```
package netsci;

import edu.uci.ics.jung.graph.*;
import edu.uci.ics.jung.graph.impl.*;
import java.util.*;
//import edu.uci.ics.jung.io.*;
//import java.io.*;

/**
 * Completes an RDS tree by using degree information from the
 * underlying graph from which the RDS tree was made
 *
 * @version     $$
 * @author      Bilal Khan
 */
public class CompletionBiased extends Completion {

    private static int _phase = 1;

    static class CompletionBiasedIterator extends Completion.CompletionIterator {

        CompletionBiasedIterator (RDS rds, Graph g) {
            super(rds, g);
        }

        public Object next() {
            return new CompletionBiased(_rds, _g);
        }
    }

    public static Iterator iterator(RDS rds, Graph g) {
        return new CompletionBiasedIterator(rds, g);
    }

    protected CompletionBiased(RDS rds, Graph g) {
        super(rds, g);
    }

    protected void buildCompletion(RDS rds, Graph g) {
        // print the status before completion
        if (DEBUG) System.out.println(""+this.toString());

        // phase 1
        _phase=1;

        buildAllOptions();
        boolean done = false;
        while (!done) {
            done = chooseOneOption() || (_options.size() == 0);
            buildAllOptions();
        }

        // phase 2
        _phase=2;

        buildAllOptions();
        done = false;
        while (!done) {
            done = chooseOneOption() || (_options.size() == 0);
            buildAllOptions();
        }
    }

    protected void buildAllOptions() {
        _options.clear();
        for (Iterator v1it = _tasks.iterator(); v1it.hasNext();) {
            Vertex cv1 = (Vertex)v1it.next();
            for (Iterator v2it = _tasks.iterator(); v2it.hasNext();) {
                Vertex cv2 = (Vertex)v2it.next();
                // check for duplicate edges and self loops
                if (cv1.isPredecessorOf(cv2) || cv2.isPredecessorOf(cv1) || (cv1==cv2)) {
                    continue;
                }
                // check that adding the edge doesn't imply that v1 or
                // v2 did not follow directions at RDS tree build time
                else if ((_phase==1) && (degreeViolation(cv1,cv2))) {
                    continue;
```

```
                }
                // everything is fine... in phase 2 degree violations are ok
                else {
                    // add the option
                    UndirectedSparseEdge e2 = new UndirectedSparseEdge(cv1,cv2);
                    _options.addLast(e2);
                }
            }
        }
        if (DEBUG) System.out.println("Options="+_options.size());
    }

    private boolean degreeViolation(Vertex cv1, Vertex cv2) {
        Vertex rv1 = getRDSVertex(cv1);
        Vertex gv1 = _rds.getGraphVertex(rv1);
        int d1target = gv1.inDegree();
        int time1 = ((Integer)_rds._gv2time.get(gv1)).intValue();

        Vertex rv2 = getRDSVertex(cv2);
        Vertex gv2 = _rds.getGraphVertex(rv2);
        int d2target = gv2.inDegree();
        int time2 = ((Integer)_rds._gv2time.get(gv2)).intValue();

        if (_rds._gv2maxdeg.get(gv1)==null) {
            System.out.println("maxdeg gv1 "+gv1+" is null");
        }
        if (_rds._gv2maxdeg.get(gv2)==null) {
            System.out.println("maxdeg gv2 "+gv2+" is null");
        }

        int maxdeg1 = ((Integer)_rds._gv2maxdeg.get(gv1)).intValue();
        int maxdeg2 = ((Integer)_rds._gv2maxdeg.get(gv2)).intValue();

        // v1 was undiscovered when v2 was discovered and v1's G
        // degree > the smallest G-degree of the RDS referrals that v2
        // made.  Ergo, v2 lied and did not give its coupons to the
        // nodes with highest degree.  This makes the proposed edge
        // illegal since we are presuming all nodes cooperated with
        // the instructions.
        if ((d1target > maxdeg2) && (time1 > time2)) {
            //System.out.print("Violation: ");
            //if (d1target > maxdeg2) System.out.print("d1target("+d1target+") > maxdeg2("+maxdeg2+")");
            //if (time1 > time2) System.out.print("time1("+time1+") > time2("+time2+")");
            //System.out.println("");
            return true;
        }

        // v2 was undiscovered when v1 was discovered and v2's G
        // degree > the smallest G-degree of the RDS referrals that v1
        // made.  Ergo, v1 lied and did not give its coupons to the
        // nodes with highest degree.  This makes the proposed edge
        // illegal since we are presuming all nodes cooperated with
        // the instructions.
        if ((d2target > maxdeg1) && (time2 > time1)) {
            //System.out.print("Violation: ");
            //if (d2target > maxdeg1) System.out.print("d2target("+d2target+") > maxdeg1("+maxdeg1+")");
            //if (time2 > time1) System.out.print("time2("+time2+") > time1("+time1+")");
            //System.out.println("");
            return true;
        }

        // everything is fine
        return false;
    }

    public String toString() {
        int v = _completion.numVertices();
        int e = _completion.numEdges();
        String s = "CompletionBiased vertices="+v+",  edges="+e+", deficiency="+_tasks.size()+"";
        /*
        s+="deficient nodes = [";
        for (Iterator it2=_tasks.iterator();it2.hasNext();) {
            Vertex fv=(Vertex)it2.next();
            s+=(""+fv+",");
        }
        s+=("]");
        */
```

```
        return s;
    }
};
```

```
package netsci;

import edu.uci.ics.jung.graph.*;
import edu.uci.ics.jung.graph.impl.*;
import java.util.*;
//import edu.uci.ics.jung.io.*;
//import java.io.*;

/**
 * Completes an RDS tree by using degree information from the
 * underlying graph from which the RDS tree was made
 *
 * @version     $$
 * @author      Bilal Khan
 */
public class Completion {

    // debugging
    protected static final boolean DEBUG = false;

    // probability of connecting outside of the discovered graph
    protected static final double CLOSURE_PROB = 1.0;
    // maximum number of attempts to find an addable edge
    protected static final int MAXTRIES = 10;

    // the graph
    final Graph _g;

    // the rds
    final RDS _rds;

    // the completion of RDS;
    final Graph _completion;

    // deficiencies in vertex degree
    protected final LinkedList _tasks = new LinkedList();

    // final options for edges
    protected final LinkedList _options = new LinkedList();

    // tables to convert from RDS vertices to completion vertices and back
    protected final Map _lutr2c = new HashMap();
    protected final Map _lutc2r = new HashMap();

    // iterator over completions
    static class CompletionIterator implements Iterator {
        protected RDS _rds;
        protected Graph _g;

        CompletionIterator (RDS rds, Graph g) {
            _rds = rds;
            _g = g;
        }

        public boolean hasNext() {
            return true;
        }

        public Object next() {
            return new Completion(_rds, _g);
        }

        public void remove() {
            // no-op
        }
    }

    // static factory over RDS completions
    public static Iterator iterator(RDS rds, Graph g) {
        return new CompletionIterator(rds, g);
    }

    protected Completion(RDS rds, Graph g) {

        // save args
        _rds = rds;
        _g = g;
```

```
        // the completion graph is a new graph
        _completion = new UndirectedSparseGraph();

        // Add the vertices of the RDS to the completion graph
        Set vset = rds._rds.getVertices();
        for (Iterator it=vset.iterator();it.hasNext();) {
            Vertex rv = (Vertex)it.next();
            Vertex gv = rds.getGraphVertex(rv);
            Vertex cv = new SimpleUndirectedSparseVertex();
            _completion.addVertex(cv);

            // save a bidirectional mapping from RDS to completion vertices
