

Dynamics of Negotiation in a Network of Intelligent Software Agents: Technical Report*

Minh Van Nguyen
nguyenminh2@gmail.com

27 February 2009

1 Early studies of social networks

Milgram [8] is one of the early quantitative studies of the structure of social networks. The study describes an experiment in which Milgram wished to send a number of letters to his friend in another city. The letters were first distributed to a random selection of people. These people were instructed to deliver the letters to the addressee, under the conditions that the letters must be passed from person to person, and the passers were permitted to only deliver the letters to people whom they knew on a first-name basis. For those letters that eventually reached the intended addressee, it was found that on average six steps were required for a letter to reach its destination. The path length of six within social networks is colloquially known as the “six degrees of separation”. Within mathematical circles, a similar type of social network is found in the scientific collaboration network of Erdős numbers [2].

2 Watts-Strogatz small-world networks

Watts and Strogatz [11] study a class of networks that has become known as small-world networks. The Watts-Strogatz model considers a generic graph G having N vertices and K edges, and satisfying the following properties:

1. G is an unweighted or topological graph.
2. G is simple in that it has no loops and no multiple edges.
3. G is sparse in the sense that $K \ll \frac{N(N-1)}{2}$.
4. G is connected such that there is a path between any distinct pair of vertices.

*The author acknowledges financial support from the Commonwealth Scientific and Industrial Research Organisation (CSIRO) during this project. The project was carried out while the author was a 2008/2009 Summer Research Scholar within CSIRO’s Mathematical and Information Sciences division, supervised by CSIRO research scientist Dr Rodolfo Garcia-Flores.

For a random graph, the quantities N and K must satisfy

$$N \gg K \gg \ln(N) \gg 1$$

where $K \gg \ln(N)$ guarantees that the graph is connected [1].

2.1 Characteristic path length and clustering coefficient

Watts and Strogatz [11] analyze the structure of such a network by means of two quantities: the characteristic path length L ; and the clustering coefficient C . Let $\{d_{ij}\}$ be the geodesic matrix of G , i.e. the matrix of shortest edge counts between pairs of vertices in G . Then the characteristic path length L is defined as the average shortest path between distinct pairs of vertices in G :

$$L(G) = \frac{1}{N(N-1)} \sum_{i \neq j \in V(G)} d_{ij} \quad (1)$$

which is a global property of G . Furthermore, Watts and Strogatz also consider a local property of G , called the clustering coefficient. To define the clustering coefficient of G , they first introduce the local clustering coefficient C_i of vertex i :

$$C_i = \frac{K_i}{N_i(N_i-1)/2}$$

where K_i is the number of edges in the graph of immediate neighbours of i and N_i is the number of immediate neighbours of vertex i . The graph of immediate neighbours of i is a subgraph of G . It consists of all vertices ($\neq i$) that are adjacent to i , preserving the adjacency relation among those vertices as found in the supergraph G . Then the clustering coefficient C of G is defined by

$$C(G) = \frac{1}{N} \sum_{i \in V(G)} C_i$$

where the sum is taken over all vertices i of G . The quantity C can be interpreted to mean the average cliquishness of vertices in G , hence C is known as a local property of G .

2.2 The Watts-Strogatz model

In [11], Watts and Strogatz propose an edge rewiring method for constructing a class of graphs that interpolate between a regular lattice and a random graph. Known as the Watts-Strogatz model, the method starts with a one-dimensional lattice G having N vertices, periodic boundary conditions, and each vertex connecting to its k neighbours for some even k . Identify the vertex set $V(G)$ with the elements of the ring $\mathbb{Z}/N\mathbb{Z}$ for some fixed integer $N > 2$. The lattice can be conceptualized as a circulant graph, where each vertex $i \in \mathbb{Z}/N\mathbb{Z}$ is linked by an edge with each of the vertices $i + j$ and $i - j$ for each $j \in \{1, 2, \dots, k/2\}$, where vertex arithmetic is performed modulo N . We refer to such a graph as a k -circulant graph on N vertices, or a ring lattice of N nodes and per-vertex

degree k . Small-world networks are graphs that are intermediate between regular ring lattices and Erdős-Rényi [3] random graphs. Figure 1 illustrates the interpolation from a ring lattice with rewiring probability $p = 0$ to an Erdős-Rényi random graph where the rewiring probability is $p = 1$. The graphs are produced using Sage's [10] interface to the NetworkX [4] Python package.

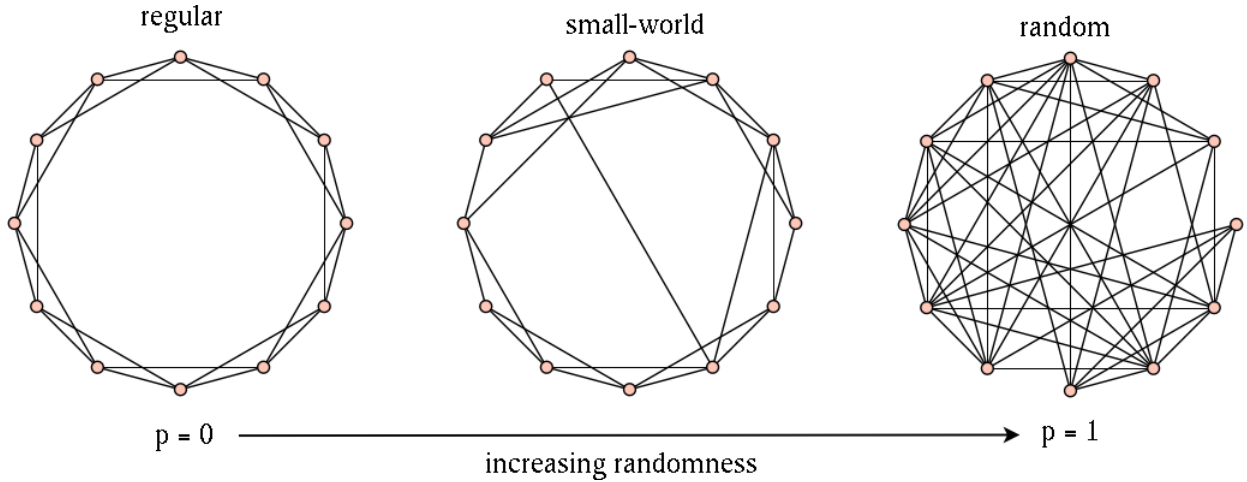


Figure 1: From a regular ring lattice (left) to a random graph (right).

Given a k -circulant graph on N vertices, the Watts-Strogatz rewiring procedure is as follows. Let the probability of choosing a vertex be uniformly distributed. Rewire each vertex with probability p to another vertex chosen at random. The rewiring must result in a graph that:

1. has no multiple edges;
2. has no loops; and
3. the number of edges does not change.

The Watts-Strogatz model does not specifically require that a rewired graph be connected, hence the result of one round of random edge rewiring may be a disconnected graph. However, by definition of the characteristic path length in (1), the underlying graph must be connected, otherwise the geodesic matrix $\{d_{ij}\}$ has ∞ as one of its entries.

Figure 2 shows a plot of the characteristic path lengths and clustering coefficients normalized. The horizontal axis follows a log scale. The plotted metrics were obtained in an effort to verify by computer simulation results reported in [11]. The ring lattice in question is a 10-circulant graph on 1000 vertices with 37 rewiring probability points. The rewiring probabilities are chosen as follows. Let G be a k -circulant graph on N vertices. For $i = 1, 2, \dots, r$ the i -th rewiring probability p_i is given by

$$p_i = p_{\min} \times F^{i-1} \quad \text{with} \quad F = \left(\frac{p_{\max}}{p_{\min}} \right)^{1/(r-1)} \quad (2)$$

where p_{\min} and p_{\max} are the minimum and maximum rewiring probabilities, respectively.

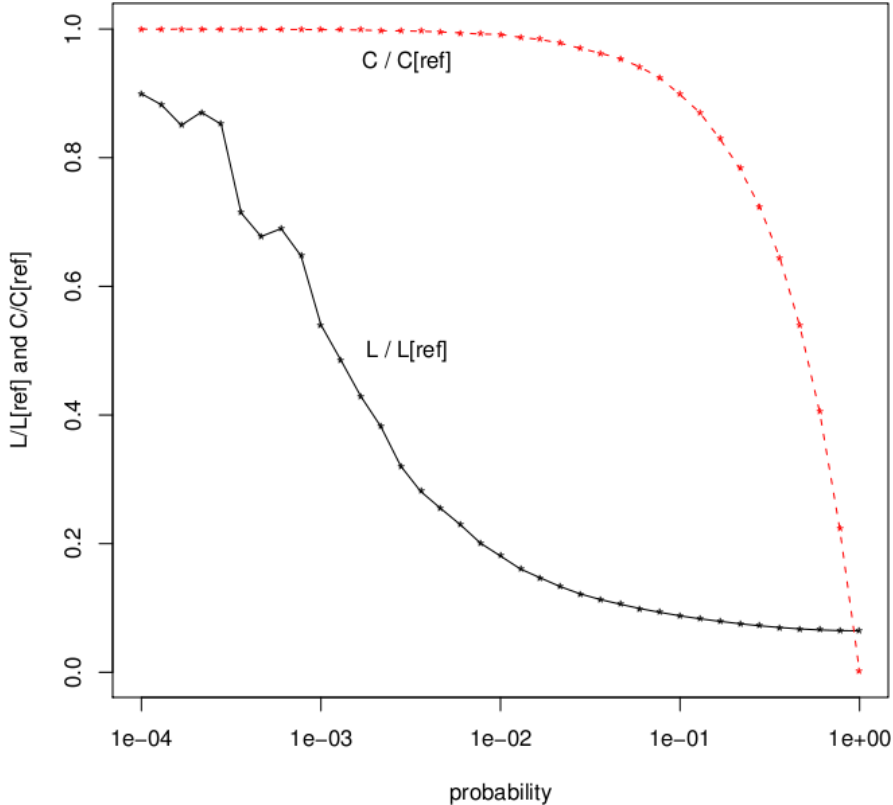


Figure 2: Normalized characteristic path lengths and clustering coefficients.

