         Controller.CONSOLE_ERR = false;
41         Controller.CONSOLE_OUT = false;
42
43         showPowerPlot = true;
44         showGraph = true;
45         showNetSequenceGraph = false;
46     }
47
48     public void setup() {
49
50         // Initializing the original model first
51         super.setup();
52
53         // Specify the parameters to be displayed for setting by the user
54         params = new String[]{"initNodes", "m", "showPowerPlot", "showGraph", "layout",
55                               "showNetSequenceGraph", "pRemoving", "removingStarts"};
56
57         layout = LAYOUT_FRUCHT;
58
59         Hashtable layoutHashtable = new Hashtable();
60         layoutHashtable.put(new Integer(LAYOUT_CIRCULAR), "Circular");
61         layoutHashtable.put(new Integer(LAYOUT_RANDOM), "Random");
62         layoutHashtable.put(new Integer(LAYOUT_FRUCHT), "Fruchterman");
63         layoutHashtable.put(new Integer(LAYOUT_KAMADA), "Kamada");
```

```

64     ListPropertyDescriptor pdLayout = new ListPropertyDescriptor("Layout", layoutHashtab
le);
65     descriptors.put("Layout", pdLayout);
66
67     // Initializing the GUI-part
68     if (powerPlot != null)
69         powerPlot.dispose();
70
71     if (netSequenceGraph != null) {
72         netSequenceGraph.dispose();
73     }
74
75     if (dsurf != null)
76         dsurf.dispose();
77
78     dsurf = new DisplaySurface(this, "Network Display");
79     registerDisplaySurface("Main", dsurf);
80     System.gc();
81 }
82
83 public void buildModel() {
84     // Build the original model first
85     super.buildModel();
86
87     if (showPowerPlot) {
88         powerPlot = new Plot("Degree Distribution");
89         powerPlot.display();
90     }
91
92     if (showNetSequenceGraph) {
93         netSequenceGraph = new NetSequenceGraph("Chart", this, agentList);
94         netSequenceGraph.setYRange(0, 5);
95         netSequenceGraph.display();
96     }
97
98     // We create the network display
99     if (showGraph) {
100         switch (layout) {
101             case LAYOUT_RANDOM:
102                 graphLayout = new RandomGraphLayout(agentList, 500, 500);
103                 break;
104             case LAYOUT_CIRCULAR:
105                 graphLayout = new CircularGraphLayout(agentList, 500, 500);
106                 break;
107             case LAYOUT_FRUCHT:
108                 graphLayout = new FruchGraphLayout(agentList, 500, 500, dsurf, 0);
109                 break;
110             case LAYOUT_KAMADA:
111                 graphLayout = new KamadaGraphLayout(agentList, 500, 500, dsurf, 0);
112                 break;
113         }
114         Network2DDisplay display = new Network2DDisplay(graphLayout);
115
116         // We add and display the new display.
117         dsurf.addDisplayableProbeable(display, "Network Display");
118         addSimEventListener(dsurf);
119         dsurf.display();
120     }
121 }
122
123 /**
124  * This method calculates the ratio of agents having a
125  * degree value equal to 'size'.

```

```

126     */
127     protected double calcProb(int size) {
128         int n = 0;
129         Iterator i = agentList.iterator();
130         while (i.hasNext()) {
131             DefaultNode node = (DefaultNode) i.next();
132             if (node.getInDegree() + node.getOutDegree() >= size) {
133                 n++;
134             }
135         }
136         return (double) (n + 1) / (double) agentList.size();
137     }
138
139     /**
140      * This method updates the 'log-log graph' of degree distributions.
141      */
142     protected void plotPoints() {
143         int linePoints = 0;
144         double x1 = 1000.0;
145         double x2 = 0.0;
146         double a, b, sx, sy, sxx, sxy;
147         sx = 0.0;
148         sy = 0.0;
149         sxx = 0.0;
150         sxy = 0.0;
151
152         powerPlot.clear(0);
153         powerPlot.clear(1);
154         powerPlot.display();
155         powerPlot.setConnected(false, 0);
156
157         Iterator i = agentList.iterator();
158         while (i.hasNext()) {
159             DefaultNode n = (DefaultNode) i.next();
160             if (n.getInDegree() + n.getOutDegree() > 0) {
161                 double x = Math.log((double) n.getInDegree() + n.getOutDegree()) /
162                     Math.log(10.0);
163                 double y = Math.log(calcProb(n.getInDegree() + n.getOutDegree())) /
164                     Math.log(10.0);
165                 powerPlot.plotPoint(x, y, 0);
166                 linePoints++;
167                 sx = sx + x;
168                 sy = sy + y;
169                 sxx = sxx + x * x;
170                 sxy = sxy + x * y;
171
172                 if (x < x1)
173                     x1 = x;
174                 if (x > x2)
175                     x2 = x;
176             }
177         }
178
179         b = (linePoints * sxy - sx * sy) / (linePoints * sxx - sx * sx);
180         a = (sy - b * sx) / linePoints;
181         powerPlot.setConnected(true, 1);
182         powerPlot.plotPoint(x1, a + b * x1, 1);
183         powerPlot.plotPoint(x2, a + b * x2, 1);
184
185         powerPlot.updateGraph();
186         powerPlot.fillPlot();
187     }
188

```

```
189     public void step() {
190         // Do the original model's step() method first
191         super.step();
192
193         // Do the GUI-part of it by updating the graph and the display
194         if (showPowerPlot)
195             plotPoints();
196
197         if(showNetSequenceGraph)
198             netSequenceGraph.step();
199
200         if (showGraph) {
201             graphLayout.setList(agentList);
202             graphLayout.updateLayout();
203             dsurf.updateDisplay();
204         }
205     }
206
207     /**
208      * Suppresses postStep() of the Model
209      */
210     public void postStep() {
211     }
212
213     public int getInitNodes() {
214         return initNodes;
215     }
216
217     public void setInitNodes(int n) {
218         initNodes = n;
219     }
220
221     public int getM() {
222         return m;
223     }
224
225     public void setM(int m) {
226         this.m = m;
227     }
228
229     public double getPRemoving() {
230         return pRemoving;
231     }
232
233     public void setPRemoving(double d) {
234         pRemoving = d;
235     }
236
237     public int getRemovingStarts() {
238         return removingStarts;
239     }
240
241     public void setRemovingStarts(int a) {
242         removingStarts = a;
243     }
244
245     public boolean getShowPowerPlot() {
246         return showPowerPlot;
247     }
248
249     public void setShowPowerPlot(boolean show) {
250         showPowerPlot = show;
251     }
```

```
252
253     public boolean getShowGraph() {
254         return showGraph;
255     }
256
257     public void setShowGraph(boolean show) {
258         showGraph = show;
259     }
260
261     public boolean isShowNetSequenceGraph() {
262         return showNetSequenceGraph;
263     }
264
265     public void setShowNetSequenceGraph(boolean showNetSequenceGraph) {
266         this.showNetSequenceGraph = showNetSequenceGraph;
267     }
268
269     public int getLayout() {
270         return layout;
271     }
272
273     public void setLayout(int layout) {
274         this.layout = layout;
275     }
276
277     public static void main(String[] args) {
278         SimInit init = new SimInit();
279         Model m = new ModelGUI();
280         init.loadModel(m, null, false);
281     }
282 }
```