            _lutr2c.put(rv, cv);
            _lutc2r.put(cv, rv);

            // put the edge deficiencies in the task list
            int rdeg = rv.inDegree();
            int gdeg = gv.inDegree();
            // if (rds._seeds.contains(rv)) System.out.print("* ");
            // System.out.println("rdeg="+rdeg+", gdeg="+gdeg);
            if (rdeg < gdeg) {
                for (int n=0; n<(gdeg-rdeg); n++) {
                    _tasks.addLast(cv);
                }
            }
        }

        // Add the edges of the RDS to the completion graph
        Set eset = rds._rds.getEdges();
        for (Iterator it=eset.iterator();it.hasNext();) {
            Edge e = (Edge)it.next();
            Vertex ru = (Vertex)e.getEndpoints().getFirst();
            Vertex cu = getCompletionVertex(ru);
            Vertex rv = (Vertex)e.getEndpoints().getSecond();
            Vertex cv = getCompletionVertex(rv);
            UndirectedSparseEdge e2 = new UndirectedSparseEdge(cu,cv);
            _completion.addEdge(e2);
        }

        // complete the graph
        buildCompletion(rds, g);
    }

    protected void buildCompletion(RDS rds, Graph g) {
        // print the status before completion
        if (DEBUG) System.out.println(""+this.toString());

        int tries = 0;
        while ((_tasks.size() > 1) && (tries <= MAXTRIES)) {
            boolean found = false;
            Vertex v1,v2;

            tries = 0;
            do {
                int n = _tasks.size();
                int r1 = (int)(Math.random() * n);
                v1 = (Vertex)_tasks.remove(r1);
                n = _tasks.size();
                int r2 = (int)(Math.random() * n);
                v2 = (Vertex)_tasks.remove(r2);

                if (DEBUG) System.out.println("contemplating edge ("+v1+", "+v2+")");

                // no duplicate edges or self loops
                if (v1.isPredecessorOf(v2) || v2.isPredecessorOf(v1) || (v1==v2)) {
                    _tasks.addLast(v1);
                    _tasks.addLast(v2);
                }
                else {
                    found = true;
                }

                tries++;
                if (tries > MAXTRIES) {
                    // we give up
                    found = true;
```

```
                }
            }
            while (!found);

            if (tries > MAXTRIES) {
                // we gave up
                if (DEBUG) System.out.println("Unable to add any more edges after "+MAXTRIES+" attempts");
            }
            else {
                if (DEBUG) System.out.println("adding ("+v1+", "+v2+")");
                UndirectedSparseEdge e = new UndirectedSparseEdge(v1,v2);
                _completion.addEdge(e);
            }
        }

        // in the second pass we do exhaustive search

        buildAllOptions();
        boolean done = false;
        while (!done) {
            done = chooseOneOption() || (_options.size() == 0);
            buildAllOptions();
        }
    }

    protected void buildAllOptions() {
        _options.clear();
        for (Iterator v1it = _tasks.iterator(); v1it.hasNext();) {
            Vertex cv1 = (Vertex)v1it.next();
            for (Iterator v2it = _tasks.iterator(); v2it.hasNext();) {
                Vertex cv2 = (Vertex)v2it.next();
                // duplicate edges and self loops are not an option
                if (cv1.isPredecessorOf(cv2) || cv2.isPredecessorOf(cv1) || (cv1==cv2)) {
                    continue;
                }
                else {
                    // add the option
                    UndirectedSparseEdge e2 = new UndirectedSparseEdge(cv1,cv2);
                    _options.addLast(e2);
                }
            }
        }
        if (DEBUG) System.out.println("Options="+_options.size());
    }

    protected boolean chooseOneOption() {
        if (_options.size() > 0) {
            int n = _options.size();
            int r = (int)(Math.random() * n);
            UndirectedSparseEdge e = (UndirectedSparseEdge)_options.get(r);
            _completion.addEdge(e);
            _tasks.remove(e.getEndpoints().getFirst());
            _tasks.remove(e.getEndpoints().getSecond());
            return false;
        }
        else return true;
    }

    Vertex getCompletionVertex(Vertex rv) {
        return (Vertex)_lutr2c.get(rv);
    }

    protected Vertex getRDSVertex(Vertex cv) {
        return (Vertex)_lutc2r.get(cv);
    }

    static Vertex getRandomRDSVertex(RDS rds) {
        Set vset = rds._rds.getVertices();
        Object[] varray = vset.toArray();
        int r = (int)(Math.random() * varray.length);
        Vertex v = (Vertex)varray[r];
        return v;
    }

    public String toString() {
        int v = _completion.numVertices();
        int e = _completion.numEdges();
```

```
        String s = "Completion vertices="+v+",  edges="+e+", deficiency="+_tasks.size()+"";
        /*
        s+="deficient nodes = [";
        for (Iterator it2=_tasks.iterator();it2.hasNext();) {
            Vertex fv=(Vertex)it2.next();
            s+=(""+fv+",");
        }
        s+=("]");
        */
        return s;
    }
};
```

```
package netsci;

import edu.uci.ics.jung.graph.*;
import edu.uci.ics.jung.graph.impl.*;
import edu.uci.ics.jung.algorithms.importance.*;
import edu.uci.ics.jung.algorithms.cluster.*;
import edu.uci.ics.jung.graph.decorators.*;
import edu.uci.ics.jung.algorithms.metrics.*;
import java.util.*;
//import edu.uci.ics.jung.io.*;
//import java.io.*;

/**
 * A class representing measuring aggregate constraint
 *
 * @version     $$
 * @author      Bilal Khan
 */
public class MeasurableAggregateConstraint implements Measurable {

    private final StructuralHoles _sh;
    private final EdgeWeightLabeller _ewl;

    MeasurableAggregateConstraint(Graph g) {
        _ewl = EdgeWeightLabeller.getLabeller(g);
        for (Iterator eit=g.getEdges().iterator(); eit.hasNext(); ) {
            Edge e = (Edge)eit.next();
            _ewl.setWeight(e,1);
        }
        _sh = new StructuralHoles(_ewl);
    }

    public double readValue(Vertex vquery) {
        double value = _sh.hierarchy(vquery);
        return value;
    }
};
```

```
package netsci;

import edu.uci.ics.jung.graph.*;
import edu.uci.ics.jung.graph.impl.*;
import edu.uci.ics.jung.algorithms.importance.*;
import edu.uci.ics.jung.algorithms.cluster.*;
import java.util.*;
//import edu.uci.ics.jung.io.*;
//import java.io.*;

/**
 * A class representing measuring authority
 *
 * @version     $$
 * @author      Bilal Khan
 */
public class MeasurableAuthority implements Measurable {

    private final HITS _ranker;
    private final List _rlist;

    MeasurableAuthority(Graph g) {
        HITS ranker = new HITS(g, true); // use authority for ranking
        _ranker = new HITS(g);
        _ranker.setRemoveRankScoresOnFinalize(false);
        _ranker.evaluate();
        _rlist = _ranker.getRankings();
    }

    public double readValue(Vertex vquery) {
        double value = -1.0;
        for (Iterator rlit=_rlist.iterator(); rlit.hasNext(); ) {
            NodeRanking nr = (NodeRanking)rlit.next();
            Vertex rv = nr.vertex;
            if (rv==vquery) {
                value = nr.rankScore;
                break;
            }
        }
        return value;
    }
};
```

```
package netsci;

import edu.uci.ics.jung.graph.*;
import edu.uci.ics.jung.graph.impl.*;
import edu.uci.ics.jung.algorithms.importance.*;
import edu.uci.ics.jung.algorithms.cluster.*;
import java.util.*;
//import edu.uci.ics.jung.io.*;
//import java.io.*;

/**
 * A class representing measuring betweenness centrality
 *
 * @version     $$
 * @author      Bilal Khan
 */
public class MeasurableBC implements Measurable {

    private final BetweennessCentrality _ranker;
    private final List _rlist;

    private final WeakComponentClusterer _clusterer = new WeakComponentClusterer();