Next, we describe the procedure for normalizing L and C . Let B be the number of batches of r ring lattices to be rewired with probabilities chosen according to (2). That is, each batch contains r k -circulant graphs on N vertices and the i -th graph from each batch is to be rewired with probability p_i . In particular, our computer simulation rewired a total of $Br = 20 \times 37 = 740$ ring lattices. Define G_{p_i} as the connected graph resulting from rewiring G with probability p_i . For each rewiring probability p_i , define the normalized characteristic path length (respectively clustering coefficient) by

$$\text{norm}_{p_i}(L) = \frac{1}{B} \sum_{G_{p_i}} \frac{L(G_{p_i})}{L(G)} \quad \text{and} \quad \text{norm}_{p_i}(C) = \frac{1}{B} \sum_{G_{p_i}} \frac{C(G_{p_i})}{C(G)} \quad (3)$$

where each sum is taken over all graphs G_{p_i} . From Figure 2, we note that there is a range of rewiring probabilities that result in connected graphs with high C and a rapid decrease in L . This is qualitatively consistent with results reported in [11]. The decrease in L is attributed to a number of vertices with links to distant vertices, while the value of C remains high because only a relatively small proportion of vertices have long-range connections. This phenomenon of graphs having the twin characteristics of high cliquishness and low average path length is referred to as the small-world effect.

3 Generalizing the Watts-Strogatz model

Whereas [11] uses the characteristic path length L and clustering coefficient C to study small-world networks, Latora and Marchiori [7] generalize the method by using the notions of local and global efficiencies as defined in section 3.1. The generalization is applicable to both directed and undirected graphs, as well as weighted and unweighted graphs. For weighted graphs, the weight can be a cost associated with the edge connecting two vertices. A graph G with low cost is said to be economic, while G is said to exhibit small-world behaviour provided that it has high efficiency at both the local and global levels. If G has these two properties—both economic and efficiency—then it is referred to as an economic small-world.

3.1 Global and local efficiencies and network cost

Let G be a graph (either weighted or unweighted) having N vertices and K edges. To define local and global efficiencies, Latora and Marchiori [7] introduce the concept of average efficiency. If i and j are distinct vertices of G , let d_{ij} be the shortest path length between i and j . Then the average efficiency of G is

$$E(G) = \frac{1}{N(N-1)} \sum_{i \neq j \in V(G)} \frac{1}{d_{ij}} \quad (4)$$

Let κ_N be the complete graph on N vertices so that $E(\kappa_N)$ is the average efficiency of κ_N . Define the global efficiency of G as

$$E_{\text{glob}} = \frac{E(G)}{E(\kappa_N)} \quad (5)$$

For each vertex i of G , let G_i be the subgraph of neighbours of i . Then vertex i is excluded from the vertex set V_i of G_i . Define the local efficiency of G by

$$E_{\text{loc}} = \frac{1}{N} \sum_{i \in V(G)} \frac{E(G_i)}{E(\kappa_{|V_i|})} \quad (6)$$

where $|V_i|$ is the cardinality of V_i . Note that the metrics (4), (5) and (6) are also applicable to directed graphs as well as weighted graphs. The cost of G can be defined as

$$C_G = \frac{\sum_{i \neq j \in V(G)} a_{ij} \gamma(\ell_{ij})}{\sum_{i \neq j \in V(G)} \gamma(\ell_{ij})}$$

where $\{a_{ij}\}$ and $\{\ell_{ij}\}$ are the adjacency and weight matrices of G , respectively. In the Watts-Strogatz model, the weight ℓ_{ij} assigned to the edge connecting vertices i and j is $\ell_{ij} = 1$. The cost evaluator function γ measures the cost needed to set up a connection with a given length. The Watts-Strogatz model assumes γ to be the identity function

$\gamma(l_{ij}) = l_{ij} = 1$ for all $i \neq j$. Thus $\{a_{ij}\} = \{l_{ij}\}$ holds for the specific case of the Watts-Strogatz model and therefore

$$C_G = \frac{\sum_{i \neq j \in V(G)} a_{ij} \gamma(l_{ij})}{\sum_{i \neq j \in V(G)} \gamma(l_{ij})} = \frac{\sum_{i \neq j \in V(G)} a_{ij}}{\sum_{i \neq j \in V(G)} 1} = \frac{2K}{N(N-1)} \quad (7)$$

However, for weighted graphs Latora and Marchiori [7] define the network cost as

$$C_G = \frac{\sum_{i \neq j \in V(G)} a_{ij} l_{ij}}{\sum_{i \neq j \in V(G)} l_{ij}} \quad (8)$$

where l_{ij} is defined in (9).

Appendix A contains an R [9] script implementing the Latora-Marchiori metrics for graphs that are unweighted, undirected and connected.

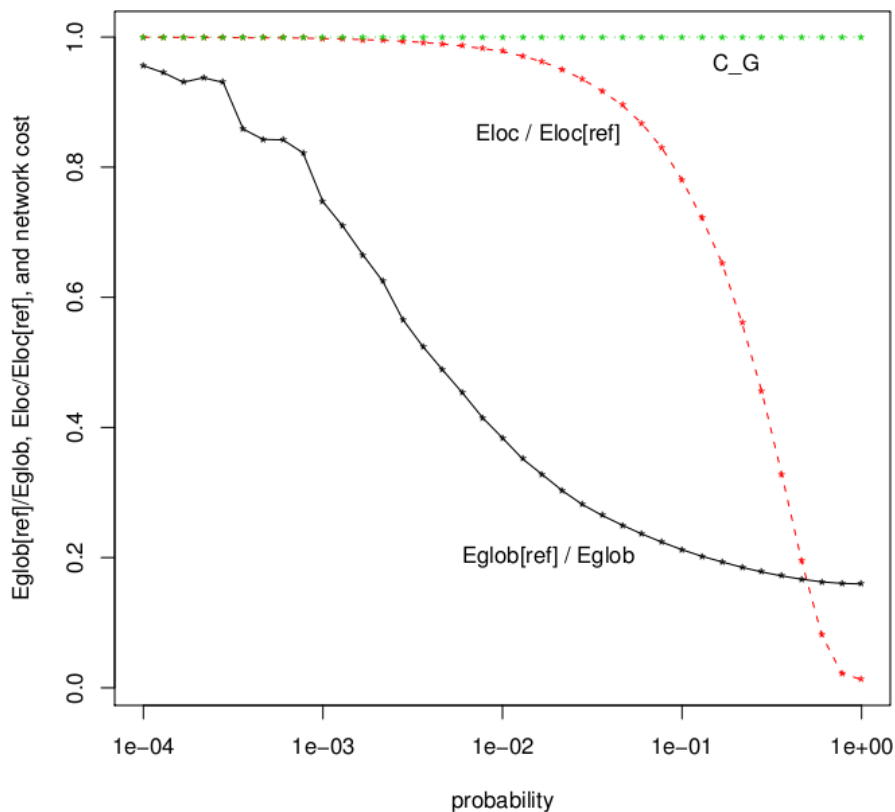


Figure 3: Normalized global and local efficiencies.

Figure 3 shows a plot of the global and local efficiencies normalized, together with normalized network costs. The results are similar to those reported by Latora and Marchiori.

The metrics were obtained from computer simulation of random edge rewiring of 20 batches of 37 ring lattices, each lattice being a 10-circulant graph on 1000 vertices. That is, each batch contained 37 ring lattices and therefore a total of $20 \times 37 = 740$ ring lattices to be rewired. Edges of the i -th ring lattice of each batch were rewired with the probability in (2). The Latora-Marchiori metrics (5) and (6) were then calculated on the rewired 740 graphs. The normalized E_{glob} and E_{loc} corresponding to rewiring probability p_i were obtained using a normalization procedure similar to (3). In particular, let B be the number of batches with r rewiring probabilities chosen according to (2), let G be a k -circulant graph on N vertices, and let G_{p_i} be the connected graph resulting from rewiring G with probability p_i . For each rewiring probability p_i , the normalized local and global efficiencies are defined by

$$\text{norm}_{p_i}(E_{\text{glob}}) = \frac{1}{B} \sum_{G_{p_i}} \frac{E_{\text{glob}}(G)}{E_{\text{glob}}(G_{p_i})} \quad \text{and} \quad \text{norm}_{p_i}(E_{\text{loc}}) = \frac{1}{B} \sum_{G_{p_i}} \frac{E_{\text{loc}}(G_{p_i})}{E_{\text{loc}}(G)}$$

where each sum is taken over all graphs G_{p_i} that have been rewired with probability p_i . The normalized network cost is similarly defined by

$$\text{norm}_{p_i}(C_G) = \frac{1}{B} \sum_{G_{p_i}} \frac{C_G}{C_{G_{p_i}}}$$

However, by definition of C_G for unweighted, undirected graphs as specified by (7), it is clear that $\text{norm}_{p_i}(C_G) = 1$ for all rewiring probabilities. Further details can be found in Appendix A.