    private final ClusterSet _clusters;

    MeasurableBC(Graph g) {

        _ranker = new BetweennessCentrality(g, true, false);
        _ranker.setRemoveRankScoresOnFinalize(false);
        _ranker.evaluate();
        _rlist = _ranker.getRankings();

        _clusters = _clusterer.extract(g);
        // System.out.println("Number of components "+_clusters.size());
    }

    public double readValue(Vertex vquery) {
        double value = -1.0;
        int i=0;
        boolean done = false;
        for (Iterator it=_clusters.iterator(); it.hasNext() && !done; ) {
            // System.out.println("Component "+i);
            HashSet onecomp = (HashSet)it.next();
            if (!onecomp.contains(vquery)) continue;
            // else we found the component which has the vertex of interest

            for (Iterator it2=onecomp.iterator(); it2.hasNext() && !done; ) {
                Vertex vx = (Vertex)it2.next();
                if (vx!=vquery) continue;
                // else we found the vertex of interest

                // calculate the relative betweenness rank of vquery
                int rankcalc = 0;
                double score = -1.0;
                for (Iterator rlit=_rlist.iterator(); rlit.hasNext(); ) {
                    NodeRanking nr = (NodeRanking)rlit.next();
                    Vertex rv = nr.vertex;
                    score = nr.rankScore;
                    if (onecomp.contains(rv)) {
                        rankcalc++;
                    }
                    if (rv==vx) break;
                }

                value = (double)rankcalc/(double)onecomp.size();
                done = true;
                // System.out.println("Vertex:"+vx+" relrank:"+rankcalc+"/"+onecomp.size()+" absrank="+score)
;
            }
            i++;
        }
        return value;
    }
};
```

```
package netsci;

import edu.uci.ics.jung.graph.*;
import edu.uci.ics.jung.graph.impl.*;
import edu.uci.ics.jung.algorithms.importance.*;
import edu.uci.ics.jung.algorithms.cluster.*;
import java.util.*;
//import edu.uci.ics.jung.io.*;
//import java.io.*;

/**
 * A class representing measuring betweenness centrality
 *
 * @version     $$
 * @author      Bilal Khan
 */
public class MeasurableBCRaw implements Measurable {

    private final BetweennessCentrality _ranker;
    private final List _rlist;
    MeasurableBCRaw(Graph g) {

        _ranker = new BetweennessCentrality(g, true, false);
        _ranker.setRemoveRankScoresOnFinalize(false);
        _ranker.evaluate();
        _rlist = _ranker.getRankings();
    }

    public double readValue(Vertex vquery) {
        double value = -1.0;
        for (Iterator rlit=_rlist.iterator(); rlit.hasNext(); ) {
            NodeRanking nr = (NodeRanking)rlit.next();
            Vertex rv = nr.vertex;
            if (rv==vquery) {
                value = nr.rankScore;
                break;
            }
        }
        return value;
    }
};
```

```
package netsci;

import edu.uci.ics.jung.graph.*;
import edu.uci.ics.jung.graph.impl.*;
import edu.uci.ics.jung.algorithms.importance.*;
import edu.uci.ics.jung.algorithms.cluster.*;
import edu.uci.ics.jung.graph.decorators.*;
import edu.uci.ics.jung.algorithms.metrics.*;
import java.util.*;
//import edu.uci.ics.jung.io.*;
//import java.io.*;

/**
 * A class representing measuring pagerank
 *
 * @version     $$
 * @author      Bilal Khan
 */
public class MeasurableConstraint implements Measurable {

    private final StructuralHoles _sh;
    private final EdgeWeightLabeller _ewl;

    MeasurableConstraint(Graph g) {
        _ewl = EdgeWeightLabeller.getLabeller(g);
        for (Iterator eit=g.getEdges().iterator(); eit.hasNext(); ) {
            Edge e = (Edge)eit.next();
            _ewl.setWeight(e,1);
        }
        _sh = new StructuralHoles(_ewl);
    }

    public double readValue(Vertex vquery) {
        double value = _sh.constraint(vquery);
        return value;
    }
};
```

```
package netsci;

import edu.uci.ics.jung.graph.*;
import edu.uci.ics.jung.graph.impl.*;
import edu.uci.ics.jung.algorithms.importance.*;
import edu.uci.ics.jung.algorithms.cluster.*;
import edu.uci.ics.jung.graph.decorators.*;
import edu.uci.ics.jung.algorithms.metrics.*;
import java.util.*;
//import edu.uci.ics.jung.io.*;
//import java.io.*;

/**
 * A class representing measuring pagerank
 *
 * @version     $$
 * @author      Bilal Khan
 */
public class MeasurableEffectiveSize implements Measurable {

    private final StructuralHoles _sh;
    private final EdgeWeightLabeller _ewl;

    MeasurableEffectiveSize(Graph g) {
        _ewl = EdgeWeightLabeller.getLabeller(g);
        for (Iterator eit=g.getEdges().iterator(); eit.hasNext(); ) {
            Edge e = (Edge)eit.next();
            _ewl.setWeight(e,1);
        }
        _sh = new StructuralHoles(_ewl);
    }

    public double readValue(Vertex vquery) {
        double value = _sh.effectiveSize(vquery);
        return value;
    }
};
```

```
package netsci;

import edu.uci.ics.jung.graph.*;
import edu.uci.ics.jung.graph.impl.*;
import edu.uci.ics.jung.algorithms.importance.*;
import edu.uci.ics.jung.algorithms.cluster.*;
import edu.uci.ics.jung.graph.decorators.*;
import edu.uci.ics.jung.algorithms.metrics.*;
import java.util.*;
//import edu.uci.ics.jung.io.*;
//import java.io.*;

/**
 * A class representing measuring pagerank
 *
 * @version     $$
 * @author      Bilal Khan
 */
public class MeasurableHierarchy implements Measurable {

    private final StructuralHoles _sh;
    private final EdgeWeightLabeller _ewl;

    MeasurableHierarchy(Graph g) {
        _ewl = EdgeWeightLabeller.getLabeller(g);
        for (Iterator eit=g.getEdges().iterator(); eit.hasNext(); ) {
            Edge e = (Edge)eit.next();
            _ewl.setWeight(e,1);
        }
        _sh = new StructuralHoles(_ewl);
    }

    public double readValue(Vertex vquery) {
        double value = _sh.hierarchy(vquery);
        return value;
    }
};
```

```
package netsci;

import edu.uci.ics.jung.graph.*;
import edu.uci.ics.jung.graph.impl.*;
import edu.uci.ics.jung.algorithms.importance.*;
import edu.uci.ics.jung.algorithms.cluster.*;
import java.util.*;
//import edu.uci.ics.jung.io.*;
//import java.io.*;

/**
 * A class representing measuring hubness
 *
 * @version     $$
 * @author      Bilal Khan
 */
public class MeasurableHubness implements Measurable {

    private final HITS _ranker;
    private final List _rlist;

    MeasurableHubness(Graph g) {
        HITS ranker = new HITS(g, false); // don't use authority for ranking
        _ranker = new HITS(g);
        _ranker.setRemoveRankScoresOnFinalize(false);
        _ranker.evaluate();
        _rlist = _ranker.getRankings();
    }

    public double readValue(Vertex vquery) {
        double value = -1.0;
        for (Iterator rlit=_rlist.iterator(); rlit.hasNext(); ) {
            NodeRanking nr = (NodeRanking)rlit.next();
            Vertex rv = nr.vertex;
            if (rv==vquery) {
                value = nr.rankScore;
                break;
            }
        }
        return value;
    }
};
```

```
package netsci;

import edu.uci.ics.jung.graph.*;
import edu.uci.ics.jung.graph.impl.*;
import java.util.*;
//import edu.uci.ics.jung.io.*;
//import java.io.*;

/**
 * An interface representing a measurable quantity
 *
 * @version     $$
 * @author      Bilal Khan
 */
public interface Measurable {
    public double readValue(Vertex vquery);
};
```

```
package netsci;

import edu.uci.ics.jung.graph.*;
import edu.uci.ics.jung.graph.impl.*;
import java.util.*;
//import edu.uci.ics.jung.io.*;
//import java.io.*;

/**
 * Build an RDS tree on a graph starting from a randomly chosen set of
 * seeds.  The RDS tree is biased in the following way: a person
 * always gives out their coupons to (previously undiscovered)
 * individuals with the highest degree.