3.2 Extending the Watts-Strogatz model to weighted networks

This section considers Latora and Marchiori's [7] generalization of the Watts-Strogatz model to the case of weighted, undirected networks. The network is a k -circulant graph on N vertices where $N = 1000$ and $k = 6$. After generating a ring lattice satisfying these parameters, one would get a graph G with $K = 3000$ edges. The Latora-Marchiori approach, as detailed in "Model 4" of [7], is to randomly eliminate $K/2 = 1500$ of the edges of G and then proceed with the rewiring process of the Watts-Strogatz model. The weight of each edge is defined in terms of the Euclidean distance. In particular, if i and j are vertices of G for $i, j = 1, 2, \dots, N$ then the Euclidean distance between i and j is

$$\ell_{ij} = \frac{2 \sin(|i - j|\pi/N)}{2 \sin(\pi/N)} = \frac{\sin(|i - j|\pi/N)}{\sin(\pi/N)} \quad (9)$$

Note that the metric (9) is specific to ring lattices. The distance between each pair of neighbouring vertices is $\ell_{ij} = 1$ and the distance from i to itself is trivially $\ell_{ii} = 0$. The weight matrix of G is denoted $\{\ell_{ij}\}$, which has zero along the main diagonal and is symmetric about this diagonal. For unweighted graphs, the geodesic matrix $\{d_{ij}\}$ is a matrix of minimum edge counts separating each pair of vertices i and j . If there are no paths from i to j , where $i \neq j$, then Latora and Marchiori [7] define $d_{ij} = +\infty$. In case $i = j$, then $d_{ij} = 0$. On the other hand, for weighted graphs the weight matrix $\{\ell_{ij}\}$ can be interpreted as the matrix of physical distances. Then d_{ij} is the minimum sum of

physical distances from i to j . Furthermore, $d_{ij} = 0$ if $i = j$, and $d_{ij} = +\infty$ whenever there are no paths from i to j .

The following scripts support computer simulation of weighted, undirected networks as described in “Model 4”:

- `k_circulant_n.sage` — This Sage script generates ring lattices, each with half the total number of edges removed.
- `rewire-lattices.r` — This R script can be used to rewire (n, k) ring lattices that have had 50 percent of their total number of edges removed.
- `mat2r.py` — This Python script converts text representation of a Latora-Marchiori network to its R code representation.
- `network-metrics-lm.r` — An R script to compute network metrics of weighted, undirected Latora-Marchiori networks.

Further details on these scripts can be found in Appendix B. Using the above scripts, the computed network metrics are plotted using R and shown in Figure 4. The results are qualitatively similar to those reported in [7].

4 Conclusion & further research

In this paper, we have provided verification of results reported in [7]. The reported results are qualitatively similar to those contained in [7].

In [6], Kaihara formulates the problem of virtual market based supply chain management (SCM) in terms of a discrete resource allocation problem, and proposes an algorithm for SCM under a dynamic environment. The simulation reported in [6] concerns a single input/output circulatory resource flow within a network of two economic agents and two virtual markets.

As a direction for future research that incorporates a network approach to economics, we propose to use the Latora-Marchiori network metrics in computer simulations of a multi-agent network of buyers and sellers. Instead of the edge weight (9), we propose to use a multi-dimensional version of the Cobb-Douglas or constant elasticity of substitution functions [5]. Our research approach has the advantage of generalizing [6] to the case of multiple input and output.

Revision

- 2010-01-09 — Some clarification suggested by David Joyner (US Naval Academy), including: explaining what is the graph of immediate neighbours of a vertex; and some improvements to the exposition of the paper.

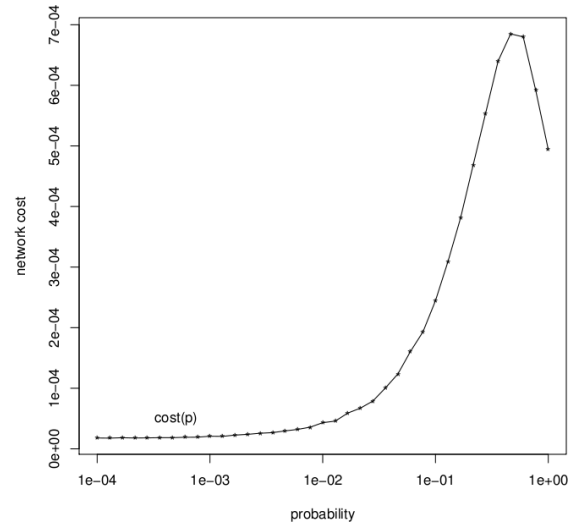
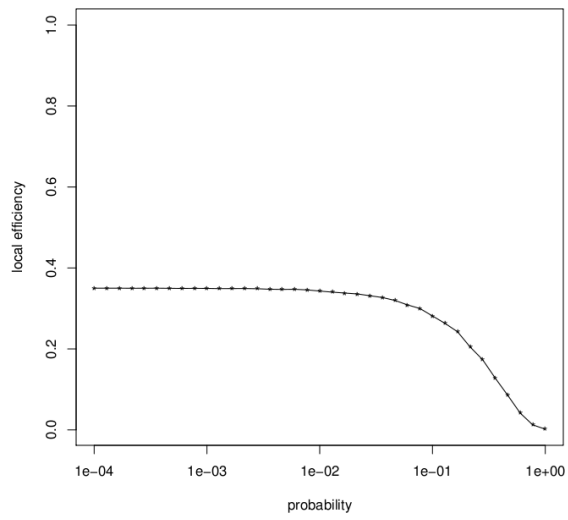
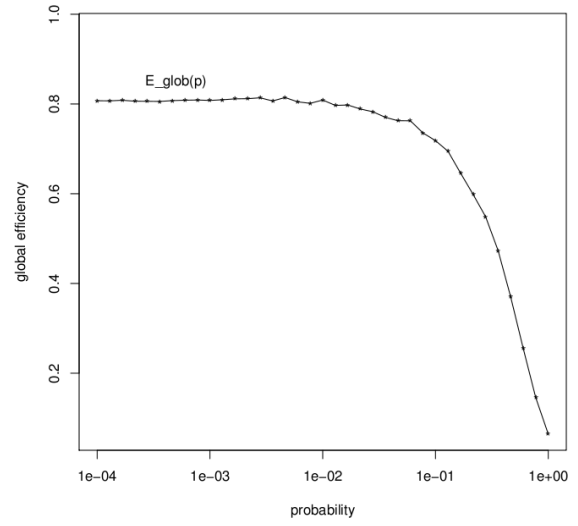
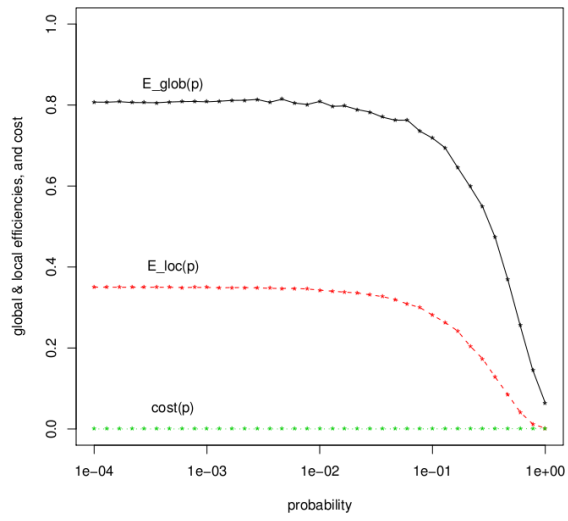


Figure 4: Network metrics for weighted, undirected graphs.

A Latora-Marchiori metrics: unweighted

This appendix presents a script written in the R [9] language for computing local and global efficiencies and network cost of unweighted, undirected graphs. The graphs in question have $N = 1000$ vertices and per-vertex degree $k = 10$. Starting with a 10-circulant graph G on 1000 vertices, the script rewires G with probability p according to the Watts-Strogatz random edge rewiring method to produce a rewired connected graph G' . The global and local efficiencies of G' are then calculated according to (5) and (6), respectively. The network cost is calculated using (7). The 37 rewiring probabilities are chosen according to (2).

```
1 # sw-latora-marchiori.r -- modelling small-world networks
2 # Copyright (C) 2008 Minh Van Nguyen <nguyenminh2@gmail.com>
3 #
4 # An R script to model small-world networks. This implements techniques
5 # in the paper:
6 #
7 # V. Latora & M. Marchiori. Economic small-world behavior in weighted
8 # networks. The European Physical Journal B, 32(2):249--263, 2003.
9 #
10 # which generalizes the approach described in:
11 #
12 # D. Watts & S.H. Strogatz. Collective dynamics of "small-world"
13 # networks. Nature, 393(4):440--442, 1998.
14 #
15 # Rodolfo Garcia-Flores has written an R script that implements the
16 # Watts-Strogatz model described in (Watts & Strogatz 1998).
17 #
18 # Minh Van Nguyen extended Rodolfo's code based on a generalization
19 # of the Watts-Strogatz model as detailed in the paper
20 # (Latora & Marchiori 2003).
21 #
22 # This program is free software; you can redistribute it and/or modify
23 # it under the terms of the GNU General Public License as published by
24 # the Free Software Foundation; either version 2 of the License, or
25 # (at your option) any later version.
26 #
27 # This program is distributed in the hope that it will be useful,
28 # but WITHOUT ANY WARRANTY; without even the implied warranty of
29 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
30 # GNU General Public License for more details.
31 #
32 # You should have received a copy of the GNU General Public License
33 # along with this program; if not, write to the Free Software
34 # Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
35
36
37 # Clear memory, removing (almost) everything in the working environment
38 # without any warning. Be careful with what you wish for.
39 rm(list = ls())
40 # For colours and fonts
41 library(grDevices)
42 # For fit, simulate and diagnose exponential-family models for networks
43 library("ergm")
44 # For social network analysis
45 library("sna")
46 # library("igraph")
47
48
49 # =====
50 # 1. SETUP (DATA, MODELS AND FUNCTIONS)
51 # =====
```

```

52
53
54 ### Problem data
55
56
57 # Actual values should be n = 1000, k = 10. When run with these values,
58 # the script should take a few hours to complete. Test values can be
59 # n = 20, k = 4
60 n <- 1000           # vertices
61 k <- 10            # edges per vertex, MUST BE EVEN.
62 timesToRepeat <- 20 # should be 20
63
64 # Data for a logarithmically-scaled probability vector.
65 numberOfPoints <- 37
66 minProb <- 1e-4
67 maxProb <- 1
68
69 # Directories and file names
70 # Subdirectory names to organise I/O.
71 input <- "input"
72 output <- "output"
73
74 # Files in input directory to search for.
75 # Output file names
76 summaryFileName <- "small-world-summary.txt"
77
78 # Platform-specific directory separator.
79 slash <- .Platform$file.sep
80
81 # (Relative) subdirectory paths.
82 outputDir <- paste(".", slash, output, slash, sep = "")
83 inputDir <- paste(".", slash, input, slash, sep = "")
84
85
86 ### Functions
87
88
89 # Simulates a circular list. We are only interested in the index i of a
90 # member of this list, which has n members. One way to conceptualize this
91 # list is to visualize all n members as arranged in a cycle graph, in
92 # which each member i has an edge connecting it to i + 1, and an edge
93 # connecting it to i - 1. Another way to think about this function is to
94 # interpret it as a simple implementation of the group  $Z/Z_n$ , where only
95 # the index of each i in  $Z/Z_n$  is returned.
96 #
97 # INPUT:
98 #   index -- integer; index of an element in this circular list.
99 #   length -- integer > 0; the number of elements in this circular list.
100 #   FIXME: Maybe it's a good idea to implement the case where
101 #   length < 0, or provide some sanity checking to take care of that
102 #   possibility.
103 #
104 # OUTPUT:
105 #   If  $0 < \text{index} \leq \text{length}$ , then return index. If  $\text{index} > \text{length}$ , then
106 #   return  $\text{index} \bmod \text{length}$ . Else  $\text{index} < 0$ , so return  $\text{index} \bmod \text{length}$ .
107 #   If length is 0, then return NaN (not a number).
108 #
109 # AUTHOR:
110 #   Rodolfo Garcia-Flores
111 #   Documentation by Minh Van Nguyen <nguyenminh2@gmail.com>
112 #
113 returnIndex <- function(index, length) {
114   if ((index > 0) && (index <= length)) {
115     index
116   }
117   else if (index > length) {

```

```

118     index - length * floor(index / length)
119   }
120   else {
121     (length + index) + length * floor(-1 * index / length)
122   }
123 }
124
125 # The re-wiring routine.
126 #
127 # INPUT:
128 #   aMatrix -- an adjacency matrix.
129 #   aProbability -- double; a probability value p such that 0 < p < 1. This
130 #   value determines the probability that an edge incident on a vertex
131 #   is re-wired.
132 #
133 # OUTPUT:
134 #   An adjacency matrix with a number of the vertices re-wired.
135 #
136 # AUTHOR:
137 #   Rodolfo Garcia-Flores
138 #   Documentation by Minh Van Nguyen <nguyenminh2@gmail.com>
139 #
140 reWire <- function(aMatrix, aProbability) {
141   currentMat <- aMatrix
142   for (i in 1:n) {
143     for (j in 1:i) {
144       if ((currentMat[i,j] != 0) && (runif(1) < aProbability)) {
145         # To vertices different to i and
146         # different to those already connected,
147         # preferably to nodes that are isolated.
148         # This should prevent having isolated regions.
149         isolatedNodes <- c(1:n)[colSums(currentMat[,]) == 0]
150         nodesAlreadyConnected <- c(1:n)[currentMat[i,] > 0]
151         excludedNodes <- c(i, nodesAlreadyConnected)
152         notExcludedNodes <- (1:n)[-excludedNodes]
153
154         # A list whose first elements are isolated nodes, the rest are
155         # shuffled, not-excluded values.
156         validPrioritisedNodes <- c(isolatedNodes,
157                                   sample(setdiff(notExcludedNodes,
158                                                    isolatedNodes)))
159
160         # Take first node index in list.
161         newVertex <- validPrioritisedNodes[1]
162         currentMat[i,j] <- 0
163         currentMat[j,i] <- 0
164         currentMat[i, newVertex] <- 1
165         currentMat[newVertex, i] <- 1
166       }
167     }
168   }
169   currentMat
170 }
171
172 # The average efficiency E of a graph G, where the graph has N vertices and
173 # K edges. This function can also be used to calculate the average
174 # efficiency even if G is a complete graph on N vertices. Such a complete
175 # graph is also denoted G{ideal}, i.e. the ideal case where G has all the
176 # possible
177 #
178 #  $N(N - 1) / 2$ 
179 #
180 # edges. Then the global and local efficiencies are defined in terms of E(G)
181 # and E(G{ideal}). These notions of efficiency of a graph are defined in
182 # the paper (Latora & Marchiori 2003).
183 #

```

```

184 # INPUT:
185 #   geodesicMat -- the matrix of the shortest path lengths between pairs of
186 #   vertices in G. If i and j are vertices of G, then d_ij denotes the
187 #   shortest path length between i and j. If geodesicMat describes the
188 #   geodesics of pairs of vertices in an undirected graph, then
189 #   geodesicMat is symmetric about the main diagonal. Note that
190 #   geodesicMat must be a square matrix, so that its row and column
191 #   dimensions both equal the number of vertices in the underlying
192 #   graph.
193 #
194 # OUTPUT:
195 #   the average efficiency E of the graph G.
196 #
197 # AUTHOR:
198 #   Minh Van Nguyen <nguyenminh2@gmail.com>
199 #
200 averageEfficiency <- function(geodesicMat) {
201   # the number of vertices in geodesicMat
202   N <- dim(geodesicMat)[1]
203   E <- NULL
204
205   # check for the case that geodesicMat is a 1 x 1 matrix
206   if (N == 1) {
207     # harmonicSum <- 0
208     E <- 0
209   }
210   else {
211     # Compute the harmonic sum of lower triangular matrix of geodesicMat,
212     # excluding the main diagonal.
213     colLimit <- 0
214     colStart <- 1
215     rowStart <- 2
216     harmonicSum <- 0
217     for (row in rowStart:N) {
218       colLimit <- colLimit + 1
219       for (col in colStart:colLimit) {
220         # Avoid the case where there's no path between vertices i and j. If
221         # no path exists between i and j, then d_ij = +oo, which is positive
222         # infinity. As d_ij -> +oo, then (1 / d_ij) -> 0.
223         if (geodesicMat[row, col] != "Inf") {
224           harmonicSum <- harmonicSum + (1 / geodesicMat[row, col])
225         }
226       }
227     }
228
229     # compute average efficiency
230     E <- harmonicSum / (N * (N - 1))
231   }
232
233   E
234 }
235
236 # The global efficiency E_glob of a graph G, where the graph has N vertices
237 # and K edges. The notion of global efficiency of a graph is defined in the
238 # paper (Latora & Marchiori 2003). See also the function averageEfficiency,
239 # which defines the average efficiency of G. The measure E_glob is defined
240 # as
241 #
242 #  $E_{glob} = E(G) / E(G^{\{ideal\}})$ 
243 #
244 # where  $G^{\{ideal\}}$  is the complete graph on N vertices. Thus E_glob is a
245 # ratio of the average efficiencies of two types of graphs: (1) the average
246 # efficiency of G itself; (2) the average efficiency of the complete graph
247 # on N vertices, which is the number of vertices in G.
248 #
249 # INPUT:

```

```

250 #     geodesicMat -- the matrix of the shortest path lengths between pairs of
251 #         vertices in G. If i and j are vertices of G, then d_ij denotes the
252 #         shortest path length between i and j. If geodesicMat describes the
253 #         geodesics of pairs of vertices in an undirected graph, then
254 #         geodesicMat is symmetric about the main diagonal.
255 #
256 # OUTPUT:
257 #     the global efficiency E_glob of the graph G.
258 #
259 # AUTHOR:
260 #     Minh Van Nguyen <nguyenminh2@gmail.com>
261 #
262 globalEfficiency <- function(geodesicMat) {
263 # the number of vertices in geodesicMat
264   N <- dim(geodesicMat)[1]
265
266   # compute the average efficiency
267   aveEfficiency <- averageEfficiency(geodesicMat)
268
269   # Construct the adjacency matrix of a complete graph on N vertices. By
270   # definition of complete graphs, a complete graph K_n and its geodesic
271   # matrix G_dist are equivalent. That is, K_n and G_dist are copies of
272   # each other. Also, G_dist has 1 everywhere, and 0 on the main diagonal.
273   gIdealMat <- matrix(nrow = N, ncol = N)
274   gIdealMat[,] <- 1
275   for (i in 1:N) {
276     gIdealMat[i, i] <- 0
277   }
278
279   # compute the average efficiency of the complete graph
280   aveEfficiencyGIdeal <- averageEfficiency(gIdealMat)
281
282   # compute the global efficiency
283   Eglob <- aveEfficiency / aveEfficiencyGIdeal
284 }
285
286 # The adjacency matrix of G_i. If G is an undirected graph and i is a
287 # vertex of G, then G_i is the subgraph of neighbours of i, excluding i
288 # itself.
289 #
290 # INPUT:
291 #     aMat -- the adjacency matrix of the graph G.
292 #     i -- the index of the vertex whose neighbours we want to consider.
293 #         Let r and c be the row and column dimensions of aMat, respectively.
294 #         Then 1 < i < r or 1 < i < c.
295 #
296 # OUTPUT:
297 #     the adjacency matrix of G_i.
298 #
299 # AUTHOR:
300 #     Minh Van Nguyen <nguyenminh2@gmail.com>
301 #
302 neighboursAdjMat <- function(aMat, i) {
303 # the row dimension of aMat
304   ## rowNum <- dim(aMat)[1]
305   # the column dimension of aMat
306   colNum <- dim(aMat)[2]
307
308   # find indices of the immediate neighbours of vertex i
309   neighIndex <- c()
310   for (col in 1:colNum) {
311     if (aMat[i, col] == 1) {
312       neighIndex <- c(neighIndex, col)
313     }
314   }
315 }