 *
 * @version     $$
 * @author      Bilal Khan
 */
public class RDSBiased extends RDS {

    static class RDSBiasedIterator extends RDS.RDSIterator {

        RDSBiasedIterator (Graph g, int seeds) {
            super(g, seeds);
        }

        public Object next() {
            return new RDSBiased(_g, _seeds);
        }

        public void remove() {
            // no-op
        }
    }

    public static Iterator iterator(Graph g, int seeds) {
        return new RDSBiasedIterator(g, seeds);
    }

    private RDSBiased(Graph g, int seeds) {
        super(g, seeds);
    }

    // overrides the updateMaxDeg in base RDS class
    protected void updateMaxDeg(Graph g, Vertex gv) {

        // GOAL: _gv2maxdeg(gv) should equal to the minimum G-degree
        // over all RDS-nighbors of gv which were discovered after gv
        // (i.e. referrals by gv)

        // get all the G-neighbors of gv
        Set e1 = gv.getInEdges();
        Set e2 = gv.getOutEdges();
        Set eall = new HashSet();
        eall.addAll(e1);
        eall.addAll(e2);
        Object earray[] = eall.toArray();

        // time gv was discovered
        int gvtime = ((Integer)_gv2time.get(gv)).intValue();

        // we are doing a minimization
        int maxdeg = 999;

        // iterate over all G-neighbors of gv
        for (int i=0;i<earray.length; i++) {
            Edge e = (Edge)earray[i];
            Vertex gu = e.getOpposite(gv);
            Vertex ru = getRDSVertex(gu);
            // but consider only RDS-neighbors of gv
            if (ru==null) continue;

            // System.out.println("Considering "+gv+"-->"+gu+"  time="+_gv2time.get(gu));

            // only consider children of gv (not the parent)
            int gutime = ((Integer)_gv2time.get(gu)).intValue();
            if (gutime < gvtime) continue;
```

```
            // get the G-degree of the RDS-neighbor
            int degu = gu.getInEdges().size() + gu.getOutEdges().size();

            // keep track of the minimum G-degree seen
            if (degu < maxdeg) {
                maxdeg = degu;
            }
        }

        // System.out.println("Maxdeg "+gv+"-->"+maxdeg);
        // save the value
        _gv2maxdeg.put(gv, new Integer(maxdeg));
    }

    // overrides the getRandomNeighborGraphVertex in base RDS class
    protected Vertex getRandomNeighborGraphVertex(Graph g, Vertex gv) {
        Set e1 = gv.getInEdges();
        Set e2 = gv.getOutEdges();
        Set eall = new HashSet();
        eall.addAll(e1);
        eall.addAll(e2);
        Object earray[] = eall.toArray();

        // find the neighbor of gv that is both undiscovered and has
        // the highest degree
        int maxdeg = -1;
        Vertex answer = null;
        for (int i=0;i<earray.length; i++) {
            Edge e = (Edge)earray[i];
            Vertex gu = e.getOpposite(gv);
            Vertex ru = getRDSVertex(gu);
            // must be undiscovered
            if (ru!=null) continue;

            int degu = gu.getInEdges().size() + gu.getOutEdges().size();
            if (degu > maxdeg) {
                // save the one with the highest degree
                maxdeg = degu;
                answer = gu;
            }
        }
        if (answer==null) {
            System.out.println("ERROR answer=null for vertex "+gv);
            System.out.println("ERROR isDeadGraphVertex(g, gv) = "+isDeadGraphVertex(g, gv));
            System.exit(0);
        }

        return answer;
    }

    public String toString() {
        int v = _rds.numVertices();
        int e = _rds.numEdges();
        String s = "RDSBiased vertices="+v+",  edges="+e;
        return s;
    }
};
```

```
package netsci;

import edu.uci.ics.jung.graph.*;
import edu.uci.ics.jung.graph.impl.*;
import java.util.*;
//import edu.uci.ics.jung.io.*;
//import java.io.*;

/**
 * Build an RDS tree on a graph starting from a randomly chosen set of seeds.
 *
 * @version     $$
 * @author      Bilal Khan
 */
public class RDS {

    // debugging
    protected static final boolean DEBUG = false;

    // RDS degree
    protected static final int REFERRALS = 3;

    // the RDS graph
    final Graph _rds;

    // A map from graph vertex to time of discovery
    HashMap _gv2time = new HashMap();

    // The current time
    int _time = 0;

    // A map from graph vertex to maximum permissable degree of
    // neighbors in the completion (excluding RDS neighbors)
    HashMap _gv2maxdeg = new HashMap();

    // initial seeds
    final Set _seeds = new HashSet();

    // vertices of G that the RDS has discovered
    protected final Set _discovered = new HashSet();

    // the frontier of RDS
    protected final LinkedList _frontier = new LinkedList();

    // tables to convert from graph to RDS vertices and back
    protected final Map _lutg2r = new HashMap();
    protected final Map _lutr2g = new HashMap();


    // An iterator over RDS trees
    static class RDSIterator implements Iterator {

        protected Graph _g;
        protected int _seeds;

        RDSIterator (Graph g, int seeds) {
            _g = g;
            _seeds = seeds;
        }

        public boolean hasNext() {
            return true;
        }

        public Object next() {
            return new RDS(_g, _seeds);
        }

        public void remove() {
            // no-op
        }
    }

    // static factor method to make an iterator over RDS trees
    public static Iterator iterator(Graph g, int seeds) {
        return new RDSIterator(g, seeds);
    }
```

```
    private boolean _virgin = true;

    protected Vertex getNewSeed(Graph g) {
        Vertex gv, rv;

        if (_virgin) {
            // first seed always lies in the big component
            gv = Test.getBigCompSeed(g);
            _virgin = false;
        }
        else {
            // get a new previously unused seed
            do {
                gv = getRandomGraphVertex(g);
                // see if its already been used as a seed
                rv = getRDSVertex(gv);
            }
            while (rv != null);
        }
        return gv;
    }

    protected void initializeSeeds(Graph g, int seeds) {
        // initialize the seeds
        for (int i=0;i<seeds;i++) {
            // get a new seed
            Vertex gv = getNewSeed(g);

            // make the new RDS vertex
            Vertex rv = new SimpleUndirectedSparseVertex();

            // save a bidirectional mapping between graph and RDS vertices
            _lutg2r.put(gv, rv);
            _lutr2g.put(rv, gv);

            // add the seed to the RDS tree
            _seeds.add(rv);
            _rds.addVertex(rv);
            if (DEBUG) System.out.println("Added seed: "+gv);

            // note the discovery time = 0
            _gv2time.put(gv, new Integer(0));

            // initially the frontier consists of just seeds
            _frontier.addLast(rv);

            // seeds are considered already discovered
            _discovered.add(rv);

            // update maxDegree
            updateMaxDeg(g, gv);
        }
    }

    protected Vertex getNextFrontierVertex(Graph g) {
        Vertex rv = null;
        Vertex gv = null;
        boolean found = false;
        while (_frontier.size() > 0) {
            rv = (Vertex)_frontier.removeFirst();
            gv = getGraphVertex(rv);
            if (DEBUG) System.out.println("Growing from = "+gv);
            if (isDeadGraphVertex(g, gv)) {
                // the frontier vertex is dead, no need to reschedule it...
                if (DEBUG) System.out.println("    All ("+gv.inDegree()+") neighbors of "+gv+" in G have been
 discovered, it's dead");
                continue;
            }
            else { // we found a non-dead frontier vertex!