```

```

316 # Adjacency matrix of neighbours of i, i.e. the adjacency matrix of G_i
317 # in the notation of the paper (Latora & Marchiori 2003).
318 neighAMat <- matrix(nrow = length(neighIndex),
319                    ncol = length(neighIndex))
320 neighAMat[,] <- 0
321 for (row in 1:length(neighIndex)) {
322   for (col in row:length(neighIndex)) {
323     if (aMat[neighIndex[row], neighIndex[col]] == 1) {
324       neighAMat[row, col] <- 1
325       neighAMat[col, row] <- 1
326     }
327   }
328 }
329
330 neighAMat
331 }
332
333 # The local efficiency E_loc of a graph G, where the graph has N vertices
334 # and K edges. The notion of local efficiency of a graph is defined in the
335 # paper (Latora & Marchiori 2003). See also the function averageEfficiency,
336 # which defines the average efficiency of G. The measure E_loc is defined
337 # as
338 #
339 #  $E_{loc} = (1/N) \sum_{i \in G} E(G_i) / E(G^{\{ideal\}}_i)$ 
340 #
341 # where  $G_i$  is the subgraph of neighbours of vertex  $i$ , and  $G^{\{ideal\}}_i$  is
342 # the complete graph on  $N_i$ , which is the number of vertices in  $G_i$ . Note
343 # that  $G_i$  excludes the vertex  $i$ , and only considers the graph formed by
344 # its immediate neighbours.
345 #
346 # INPUT:
347 #   aMat -- the adjacency matrix of the graph G. If G is an undirected
348 #           graph, then aMat is symmetric about the main diagonal.
349 #
350 # OUTPUT:
351 #   the local efficiency E_loc of the graph G.
352 #
353 # AUTHOR:
354 #   Minh Van Nguyen <nguyenminh2@gmail.com>
355 #
356 localEfficiency <- function(aMat) {
357   # The number of vertices in the underlying graph G. Thus the column and
358   # row dimensions must be equal.
359   N <- dim(aMat)[1]
360
361   # summing the ratios (EGi / EIdealGi) for all vertices i
362   cumSum <- 0 # the cumulative sum
363   EG_i <- 0 # average efficiency of G_i
364   EIdealGi <- 0 # average efficiency of G^{\{ideal\}}_i
365   geodesicGi <- 0 # geodesic matrix of G_i
366   geodesicIdealGi <- 0 # geodesic matrix of G^{\{ideal\}}_i
367   idealGi <- 0 # adjacency matrix of G^{\{ideal\}}_i
368   for (i in 1:N) {
369     geodesicGi <- geodist(neighboursAdjMat(aMat, i))$gdist
370     EG_i <- averageEfficiency(geodesicGi)
371
372     # Construct the adjacency matrix of a complete graph on K_i vertices. By
373     # definition of complete graphs, a complete graph K_n and its geodesic
374     # matrix G_dist are equivalent. That is, K_n and G_dist are copies of
375     # each other. Also, G_dist has 1 everywhere, and 0 on the main diagonal.
376     idealGi <- matrix(nrow = dim(geodesicGi)[1], ncol = dim(geodesicGi)[2])
377     idealGi[,] <- 1
378     for (j in 1:dim(idealGi)[1]) {
379       idealGi[j, j] <- 0
380     }
381

```

```

382 geodesicIdealGi <- geodist(idealGi)$gdist
383 EIdealGi <- averageEfficiency(geodesicIdealGi)
384
385 # Prevent division by zero, which is possible when EIdealGi = 0. If
386 # both EGi and EIdealGi are zero, then we get (0 / 0), which returns
387 # a NaN for "not a number". Caution: we need to consider four cases:
388 #
389 # EGi  EIdealGi
390 # -----
391 # 0    0          <- (EGi / EIdealGi) = 0
392 # 0    y1        <- (EGi / EIdealGi) = 0
393 # x1 0          <- Is it possible to get this case?
394 # x2 y2        <- (EGi / EIdealGi) \in RR\{0}
395 #
396 # where x1, x2, y1, y2 \in RR
397 if ((EGi == 0)) {
398   cumSum <- cumSum + 0
399 }
400 else {
401   cumSum <- cumSum + (EGi / EIdealGi)
402 }
403 }
404
405 Eloc <- cumSum / N
406 }
407
408 # The cost of a network G with N vertices and K edges. For now, we assume
409 # that G is an undirected graph so that its adjacency matrix is symmetric
410 # about the main diagonal. The generalization of the Watts-Strogatz model
411 # contained in (Latora & Marchiori 2003) considers directed as well as
412 # undirected graphs.
413 #
414 # INPUT:
415 #   adjMat -- the adjacency matrix of G. This adjacency matrix must have
416 #             the same dimensions as the matrix of distances of G.
417 #   distMat -- the matrix of distances between pairs of vertices. This
418 #              distance matrix has the same dimensions as the adjacency matrix of
419 #              G.
420 #   gamma -- the cost evaluator function, default is "gamma = WS" for the
421 #            Watts-Strogatz model. TODO: define further models here apart from
422 #            Watts-Strogatz.
423 #
424 # OUTPUT:
425 #   the cost of the network G.
426 #
427 # AUTHOR:
428 #   Minh Van Nguyen <nguyenminh2@gmail.com>
429 #
430 networkCost <- function(adjMat, distMat, gamma = "WS") {
431   netCost <- 0
432   if (gamma == "WS") {
433     N <- dim(adjMat)[1] # the number of vertices
434     K <- 0              # the number of edges
435
436     # For an undirected graph G with adjacency matrix adjMat, both G and
437     # adjMat are symmetric about the main diagonal. Hence we need only
438     # consider either the upper triangular or lower triangular matrices,
439     # excluding entries along the main diagonal, in counting the number of
440     # edges in G. On the other hand, we can also sum the entries in adjMat
441     # and divide the result by 2 to get the number of edges in g.
442     K <- sum(adjMat) / 2
443
444     netCost <- (2 * K) / (N * (N - 1))
445   }
446
447   netCost

```



```

448 }
449
450 # The main routine. This is where the network metrics are calculated. For
451 # the Watts-Strogatz model, the network metrics is comprised of the
452 # characteristic path length L and the clustering coefficient C. As regards
453 # the generalization of Watts-Strogatz contained in the paper
454 # (Latora & Marchiori 2003), the network metrics are the local efficiency
455 # E_loc, the global efficiency E_glob, and the network cost C.
456 #
457 # INPUT:
458 #   regmat -- a regular matrix
459 #   probabilities -- a set of re-wiring probabilities
460 #
461 # OUTPUT:
462 #
463 # AUTHOR:
464 #   Minh Van Nguyen <nguyenminh2@gmail.com>
465 #
466 calculateNetworks <- function(regmat, probabilities) {
467   # A set of adjacency matrices.
468   nets <- array(NA, dim = c(length(probabilities) + 1, n, n))
469
470   # First matrix is the regular matrix.
471   nets[1,,] <- regmat[,]
472
473   # Re-wire with probability p.
474   # Put this in a function.
475   counter <- 1
476   for (p in probabilities) {
477     reWiredMat <- array(0, dim = dim(regmat))
478     while(connectedness(reWiredMat) < 1) {
479       reWiredMat <- reWire(regmat, p)
480       print(c("Re-wiring with probability ", p))
481     }
482     nets[counter + 1,,] <- reWiredMat[,]
483     # plot(network(reWiredMat, directed=FALSE),
484     #       displaylabels=TRUE, mode="circle")
485     # par(ask=TRUE)
486     counter <- counter + 1
487   }
488
489   # This section is for the Latora-Marchiori generalization.
490   # Global and local efficiencies
491   Eglob <- NULL
492   Eloc <- NULL
493   for (counter in 1:(length(probabilities) + 1)) {
494     Eglob[counter] <- globalEfficiency(geodist(nets[counter,,])$gdist)
495     Eloc[counter] <- localEfficiency(nets[counter,,])
496   }
497
498   # structure to return
499   result <- cbind(Eglob[1] / Eglob, Eloc / Eloc[1])
500 }
501
502
503 # =====
504 # 2. MAIN SCRIPT
505 # =====
506
507 # connectivity matrix of a regular network with no loops
508 regularMatrix <- matrix(0, nrow = n, ncol = n)
509 for (i in 1:n) {
510   for (index in (k / 2):1) {
511     # Get right the indexes.
512     jplus <- returnIndex(i + index, n)
513     jminus <- returnIndex(i - index, n)