                if (isSaturatedRDSVertex(rv)) {
                    // if the frontier vertex is rds saturated, continue
                    if (DEBUG) System.out.println("    Done growing from "+rv+" All ("+rv.inDegree()+") RDS-n
eighbors have been selected");
                    continue;
                }
                else {
```

```
                    // not dead, not saturated
                    _frontier.addLast(rv);
                    found = true;
                    break;
                }
            }
        }

        if (found==false) {
            return null;
        }
        else {
            if (isDeadGraphVertex(g,gv)) {
                System.out.println("FRONTIER ERROR Dead vertex "+gv+" has no neighbors!");
                System.exit(0);
            }
            if (isSaturatedRDSVertex(rv)) {
                System.out.println("FRONTIER ERROR Saturated vertex "+rv+" needs no neighbors!");
                System.exit(0);
            }
            return rv;
        }
    }

    protected void printFrontier() {
        System.out.print("Frontier = [");
        for (Iterator it2=_frontier.iterator();it2.hasNext();) {
            Vertex fv=(Vertex)it2.next();
            System.out.print(""+fv+",");
        }
        System.out.println("]");
    }

    protected void grow(Graph g, Vertex rv) {
        // get the graph vertex corresponding to rv
        Vertex gv = getGraphVertex(rv);
        // find a random yet undiscovered neighbor
        Vertex gu = null;
        Vertex ru = null;
        do {
            gu = getRandomNeighborGraphVertex(g,gv);
            ru = getRDSVertex(gu);
            // System.out.println("Considering neighbor = "+ru);
        }
        while ( ru != null );

        // add the discovered vertex
        ru = new SimpleUndirectedSparseVertex();

        // save a bidirectional mapping
        _lutg2r.put(gu, ru);
        _lutr2g.put(ru, gu);

        // add ru to the RDS tree
        _rds.addVertex(ru);

        // note the discovery time
        _time++;
        _gv2time.put(gu, new Integer(_time));
        if (DEBUG) System.out.println("Added vertex: "+gu);

        // augment the frontier and discovered sets
        _discovered.add(ru);
        _frontier.add(ru);

        // add the edge
        UndirectedSparseEdge e = new UndirectedSparseEdge(rv,ru);
        _rds.addEdge(e);

        // update the max degree information
        updateMaxDeg(g, gv);
        updateMaxDeg(g, gu);

        if (DEBUG) System.out.println("Added edge: "+rv+"-->"+ru);
    }

    protected RDS(Graph g, int seeds) {
```

```
        // the time is zero
        _time = 0;

        // the RDS tree is a new graph
        _rds = new UndirectedSparseGraph();

        // add the seeds to the RDS tree, initializing the frontier
        initializeSeeds(g, seeds);

        // Add the vertices of g to the RDS tree
        Vertex rv;
        while ((rv = getNextFrontierVertex(g)) != null) {
            grow(g, rv);
        }
    }

    Vertex getRDSVertex(Vertex gv) {
        return (Vertex)_lutg2r.get(gv);
    }

    Vertex getGraphVertex(Vertex rv) {
        return (Vertex)_lutr2g.get(rv);
    }

    static Vertex getRandomGraphVertex(Graph g) {
        Set vset = g.getVertices();
        Object[] varray = vset.toArray();
        int r = (int)(Math.random() * varray.length);
        Vertex v = (Vertex)varray[r];
        return v;
    }

    protected boolean isSaturatedRDSVertex(Vertex rv) {
        if (_seeds.contains(rv)) {
            if (rv.inDegree()==REFERRALS) return true;
            else return false;
        }
        else  {
            if (rv.inDegree()==REFERRALS+1) return true;
            else return false;
        }
    }

    protected boolean isDeadGraphVertex(Graph g, Vertex gv) {
        Set e1 = gv.getInEdges();
        Set e2 = gv.getOutEdges();
        Set eall = new HashSet();
        eall.addAll(e1);
        eall.addAll(e2);

        Object earray[] = eall.toArray();
        if (DEBUG) System.out.println("Num neighbors="+earray.length);

        for (int i=0; i<earray.length; i++) {
            Edge e = (Edge)earray[i];
            Vertex gu = e.getOpposite(gv);
            Vertex ru = getRDSVertex(gu);

            // we found an undiscovered neighbor,
            // so this vertex isn't dead
            if (ru==null) {
                // System.out.println("Neighbor="+ru);
                return false;
            }
        }
        // all neighbors are already discovered
        // so this vertex is dead
        return true;
    }

    protected void updateMaxDeg(Graph g, Vertex gv) {
        // no op
    }

    protected Vertex getRandomNeighborGraphVertex(Graph g, Vertex gv) {

        if (isDeadGraphVertex(g,gv)) {
```

```
            System.out.println("ERROR Dead vertex has no neighbors!");
            System.exit(0);
        }

        Set e1 = gv.getInEdges();
        Set e2 = gv.getOutEdges();
        Set eall = new HashSet();
        eall.addAll(e1);
        eall.addAll(e2);
        Object earray[] = eall.toArray();
        int r = (int)(Math.random() * earray.length);
        Edge e = (Edge)earray[r];
        Vertex gu = e.getOpposite(gv);
        return gu;
    }

    public String toString() {
        int v = _rds.numVertices();
        int e = _rds.numEdges();
        String s = "RDS vertices="+v+",  edges="+e;
        return s;
    }
};
```

```
package netsci;

import edu.uci.ics.jung.io.*;
import edu.uci.ics.jung.graph.*;
import java.io.*;
import java.util.*;
import java.text.DecimalFormat;

/**
 * A class to collect measurements of the graph's vertices and
 * aggregate statistics about the vertices of the completions
 *
 * @version     $$
 * @author      Bilal Khan
 */
public class Stats {

    // map from G vertices to their measures
    private HashMap _g_rank = new HashMap();
    // map from completion vertices to their accumulated measures
    private HashMap _c_rank_accum = new HashMap();
    // map from completion vertices to number of measurements
    private HashMap _c_rank_ct = new HashMap();

    // initialize the data structures for a given vertex
    public void initialize(Vertex vquery_g, double val1) {
        _g_rank.put(vquery_g, new Double(val1));
        _c_rank_accum.put(vquery_g, new Double(0.0));
        _c_rank_ct.put(vquery_g, new Double(0.0));
    }

    // accumulate the measure of a vertex in the completion
    public void accum(Vertex vquery_g, double val3) {
        _c_rank_accum.put(vquery_g, new Double(((Double)_c_rank_accum.get(vquery_g)).doubleValue()+val3));
        _c_rank_ct.put(vquery_g, new Double(((Double)_c_rank_ct.get(vquery_g)).doubleValue()+1.0));
    }

    // get the measure of a vertex in G
    public double get_grank(Vertex vquery_g) {
        Double grank = (Double)_g_rank.get(vquery_g);
        return grank.doubleValue();
    }

    // get the number of measures taken for a vertex (in the completions)
    public double get_count(Vertex vquery_g) {
        Double ctr = (Double)_c_rank_ct.get(vquery_g);
        return ctr.doubleValue();
    }

    // get the mean measures taken for a vertex (in the completions)
    public double get_crank(Vertex vquery_g) throws Exception {
        double ctr =  get_count(vquery_g);
        if (ctr == 0.0) {
            throw new Exception("No data about this vertex...");
        }
        Double crank = (Double)_c_rank_accum.get(vquery_g);
        return crank.doubleValue()/ctr;
    }


    // If I were to rank the vertices of the completion(s) by their
    // G-rank, and then ask for the cutoff value of G-rank which would
    // separate the top cutoff% from the rest, what would this cutoff
    // value be?