```

```

514     regularMatrix[i, jplus] <- regularMatrix[i, jminus] <- 1
515   }
516 }
517 dimnames(regularMatrix)[[2]] <- paste("node", (1:n), sep = "-")
518 dimnames(regularMatrix)[[1]] <- paste("node", (1:n), sep = "-")
519
520 # logarithmically-scaled probability vector
521 factor <- (maxProb / minProb)^(1 / (numberOfPoints - 1))
522 probs <- NULL
523 for (pt in 1:numberOfPoints) {
524   probs[pt] <- minProb * factor^(pt - 1)
525 }
526
527 # Call main routine here timesToRepeat times.
528 # variable aliases
529 GE <- 1     # global efficiency
530 LE <- 2     # local efficiency
531
532 # Remember, the first probability is zero, i.e. the regular matrix.
533 results <- array(NA, dim = c(timesToRepeat, length(probs) + 1, 2))
534 dimnames(results)[[3]] <- c("L / LRef", "C / CRef")
535 dimnames(results)[[2]] <- paste("prob", c(0, probs), sep = "-")
536 dimnames(results)[[1]] <- paste("experiment", (1:timesToRepeat), sep = "-")
537 for (experiment in 1:timesToRepeat) {
538   print(c("Executing experiment ", experiment))
539   results[experiment,,] <- calculateNetworks(regularMatrix, probs)
540 }
541
542 # averages
543 GEmeans <- colSums(results[, , GE]) / timesToRepeat
544 LEmeans <- colSums(results[, , LE]) / timesToRepeat
545
546 # Write table of results
547 summary <- cbind(c(0, probs), GEmeans, LEmeans)
548 write.table(format(summary, digits = 6, nsmall = 4, justify = "left"),
549             file = paste(outputDir, summaryFileName, sep = ""), sep = "\t")
550
551 # plot variables
552 xdata <- probs
553 # All except the first value, where prob = 0.
554 ydata <- cbind(GEmeans[-1], LEmeans[-1])
555 matplot(xdata, ydata, log = "x",
556         main = "Global efficiency and local efficiency",
557         xlab = "prob",
558         ylab = "Eglob[ref] / Eglob and Eloc / Eloc[ref]",
559         type = "b")

```

B Latora-Marchiori metrics: weighted

This appendix presents a number of scripts written in Python, R [9] and Sage [10] for computing local and global efficiencies and network cost of weighted, undirected graphs. The graphs in question have $N = 1000$ vertices and per-vertex degree $k = 10$. The Sage script `k_circulant_n.sage` generates 10-circulant graphs on 1000 vertices, each such graph with half the total number of edges removed. Using the R script `rewire-lattices.r`, the generated graphs can be randomly rewired according to the Watts-Strogatz random edge rewiring method. The rewired graphs are output to disk in matrix (plain textual) notation. These graphs can be converted to its R code representation using the Python script `mat2r.py`. Finally, using the script `network-metrics-lm.r`, the global and local efficiencies of the rewired graphs can be calculated according to (5) and (6), respectively, taking into account the case that we are dealing with weighted graphs. The network cost is calculated using (8). The following sections list the contents of the above scripts.

B.1 The script `k_circulant_n.sage`

```
1 # -----
2 # k_circulant_n.sage
3 # Copyright (C) 2009 Minh Van Nguyen <nguyenminh2@gmail.com>
4 #
5 # This Sage script generates ring lattices, each with half the total number
6 # of edges removed. Such graphs can then be rewired as per Watts & Strogatz.
7 # Note that this script does not consider the problem of random edge
8 # rewiring. This script was written and tested using Sage 3.2.x. For more
9 # information about Sage, please visit www.sagemath.org. Before running this
10 # script, make sure that a directory named "networks-half-edges-r" exists
11 # in the current directory.
12 #
13 # REFERENCES:
14 # [1] V. Latora & M. Marchiori. Economic small-world behavior in weighted
15 #     networks. The European Physical Journal B, 32(2):249--263, 2003.
16 #
17 # [2] D. Watts & S.H. Strogatz. Collective dynamics of "small-world"
18 #     networks. Nature, 393(4):440--442, 1998.
19 #
20 # This program is free software; you can redistribute it and/or modify
21 # it under the terms of the GNU General Public License as published by
22 # the Free Software Foundation; either version 2 of the License, or
23 # (at your option) any later version.
24 #
25 # This program is distributed in the hope that it will be useful,
26 # but WITHOUT ANY WARRANTY; without even the implied warranty of
27 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
28 # GNU General Public License for more details.
29 #
30 # You should have received a copy of the GNU General Public License
31 # along with this program; if not, write to the Free Software
32 # Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
33 # -----
34
35
36 def to_r(e, g):
37     """
38     Generates R code representation of the (n, k) ring lattice g.
39
40     The (n, k) ring lattice g is assumed to have half the total number of
41     its edges already removed. The R code matrix representation of g is an
42     adjacency matrix and is written to a text file.
```

```

43
44 INPUT:
45     e -- a positive integer index used for name the R script
46         containing R code matrix representation of g.
47     g -- an (n, k) ring lattice with half of its total number of edges
48         removed.
49
50 OUTPUT:
51     Write the R code matrix representation of g to an R script.
52     """
53     nVertices = g.order()
54     outFile = open("networks-half-edges-r/graph-" + str(e) + ".r", "w")
55     nrow = g.order()
56     ncol = g.order()
57     outFile.write("mat <- matrix(nrow = " + str(nrow)
58                   + ", ncol = " + str(ncol) + ")\n")
59     for i in xrange(nrow):
60         row = str(g.adjacency_matrix()[i])
61         row = "c" + row
62         outFile.write("mat[" + str(i+1) + ",] <- " + row + "\n")
63     outFile.close()
64
65 def remove_half_edges(n, k):
66     """
67     Randomly removes half the total number of edges from a k-circulant graph
68     with n vertices.
69
70     A k-circulant graph with n vertices is simply a ring lattice with n
71     nodes, each of which is connected to its k neighbours. Such a graph
72     is also referred to as an (n, k) ring lattice. Such random removal is
73     used in "Model 4" of Latora & Marchiori [2].
74
75     INPUT:
76         n -- the number of vertices.
77         k -- the number of per-vertex degree (must be an even integer).
78
79     OUTPUT:
80         An (n, k) ring lattice with half of the total number of its edges
81         removed.
82     """
83     from sage.misc.prandom import choice
84
85     adj = [a for a in xrange(1, k/2+1)]
86     G = graphs.CirculantGraph(n, adj)
87
88     # remove half the total number of edges from G
89     nEdges = list(G.edge_iterator(labels = False))
90     elimTotal = G.size() / 2
91     elim = 0
92     while elim < elimTotal:
93         edge = choice(nEdges)
94         G.delete_edge(edge[0], edge[1])
95         while not G.is_connected():
96             G.add_edge(edge[0], edge[1])
97             edge = choice(nEdges)
98             G.delete_edge(edge[0], edge[1])
99         nEdges = list(G.edge_iterator(labels = False))
100        elim += 1
101
102     return G
103
104 # As used by Watts & Strogatz [1] and Latora & Marchiori [2].
105 n = 1000
106
107 # As used by Latora & Marchiori [2] in their "Model 4".
108 k = 6

```

```

109
110 # Also known as the number of rewiring probabilities. This number depends
111 # on how many rewiring probability points you want to use.
112 nTimes = 37
113
114 for i in xrange(nTimes):
115     print "[%s] generating network" % (i+1)
116     g = remove_half_edges(n, k)
117     print "[%s] converting network to R code" % (i+1)
118     to_r(i+1, g)

```

B.2 The script rewire-lattices.r

```

1 # -----
2 # rewire-lattices.r
3 # Copyright (C) 2009 Minh Van Nguyen <nguyenminh2@gmail.com>
4 #
5 # This R script can be used to rewire (n, k) ring lattices that have had
6 # 50 percent of their total number of edges removed. The rewiring process
7 # is per Watts & Strogatz [2]. The resulting rewired networks are used
8 # in "Model 4" of Latora & Marchiori [1]. Before running this script, make
9 # sure that a directory named "networks-dat" exists in the current directory.
10 # For more information about R, please visit www.r-project.org.
11 #
12 # REFERENCES:
13 # [1] V. Latora & M. Marchiori. Economic small-world behavior in weighted
14 #     networks. The European Physical Journal B, 32(2):249--263, 2003.
15 #
16 # [2] D. Watts & S.H. Strogatz. Collective dynamics of "small-world"
17 #     networks. Nature, 393(4):440--442, 1998.
18 #
19 # This program is free software; you can redistribute it and/or modify
20 # it under the terms of the GNU General Public License as published by
21 # the Free Software Foundation; either version 2 of the License, or
22 # (at your option) any later version.
23 #
24 # This program is distributed in the hope that it will be useful,
25 # but WITHOUT ANY WARRANTY; without even the implied warranty of
26 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
27 # GNU General Public License for more details.
28 #
29 # You should have received a copy of the GNU General Public License
30 # along with this program; if not, write to the Free Software
31 # Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA.
32 # -----
33
34
35 # Some housekeeping before generating random Latora-Marchiori networks.
36 # Clear memory, removing (almost) everything in the working environment
37 # without any warning. Be careful with what you wish for.
38 rm(list = ls())
39 # For social network analysis
40 library("sna")
41 # For various graph-theoretic operations, in particular, weighted shortest
42 # paths.
43 library("igraph")
44
45
46 ### Functions
47
48
49 # Simulates a circular list. We are only interested in the index i of a
50 # member of this list, which has n members. One way to conceptualize this
51 # list is to visualize all n members as arranged in a cycle graph, in