    public double get_grank_threshold(double cutoff, Graph g) {

        double grank_threshold = -1.0;

        TreeMap rank2list = new TreeMap();
        Set vset = g.getVertices();
        int n = 0;

        for (Iterator vit= vset.iterator(); vit.hasNext(); ) {
            Vertex vquery_g = (Vertex)vit.next();
            // if we don't have completion data skip it
            if (get_count(vquery_g)<1.0) {
                continue;
```

```
            }
            n++;

            double sval = get_grank(vquery_g);
            Double score = new Double(sval);
            LinkedList lst = (LinkedList)rank2list.get(score);
            if (lst==null) {
                lst = new LinkedList();
                rank2list.put(score,lst);
                // System.out.println("new list for score="+score);
            }
            lst.addLast(vquery_g);
        }

        int n_cutoff = (int)((double)n*cutoff);

        //System.out.println("n="+n);
        //System.out.println("n_cutoff="+n_cutoff);

        int ct=0;
        for (Iterator vit= rank2list.entrySet().iterator(); vit.hasNext(); ) {
            Map.Entry ent = (Map.Entry)vit.next();
            Double score = (Double)ent.getKey();
            LinkedList lst = (LinkedList)ent.getValue();
            ct += lst.size();

            // System.out.println("ct="+ct+", score="+score);

            if (ct > n_cutoff) {
                grank_threshold = score.doubleValue();
                break;
            }
        }

        return grank_threshold;
    }

    public double get_crank_threshold(double cutoff, Graph g) {

        double crank_threshold = -1.0;

        TreeMap rank2list = new TreeMap();
        Set vset = g.getVertices();
        int n = 0;

        for (Iterator vit= vset.iterator(); vit.hasNext(); ) {
            Vertex vquery_g = (Vertex)vit.next();
            // if we don't have completion data skip it
            if (get_count(vquery_g)<1.0) {
                continue;
            }
            n++;

            double sval;
            try {
                sval = get_crank(vquery_g);
            }
            catch (Exception ex) {
                // no data on this vertex
                continue;
            }
            Double score = new Double(sval);

            LinkedList lst = (LinkedList)rank2list.get(score);
            if (lst==null) {
                lst = new LinkedList();
                rank2list.put(score,lst);
                // System.out.println("new list for score="+score);
            }
            lst.addLast(vquery_g);
        }

        int n_cutoff = (int)((double)n*cutoff);

        //System.out.println("n="+n);
        //System.out.println("n_cutoff="+n_cutoff);
```

```
        int ct=0;
        for (Iterator vit= rank2list.entrySet().iterator(); vit.hasNext(); ) {
            Map.Entry ent = (Map.Entry)vit.next();
            Double score = (Double)ent.getKey();
            LinkedList lst = (LinkedList)ent.getValue();
            ct += lst.size();

            // System.out.println("ct="+ct+", score="+score);

            if (ct > n_cutoff) {
                crank_threshold = score.doubleValue();
                break;
            }
        }

        return crank_threshold;
    }

    public LinkedList filterVertices_min_grank(Graph g, double min_grank) {
        LinkedList lst = new LinkedList();
        Set vset = g.getVertices();
        for (Iterator vit= vset.iterator(); vit.hasNext(); ) {
            Vertex vquery_g = (Vertex)vit.next();
            // if we don't have completion data skip it
            if (get_count(vquery_g)<1.0) {
                continue;
            }
            double sval = get_grank(vquery_g);
            if (sval > min_grank) {
                lst.addLast(vquery_g);
            }
        }
        return lst;
    }

    public LinkedList filterVertices_min_crank(Graph g, double min_crank) {
        LinkedList lst = new LinkedList();
        Set vset = g.getVertices();
        for (Iterator vit= vset.iterator(); vit.hasNext(); ) {
            Vertex vquery_g = (Vertex)vit.next();
            // if we don't have completion data skip it
            if (get_count(vquery_g)<1.0) {
                continue;
            }
            double sval;
            try {
                sval = get_crank(vquery_g);
            }
            catch (Exception ex) {
                // no data on this vertex
                continue;
            }
            if (sval > min_crank) {
                lst.addLast(vquery_g);
            }
        }
        return lst;
    }

    public String error(Graph g, double th) {
        double thg = get_grank_threshold(th, g);
        double thc = get_crank_threshold(th, g);
        LinkedList glist = filterVertices_min_grank(g, thg);
        LinkedList clist = filterVertices_min_crank(g, thc);
        clist.removeAll(glist);
        double err = (double)clist.size()/(double)glist.size();
        String s = (""+Test.twoPlaces.format(th)+
                    "\t"+Test.twoPlaces.format(err)+
                    "\t|\t"+glist.size()+"\t"+
                    Test.twoPlaces.format(thg)+"\t|\t"+
                    clist.size()+"\t"+
                    Test.twoPlaces.format(thc));
        return s;
    }

    public double pearson(Graph g) {
        Set vset = g.getVertices();
```

```
        int N = 0;
        for (Iterator vit= vset.iterator(); vit.hasNext(); ) {
            Vertex vquery_g = (Vertex)vit.next();
            // if we don't have completion data skip it
            if (get_count(vquery_g)<1.0) {
                continue;
            }
            N++;
        }
        double x[] = new double[N];
        double y[] = new double[N];

        int j=0;
        for (Iterator vit= vset.iterator(); vit.hasNext(); ) {
            Vertex vquery_g = (Vertex)vit.next();
            // if we don't have completion data skip it
            if (get_count(vquery_g)<1.0) {
                continue;
            }
            try {
                x[j]=get_grank(vquery_g);
                y[j]=get_crank(vquery_g);
                j++;
            }
            catch (Exception ex) {
                System.out.printf("Invariant violation!");
                System.exit(0);
            }
        }

        double sum_sq_x = 0;
        double sum_sq_y = 0;
        double sum_coproduct = 0;
        double mean_x = x[1];
        double mean_y = y[1];
        for (int i=2; i<N; i++) {
            double sweep = (i - 1.0) / i;
            double delta_x = x[i] - mean_x;
            double delta_y = y[i] - mean_y;
            sum_sq_x += delta_x * delta_x * sweep;
            sum_sq_y += delta_y * delta_y * sweep;
            sum_coproduct += delta_x * delta_y * sweep;
            mean_x += delta_x / i;
            mean_y += delta_y / i;
        }
        double pop_sd_x = Math.sqrt( sum_sq_x / (double)N );
        double pop_sd_y = Math.sqrt( sum_sq_y / (double)N );
        double cov_x_y = sum_coproduct / (double)N;
        double correlation = cov_x_y / (pop_sd_x * pop_sd_y);
        return correlation;
    }
}
```

```
package netsci;

import edu.uci.ics.jung.algorithms.transformation.*;
import edu.uci.ics.jung.io.*;
import edu.uci.ics.jung.graph.*;
import edu.uci.ics.jung.algorithms.cluster.*;
import java.io.*;
import java.util.*;
import java.text.DecimalFormat;

/**
 * Test driver class for the NetSci
 *
 * @version     $$
 * @author      Bilal Khan
 */
public class Test {

    // debugging
    protected static final boolean DEBUG = true;

    // number of seeds
    private static int NUM_SEEDS = 10;

    // number of RDS trees to make
    private static int NUM_RDS_TREES = 10;

    // the number of completions for each RDS tree
    private static int NUM_COMPLETIONS = 10;

    // the number of snapshots
    private static int NUM_SNAPSHOTS = 1;

    // biased RDS and Completions
    private static boolean BIASED = false;

    // the experiment
    private static String EXPERIMENT = "unknown";

    // string formatter
    public static DecimalFormat twoPlaces = new DecimalFormat("0.00000");

    // statistics collector
    private static Stats _stats = new Stats();

    /**
     * Main class
     *
     * @param args -- an array of arguments that are passed in on the command line.