```

```

52 # which each member i has an edge connecting it to i + 1, and an edge
53 # connecting it to i - 1. Another way to think about this function is to
54 # interpret it as a simple implementation of the group  $Z/Z_n$ , where only
55 # the index of each i in  $Z/Z_n$  is returned.
56 #
57 # INPUT:
58 #   index -- integer; index of an element in this circular list.
59 #   length -- integer > 0; the number of elements in this circular list.
60 #   FIXME: Maybe it's a good idea to implement the case where
61 #   length < 0, or provide some sanity checking to take care of that
62 #   possibility.
63 #
64 # OUTPUT:
65 #   If  $0 < \text{index} \leq \text{length}$ , then return index. If  $\text{index} > \text{length}$ , then
66 #   return  $\text{index} \bmod \text{length}$ . Else  $\text{index} < 0$ , so return  $\text{index} \bmod \text{length}$ .
67 #   If length is 0, then return NaN (not a number).
68 #
69 # AUTHOR:
70 #   Rodolfo Garcia-Flores
71 #   Documentation by Minh Van Nguyen <nguyenminh2@gmail.com>
72 #
73 returnIndex <- function(index, length) {
74   if ((index > 0) && (index <= length)) {
75     index
76   }
77   else if (index > length) {
78     index - length * floor(index / length)
79   }
80   else {
81     (length + index) + length * floor(-1 * index / length)
82   }
83 }
84
85
86 # The rewiring routine.
87 #
88 # INPUT:
89 #   aMatrix -- an adjacency matrix.
90 #   aProbability -- double; a probability value p such that  $0 < p < 1$ . This
91 #   value determines the probability that an edge incident on a vertex
92 #   is re-wired.
93 #
94 # OUTPUT:
95 #   An adjacency matrix with a number of the vertices re-wired.
96 #
97 # AUTHOR:
98 #   Rodolfo Garcia-Flores
99 #   Documentation by Minh Van Nguyen <nguyenminh2@gmail.com>
100 #
101 rewire <- function(aMatrix, aProbability) {
102   currentMat <- aMatrix
103   for (i in 1:n) {
104     for (j in 1:i) {
105       if ((currentMat[i, j] != 0) && (runif(1) < aProbability)) {
106         # To vertices different to i and
107         # different to those already connected,
108         # preferably to nodes that are isolated.
109         # This should prevent having isolated regions.
110         isolatedNodes <- c(1:n)[colSums(currentMat[,]) == 0]
111         nodesAlreadyConnected <- c(1:n)[currentMat[i,] > 0]
112         excludedNodes <- c(i, nodesAlreadyConnected)
113         notExcludedNodes <- (1:n)[-excludedNodes]
114
115         # A list whose first elements are isolated nodes, the rest are
116         # shuffled, not-excluded values.
117         validPrioritisedNodes <- c(isolatedNodes,

```

```

118         sample(setdiff(notExcludedNodes,
119                         isolatedNodes)))
120
121     # Take first node index in list.
122     newVertex <- validPrioritisedNodes[1]
123     currentMat[i, j] <- 0
124     currentMat[j, i] <- 0
125     currentMat[i, newVertex] <- 1
126     currentMat[newVertex, i] <- 1
127   }
128 }
129 }
130
131 currentMat
132 }
133
134
135 # The main routine. This is where the network metrics are calculated. For
136 # the Watts-Strogatz model, the network metrics is comprised of the
137 # characteristic path length L and the clustering coefficient C. As regards
138 # the generalization of Watts-Strogatz contained in the paper
139 # (Latora & Marchiori 2003), the network metrics are the local efficiency
140 # E_loc, the global efficiency E_glob, and the network cost C_G.
141 #
142 # INPUT:
143 #   regmat -- a regular matrix.
144 #   probabilities -- a set of re-wiring probabilities.
145 #
146 # OUTPUT:
147 #
148 # AUTHOR:
149 #   Minh Van Nguyen <nguyenminh2@gmail.com>
150 #
151 latoraMarchioriGraphs <- function(regMat, probs, experiment) {
152   for (p in 1:length(probs)) {
153     # Reads in an (n, k) ring lattice which has 50 percent of its total
154     # number of edges removed. The matrix is read into memory and named "mat".
155     source(paste("networks-half-edges-r/graph-",
156                 experiment, "-", p, ".r", sep = ""))
157
158     # rewire with p-th probability
159     print(c("rewiring with probability ", probs[p]))
160     rewiredMat <- rewire(mat, probs[p])
161     while (connectedness(rewiredMat) < 1) {
162       rewiredMat <- rewire(regMat, probs[p])
163     }
164     # The number 1000 refers both to the column and row dimensions of the
165     # Latora-Marchiori network.
166     write.table(1000, file = paste("networks-dat/graph-",
167                                   experiment, "-", p, ".dat", sep = ""),
168                row.names = FALSE, col.names = FALSE)
169     write.table(1000, file = paste("networks-dat/graph-",
170                                   experiment, "-", p, ".dat", sep = ""),
171                row.names = FALSE, col.names = FALSE, append = TRUE)
172     write.table(probs[p], file = paste("networks-dat/graph-",
173                                       experiment, "-", p, ".dat", sep = ""),
174                row.names = FALSE, col.names = FALSE, append = TRUE)
175     write.table(rewiredMat, file = paste("networks-dat/graph-",
176                                         experiment, "-", p, ".dat", sep = ""),
177                row.names = FALSE, col.names = FALSE, append = TRUE)
178   }
179 }
180
181
182 ### Start generate Latora-Marchiori networks here
183

```

```

184 # Experimental parameters.
185 # Actual values should be n = 1000, k = 10 or k = 6. When run with these
186 # values, the script should take a few hours to complete. Test values can be
187 # n = 20, k = 4
188 n <- 1000           # vertices
189 k <- 6             # edges per vertex, MUST BE EVEN.
190 nTimes <- 20      # should be 20
191
192 # data for a logarithmically-scaled probability vector
193 numPoints <- 37
194 minProb <- 1e-4
195 maxProb <- 1
196
197
198 # logarithmically-scaled probability vector
199 factor <- (maxProb / minProb)^(1 / (numPoints - 1))
200 probs <- NULL
201 for (pt in 1:numPoints) {
202   probs[pt] <- minProb * factor^(pt - 1)
203 }
204
205 # Create rewired networks. The rewired networks are written to text files.
206 for (experiment in 1:nTimes) {
207   print(c("Executing experiment ", experiment))
208   latoraMarchioriGraphs(matHalfEdges, probs, experiment)
209 }

```

B.3 The script mat2r.py

```

1 # -----
2 # mat2r.py
3 # Copyright (C) 2009 Minh Van Nguyen <nguyenminh2@gmail.com>
4 #
5 # Convert text representation of a Latora-Marchiori network to its R code
6 # representation. This Python script essentially generates R code to
7 # represent Latora-Marchiori networks stored in text files. Latora-Marchiori
8 # networks are (n, k) ring lattices, each with half the total number of its
9 # edges removed and the resulting network rewired as per Watts & Strogatz [2].
10 # Such networks are used in "Model 4" of Latora & Marchiori [1]. Before
11 # running this Python script, make sure that a directory named "networks-r"
12 # exists in the current directory.
13 #
14 # REFERENCES:
15 # [1] V. Latora & M. Marchiori. Economic small-world behavior in weighted
16 #   networks. The European Physical Journal B, 32(2):249--263, 2003.
17 #
18 # [2] D. Watts & S.H. Strogatz. Collective dynamics of "small-world"
19 #   networks. Nature, 393(4):440--442, 1998.
20 #
21 # This program is free software; you can redistribute it and/or modify
22 # it under the terms of the GNU General Public License as published by
23 # the Free Software Foundation; either version 2 of the License, or
24 # (at your option) any later version.
25 #
26 # This program is distributed in the hope that it will be useful,
27 # but WITHOUT ANY WARRANTY; without even the implied warranty of
28 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
29 # GNU General Public License for more details.
30 #
31 # You should have received a copy of the GNU General Public License
32 # along with this program; if not, write to the Free Software
33 # Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
34 # -----
35

```



```

36
37 nExperiments = 20
38 nProbs = 37
39 for e in xrange(1, nExperiments + 1):
40     print "generate R code for networks in experiment %s" % e
41     for p in xrange(1, nProbs + 1):
42         inFile = open("networks-dat/graph-"
43                       + str(e) + "-" + str(p) + ".dat", "r")
44         outFile = open("networks-r/graph-"
45                       + str(e) + "-" + str(p) + ".r", "w")
46         nrow = int(inFile.readline().strip())
47         ncol = int(inFile.readline().strip())
48         rewiringProb = float(inFile.readline().strip()) # don't write to file
49         outFile.write("mat <- matrix(nrow = " + str(nrow)
50                       + ", ncol = " + str(ncol) + ")\n")
51         for i in xrange(1, nrow + 1):
52             row = inFile.readline().strip()
53             row = row.replace(" ", ", ")
54             row = "c(" + row + ")"
55             outFile.write("mat[" + str(i) + ",] <- " + row + "\n")
56         inFile.close()
57         outFile.close()

```

B.4 The script network-metrics-lm.r

```

1 # -----
2 # network-metrics-lm.r
3 # Copyright (C) 2008, 2009 -- Minh Van Nguyen <nguyenminh2@gmail.com>
4 #
5 # An R script to compute network metrics of Latora-Marchiori networks.
6 # That is, this script calculates the local and global efficiencies and
7 # network cost defined by Latora & Marchiori [1] as generalizations of
8 # the Watts-Strogatz [2] small world network metrics.
9 #
10 # REFERENCES:
11 # [1] V. Latora & M. Marchiori. Economic small-world behavior in weighted
12 #     networks. The European Physical Journal B, 32(2):249--263, 2003.
13 #
14 # [2] D. Watts & S.H. Strogatz. Collective dynamics of "small-world"
15 #     networks. Nature, 393(4):440--442, 1998.
16 #
17 # This program is free software; you can redistribute it and/or modify
18 # it under the terms of the GNU General Public License as published by
19 # the Free Software Foundation; either version 2 of the License, or
20 # (at your option) any later version.
21 #
22 # This program is distributed in the hope that it will be useful,
23 # but WITHOUT ANY WARRANTY; without even the implied warranty of
24 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
25 # GNU General Public License for more details.
26 #
27 # You should have received a copy of the GNU General Public License
28 # along with this program; if not, write to the Free Software
29 # Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
30 # -----
31
32
33 # Clear memory, removing (almost) everything in the working environment
34 # without any warning. Be careful with what you wish for.
35 rm(list = ls())
36 library("grDevices") # for colours
37 library("sna")       # for social network analysis
38 # For various graph-theoretic operations, in particular, weighted shortest
39 # paths.