     */

    // static method that defines the graph parameter we want to study
    private static Measurable makeMeasurable(Graph graph) {
        if (EXPERIMENT.equals("H")) return new MeasurableHubness(graph);
        else if (EXPERIMENT.equals("A")) return new MeasurableAuthority(graph);
        else if (EXPERIMENT.equals("BC")) return new MeasurableBCRaw(graph);
        else if (EXPERIMENT.equals("ES")) return new MeasurableEffectiveSize(graph);
        else if (EXPERIMENT.equals("CON")) return new MeasurableConstraint(graph);
        else {
            System.out.println("Unknown experiment: "+EXPERIMENT);
            System.exit(0);
        }
        return null;
    }

    // static method that defines how we build RDS trees
    private static Iterator makeRDSIterator(Graph graph, int seeds) {
        if (BIASED) {
            return RDSBiased.iterator(graph, seeds);
        }
        else {
            return RDS.iterator(graph, seeds);
        }
    }

    // static method that defines how we build completions
    private static Iterator makeCompletionIterator(RDS rds, Graph g) {
```

```
        if (BIASED) {
            return CompletionBiased.iterator(rds, g);
        }
        else {
            return Completion.iterator(rds, g);
        }
    }

    public static void main(String [] args) {

        if (args.length!=6) {
            System.out.println("Usage: <filename> <numSeeds> <numRDStrees> <numCompletions> <biased> <exp>");
            System.out.println("biased: 0/1");
            System.out.println("exp: H, A, BC, ES, CON");
        }

        String fname = args[0];
        String NUM_SEEDSStr = args[1];
        String NUM_RDS_TREESStr = args[2];
        String NUM_COMPLETIONSStr = args[3];
        String BIASEDStr = args[4];

        NUM_SEEDS = Integer.parseInt(NUM_SEEDSStr);
        NUM_RDS_TREES = Integer.parseInt(NUM_RDS_TREESStr);
        NUM_COMPLETIONS = Integer.parseInt(NUM_COMPLETIONSStr);
        int tmpBIASED = Integer.parseInt(BIASEDStr);
        if (tmpBIASED==1) {
            BIASED = true;
        }
        else {
            BIASED = false;
        }
        EXPERIMENT = args[5];

        long stage = 0;
        long maxstage = NUM_RDS_TREES * NUM_COMPLETIONS;
        long blip = maxstage / NUM_SNAPSHOTS;
        if (blip<1) blip=1;

        if (DEBUG) System.out.println("Begin reading: "+fname);
        try {
            PajekNetReader pnr = new PajekNetReader();
            Graph g = pnr.load(fname);
            int v = g.numVertices();
            int e = g.numEdges();
            if (DEBUG) System.out.println("Done reading graph: "+fname);
            if (DEBUG) System.out.println("Graph specs: vertices="+v+", edges="+e);
            // if (DEBUG) debugOutput(g);

            // undirectify the graph
            g = undirectify(g);

            // compute measures for vertices in the graph
            Measurable measG = makeMeasurable(g);
            for (Iterator vit= g.getVertices().iterator(); vit.hasNext(); ) {
                Vertex vquery_g = (Vertex)vit.next();
                double val1 = measG.readValue(vquery_g);
                _stats.initialize(vquery_g, val1);
            }

            int rds_ct = 1;
            // iterate over RDS trees
            for (Iterator it = makeRDSIterator(g,NUM_SEEDS); it.hasNext();) {
                // get the next RDS tree
                RDS rds = (RDS)it.next();
                if (DEBUG) System.out.println(""+rds.toString());

                int comp_ct = 1;
                // iterate over completions
                for (Iterator it2 = makeCompletionIterator(rds,g); it2.hasNext();) {
                    // get the next completion
                    Completion comp = (Completion)it2.next();
                    if (DEBUG) System.out.println(""+comp.toString());

                    // make the measurement apparatus
                    Measurable measComp = makeMeasurable(comp._completion);
```

```
                    // now take measurements of each vertex in the completion
                    for (Iterator vit= g.getVertices().iterator(); vit.hasNext(); ) {
                        // for each vertex in G
                        Vertex vquery_g = (Vertex)vit.next();
                        Vertex vquery_r = rds.getRDSVertex(vquery_g);
                        if (vquery_r==null) {
                            // only consider vertices also in the RDS tree
                            continue;
                        }

                        // compute measures for vertices in the completion
                        Vertex vquery_c = comp.getCompletionVertex(vquery_r);
                        double val3 = measComp.readValue(vquery_c);
                        _stats.accum(vquery_g, val3);
                    }

                    WeakComponentClusterer wcc = new WeakComponentClusterer();
                    ClusterSet cs = wcc.extract(comp._completion);
                    if (DEBUG) System.out.println("Components = "+cs.size());

                    stage++;
                    if (stage % blip == 0) {
                        // output aggregate statistics to a file every blip completions....
                        output(stage, g);
                    }

                    comp_ct++;
                    if (comp_ct> NUM_COMPLETIONS ) {
                        // done making completions
                        break;
                    }
                }

                rds_ct++;
                if (rds_ct> NUM_RDS_TREES ) {
                    // done making RDS trees
                    break;
                }
            }

            // output the final aggregate statistics
            output(-1, g);

            // output the lift curve & pearson
            outputError(g);
            outputErrorPearson(g);
        }
        catch (IOException ex) {
            System.out.println("I/O error: "+ex);
        }
    }

    public static void outputError(Graph g) {
        try {
            String logname = "output."+NUM_SEEDS+"."+NUM_RDS_TREES+"."+NUM_COMPLETIONS+".error";
            BufferedWriter out = new BufferedWriter(new FileWriter(logname));
            for (double th=0.5; th<1.0; th+=0.05) {
                out.write(_stats.error(g, th)+"\n");
            }
            out.close();
        } catch (IOException ex) {
        }
    }

    public static void outputErrorPearson(Graph g) {
        try {
            String logname = "output."+NUM_SEEDS+"."+NUM_RDS_TREES+"."+NUM_COMPLETIONS+".pearson";
            BufferedWriter out = new BufferedWriter(new FileWriter(logname));
            out.write("r = "+twoPlaces.format(_stats.pearson(g)));
            out.close();
        } catch (IOException ex) {
        }
    }

    public static void output(long stage, Graph g) {
        try {
            String logname;
```

```
            if (stage<0) {
                logname = "output."+NUM_SEEDS+"."+NUM_RDS_TREES+"."+NUM_COMPLETIONS+"-final";
            }
            else {
                logname = "output."+NUM_SEEDS+"."+NUM_RDS_TREES+"."+NUM_COMPLETIONS+"-"+stage;
            }

            BufferedWriter out = new BufferedWriter(new FileWriter(logname));
            for (Iterator vit= g.getVertices().iterator(); vit.hasNext(); ) {
                Vertex vquery_g = (Vertex)vit.next();
                double grank = _stats.get_grank(vquery_g);
                double crank;
                double ctr =  _stats.get_count(vquery_g);
                try {
                    crank = _stats.get_crank(vquery_g);
                }
                catch (Exception ex2) {
                    // skip the vertex for which we have no data
                    continue;
                }
                double diff = Math.abs(crank-grank);
                out.write(""+
                          twoPlaces.format(diff)+"\t"+
                          twoPlaces.format(ctr)+"\t"+
                          twoPlaces.format(grank)+"\t"+
                          twoPlaces.format(crank)+"\t"+
                          vquery_g+"\n");
            }
            out.close();
        } catch (IOException ex) {
        }
    }

    public static void debugOutput(Graph g) {
        for (Iterator vit= g.getVertices().iterator(); vit.hasNext(); ) {
            Vertex vquery_g = (Vertex)vit.next();
            if (vquery_g.inDegree() != vquery_g.outDegree()) {
                System.out.print("***");
            }
            System.out.println(" "+vquery_g+" in="+vquery_g.inDegree()+" out="+vquery_g.outDegree());
        }
        System.exit(0);
    }

    public static Graph undirectify(Graph g) {
        // duplicate vertices
        return DirectionTransformer.toUndirected(g, false);
    }

    public static Vertex getBigCompSeed(Graph g) {

        Vertex bigcompSeed = null;
        WeakComponentClusterer wccg = new WeakComponentClusterer();
        ClusterSet csg = wccg.extract(g);

        int big = -1;
        for (Iterator itc=csg.iterator(); itc.hasNext();) {
            Set clus = (Set)itc.next();
            if (clus.size() > big) big = clus.size();
        }

        for (Iterator itc=csg.iterator(); itc.hasNext();) {
            Set clus = (Set)itc.next();
            if (clus.size()==big) {
                Object[] varray = clus.toArray();
                int r = (int)(Math.random() * varray.length);
                bigcompSeed = (Vertex)varray[r];
                if (DEBUG) System.out.println("seed "+bigcompSeed+" has cluster size "+big);
                break;
            }
        }
        return bigcompSeed;
    }
};
```

```
#!/bin/csh -f
set F = ‘ls *.png‘
gnuplot < gnuplot.script
pngtopnm output.png > output.pnm
pnmtops output.pnm > output.ps
mv output.png $F
cp $F ~/Desktop/RDS
```

```
#!/bin/csh -f
set SEEDS = $1
set TREES = $2
set COMPS = $3
set BIASED = $4
set EXP = $5

if ($BIASED == "0") then
    set MODE = "Unbiased RDS/Completions"
endif
if ($BIASED == "1") then
    set MODE = "Biased RDS/Completions"
endif

#---------------------------------------
if ($EXP == "BC") then
    set MEASURE = "Betweenness Centrality"
endif
if ($EXP == "ES") then
    set MEASURE = "Effective Size"
endif
if ($EXP == "H") then
    set MEASURE = "Hubness"
endif
if ($EXP == "A") then
    set MEASURE = "Authority"
endif
if ($EXP == "CON") then
    set MEASURE = "Constraint"
endif

set TITLE = "Estimating $MEASURE w/ $MODE ($TREES trs x $COMPS cmps)"

java -cp ./classes:lib/colt.jar:lib/jung-1.7.6.jar:lib/resolver.jar:lib/xercesImpl.jar:lib/xml-apis.jar:lib/c
oncurrent.jar:lib/jung-1.7.6-src.jar:lib/serializer.jar:lib/xercesSamples.jar:lib/commons-collections-3.2.jar
:lib/commons-collections-testframework-3.2.jar netsci.Test /home/bilal/dev/netsci/dyads.txt $SEEDS $TREES $CO
MPS $BIASED $EXP

set DIR = "exp-$EXP.$BIASED.$SEEDS.$TREES.$COMPS.dat"

mkdir -p $DIR

set PEARSON = ‘cat output.$SEEDS.$TREES.$COMPS.pearson‘

mv "output.$SEEDS.$TREES.$COMPS.error" $DIR
mv "output.$SEEDS.$TREES.$COMPS.pearson" $DIR

foreach F (‘ls output.$SEEDS.$TREES.$COMPS-*’)
    cat $F | sort -k 1 -n > $F.sorted
    mv $F.sorted $F
    mv $F "$DIR"
end

cd $DIR
cat "output.$SEEDS.$TREES.$COMPS.pearson" | sed -e ’s/r = //’ > "output.$SEEDS.$TREES.$COMPS.pearson2"
echo "" >> "output.$SEEDS.$TREES.$COMPS.pearson2"
```

```
#!/bin/csh -f
set SEEDS = $1
set TREES = $2
set COMPS = $3
set BIASED = $4
set EXP = $5

if ($BIASED == "0") then
    set MODE = "Unbiased RDS/Completions"
endif
if ($BIASED == "1") then
    set MODE = "Biased RDS/Completions"
endif

#---------------------------------------
if ($EXP == "BC") then
    set MEASURE = "Betweenness Centrality"
endif
if ($EXP == "ES") then
    set MEASURE = "Effective Size"
endif
if ($EXP == "H") then
    set MEASURE = "Hubness"
endif
if ($EXP == "A") then
    set MEASURE = "Authority"
endif
if ($EXP == "CON") then
    set MEASURE = "Constraint"
endif

set TITLE = "Estimating $MEASURE w/ $MODE ($TREES trs x $COMPS cmps)"

java -cp ./classes:lib/colt.jar:lib/jung-1.7.6.jar:lib/resolver.jar:lib/xercesImpl.jar:lib/xml-apis.jar:lib/c
oncurrent.jar:lib/jung-1.7.6-src.jar:lib/serializer.jar:lib/xercesSamples.jar:lib/commons-collections-3.2.jar
:lib/commons-collections-testframework-3.2.jar netsci.Test /home/bilal/dev/netsci/dyads.txt $SEEDS $TREES $CO
MPS $BIASED $EXP

set DIR = "exp-$EXP.$BIASED.$SEEDS.$TREES.$COMPS.dat"

mkdir -p $DIR

set PEARSON = ‘cat output.$SEEDS.$TREES.$COMPS.pearson‘

mv "output.$SEEDS.$TREES.$COMPS.error" $DIR
mv "output.$SEEDS.$TREES.$COMPS.pearson" $DIR

echo "set terminal png" > "$DIR"/gnuplot.script
echo "set output "\""output.png"\""" >> "$DIR"/gnuplot.script
echo "set xlabel "\""Actual $MEASURE"\""" >> "$DIR"/gnuplot.script
echo "set ylabel "\""Estimated $MEASURE"\""" >> "$DIR"/gnuplot.script
echo "set title "\""$TITLE"\""" >> "$DIR"/gnuplot.script
echo "plot "\""output.$SEEDS.$TREES.$COMPS-final"\"" using 3:4 title "\""$PEARSON"\""" >> "$DIR"/gnuplot.scri
pt

foreach F (‘ls output.$SEEDS.$TREES.$COMPS-*‘)
    cat $F | sort -k 1 -n > $F.sorted
    mv $F.sorted $F
    mv $F "$DIR"
end

cd "$DIR"
gnuplot < gnuplot.script

pngtopnm output.png > output.pnm
pnmtops output.pnm > output.ps
echo "Done"

mv output.png "$EXP-$BIASED-$SEEDS-$TREES-$COMPS.png"

evince output.ps
```

```
#!/bin/csh -f
set SEEDS = $1
set TREES = $2
set COMPS = $3
set BIASED = $4
set EXP = $5

set DIR = "exp-$EXP.$BIASED.$SEEDS.$TREES.$COMPS.dat"
mkdir -p $DIR
rm -f $DIR/rds-trees
touch -f $DIR/rds-trees

foreach i (1 2 3 4 5 6 7 8 9 10)
    ./run-fast.sh $*
    cat $DIR/*.pearson2 >> $DIR/rds-trees
end

 ./summary < $DIR/rds-trees > $DIR/rds-stats

cat $DIR/rds-stats
```

```
all:    summary
        javac -d ./classes -classpath lib/colt.jar:lib/jung-1.7.6.jar:lib/resolver.jar:lib/xercesImpl.jar:lib
/xml-apis.jar:lib/concurrent.jar:lib/jung-1.7.6-src.jar:lib/serializer.jar:lib/xercesSamples.jar:lib/commons-
collections-3.2.jar:lib/commons-collections-testframework-3.2.jar RDS.java RDSBiased.java Test.java Completio
n.java CompletionBiased.java Measurable.java MeasurableBC.java MeasurableBCRaw.java MeasurableEffectiveSize.j
ava MeasurableHierarchy.java MeasurableConstraint.java MeasurableAggregateConstraint.java MeasurableHubness.j
ava MeasurableAuthority.java Stats.java

summary:        summary.c
        gcc summary.c -o summary -lm

run:
        java -cp ./classes:lib/colt.jar:lib/jung-1.7.6.jar:lib/resolver.jar:lib/xercesImpl.jar:lib/xml-apis.j
ar:lib/concurrent.jar:lib/jung-1.7.6-src.jar:lib/serializer.jar:lib/xercesSamples.jar:lib/commons-collections
-3.2.jar:lib/commons-collections-testframework-3.2.jar netsci.Test /home/bilal/dev/netsci/dyads.txt

demo:
        java -cp ./classes:lib/colt.jar:lib/jung-1.7.6.jar:lib/resolver.jar:lib/xercesImpl.jar:lib/xml-apis.j
ar:lib/concurrent.jar:lib/jung-1.7.6-src.jar:lib/serializer.jar:lib/xercesSamples.jar:lib/commons-collections
-3.2.jar:lib/commons-collections-testframework-3.2.jar samples.graph.GraphEditorDemo 5 5

tidy:
        rm *~;

clean:
        rm -rf ./classes/*;
```