```

```

40 library("igraph")
41
42
43 ### Functions
44
45
46 # The average efficiency E of a graph G, where the graph has N vertices and
47 # K edges. This function can also be used to calculate the average
48 # efficiency even if G is a complete graph on N vertices. Such a complete
49 # graph is also denoted  $G^{\text{ideal}}$ , i.e. the ideal case where G has all the
50 # possible
51 #
52 #  $N(N - 1) / 2$ 
53 #
54 # edges. Then the global and local efficiencies are defined in terms of  $E(G)$ 
55 # and  $E(G^{\text{ideal}})$ . These notions of efficiency of a graph are defined in
56 # the paper (Latora & Marchiori 2003).
57 #
58 # INPUT:
59 #   adjMat -- the adjacency matrix of the underlying graph.
60 #   weightedGeoMat -- a matrix of weighted geodesics, or weighted
61 #   shortest paths.
62 #   level -- whether the average efficiency returned would be used in
63 #   computing the local or global efficiency. The parameter level can
64 #   take on either of two string arguments: "global" to indicate that
65 #   the returned average efficiency is to be used in calculating the
66 #   global efficiency of a network; and "local" which signifies that
67 #   the returned average efficiency is to be used in calculating the
68 #   local efficiency of a network. Default is "global".
69 #
70 # OUTPUT:
71 #   the average efficiency E of the graph G.
72 #
73 # AUTHOR:
74 #   Minh Van Nguyen <nguyenminh2@gmail.com>
75 #
76 averageEfficiency <- function(adjMat, weightedGeoMat) {
77 # The number of vertices in weightedGeoMat. Thus adjMat and weightedGeoMat
78 # are both N x N matrices.
79 N <- dim(weightedGeoMat)[1]
80
81 # for storing the average efficiency
82 E <- NULL
83
84 # check for the case that weightedGeoMat is a 1 x 1 matrix
85 if (N == 1) {
86 # harmonicSum <- 0
87 E <- 0
88 }
89 else {
90 # Compute the harmonic sum of lower triangular matrix of weightedGeoMat,
91 # excluding the main diagonal.
92 collimit <- 0
93 colStart <- 1
94 rowStart <- 2
95 harmonicSum <- 0
96 for (row in rowStart:N) {
97 collimit <- collimit + 1
98 for (col in colStart:collimit) {
99 # Avoid the case where there's no path between vertices i and j. If
100 # no path exists between i and j, then  $d_{ij} = +\infty$ , which is positive
101 # infinity. As  $d_{ij} \rightarrow +\infty$ , then  $(1 / d_{ij}) \rightarrow 0$ . If adjMat
102 # represents a totally isolated graph G, then the average efficiency
103 # of G is 0.
104 if (weightedGeoMat[row, col] != "Inf") {
105 harmonicSum <- harmonicSum + (1 / weightedGeoMat[row, col])

```

```

106     }
107   }
108 }
109
110   # compute average efficiency
111   E <- harmonicSum / (N * (N - 1))
112 }
113
114 E
115 }
116
117
118 # The global efficiency E_glob of a graph G, where the graph has N vertices
119 # and K edges. The notion of global efficiency of a graph is defined in the
120 # paper (Latora & Marchiori 2003). See also the function averageEfficiency,
121 # which defines the average efficiency of G. The measure E_glob is defined
122 # as
123 #
124 #  $E\_glob = E(G) / E(G^{\{ideal\}})$ 
125 #
126 # where  $G^{\{ideal\}}$  is the complete graph on N vertices. Thus E_glob is a
127 # ratio of the average efficiencies of two types of graphs: (1) the average
128 # efficiency of G itself; (2) the average efficiency of the complete graph
129 # on N vertices, which is the number of vertices in G.
130 #
131 # INPUT:
132 #   adjMat -- an N x N adjacency matrix of G.
133 #   weightMat -- an N x N weight matrix of G. See the function weightMatrix
134 #               for further details.
135 #
136 # OUTPUT:
137 #   the global efficiency E_glob of the graph G.
138 #
139 # AUTHOR:
140 #   Minh Van Nguyen <nguyenminh2@gmail.com>
141 #
142 globalEfficiency <- function(adjMat, weightMat) {
143   # The number of vertices in the underlying graph G. Thus the column and
144   # row dimensions must be equal.
145   N <- dim(adjMat)[1]
146
147   # matrix of weighted geodesics for G
148   weightedGeoMat <- weightedGeodesics(adjMat, weightMat)
149   # average efficiency of G
150   aveEfficiency <- averageEfficiency(adjMat, weightedGeoMat)
151
152   # Construct the adjacency matrix of a complete graph on N vertices. By
153   # definition of complete graphs, a complete graph  $K_n$  and its geodesic
154   # matrix  $G\_dist$  are equivalent. That is,  $K_n$  and  $G\_dist$  are copies of
155   # each other. Also,  $G\_dist$  has 1 everywhere, and 0 on the main diagonal.
156   gIdealMat <- matrix(1, nrow = N, ncol = N)
157   for (i in 1:N) {
158     gIdealMat[i, i] <- 0
159   }
160
161   # matrix of weighted geodesics for  $G^{\{ideal\}}$ 
162   weightedGeoMat <- weightedGeodesics(gIdealMat, weightMat)
163   # average efficiency of  $G^{\{ideal\}}$ 
164   aveEfficiencyGIdeal <- averageEfficiency(gIdealMat, weightedGeoMat)
165
166   # compute the global efficiency
167   Eglob <- aveEfficiency / aveEfficiencyGIdeal
168 }
169
170
171 # Let adjMat be an adjacency matrix of an undirected graph G. For a given

```

```

172 # vertex i of G, find the indices of the immediate neighbours of i.
173 #
174 # INPUT:
175 #   adjMat -- an adjacency matrix of an undirected graph G.
176 #   i -- a vertex of G.
177 #
178 # OUTPUT:
179 #   a vector containing vertices that are immediate neighbours of i.
180 #
181 # AUTHOR:
182 #   Minh Van Nguyen <nguyenminh2@gmail.com>
183 #
184 immediateNeighbours <- function(adjMat, i) {
185 # The column dimension of adjMat. As adjMat is an adjacency matrix, it
186 # doesn't matter if we get either of its row or column dimensions.
187   colNum <- dim(adjMat)[2]
188
189   # for storing indices of the immediate neighbours of vertex i
190   neighIndex <- c()
191
192   # find indices of the immediate neighbours of vertex i
193   for (col in 1:colNum) {
194     if (adjMat[i, col] == 1) {
195       neighIndex <- c(neighIndex, col)
196     }
197   }
198
199   neighIndex
200 }
201
202
203 # The adjacency matrix of G_i. If G is an undirected graph and i is a
204 # vertex of G, then G_i is the subgraph of neighbours of i, excluding i
205 # itself.
206 #
207 # INPUT:
208 #   aMat -- the adjacency matrix of the graph G.
209 #   i -- the index of the vertex whose neighbours we want to consider.
210 #       Let r and c be the row and column dimensions of aMat, respectively.
211 #       Then  $1 < i < r$  or  $1 < i < c$ .
212 #
213 # OUTPUT:
214 #   an adjacency matrix of G_i. If i is an isolated vertex, then return
215 #   an n x n zero matrix. If i is not isolated but all vertices in G_i are
216 #   isolated from each other, then return an n x n zero matrix. Else we know
217 #   that i is not isolated and there is a pair of vertices in G_i that
218 #   is connected by an edge; in this case, return an n x n matrix where
219 #    $n > 0$ .
220 #
221 # AUTHOR:
222 #   Minh Van Nguyen <nguyenminh2@gmail.com>
223 #
224 neighboursAdjMat <- function(aMat, i) {
225 # The column dimension of aMat. As aMat is an adjacency matrix, it
226 # doesn't matter if we get either of its row or column dimensions.
227   colNum <- dim(aMat)[2]
228
229   # an adjacency matrix of the neighbours of vertex i
230   neighAMat <- NULL
231
232   # for storing indices of the immediate neighbours of vertex i
233   neighIndex <- NULL
234
235   # check if i is an isolated vertex
236   if (sum(aMat[i, ]) == 0) {
237     # If i is an isolated vertex, then return an n x n zero matrix, which

```

```

238     # is of the same dimensions as those of aMat.
239     neighAMat <- matrix(0, nrow = dim(aMat)[1], ncol = dim(aMat)[2])
240 }
241 # now we know that i is connected to at least another vertex
242 else {
243     # get indices of the immediate neighbours of i
244     neighIndex <- immediateNeighbours(aMat, i)
245
246     # The variables neighRowIndex and neighColIndex should be vectors of
247     # equal length. Let neighRowIndex be of length n, then neighColIndex
248     # also has length n. For k = 1,...,n, neighRowIndex[k] and
249     # neighColIndex[k] refer to vertices that are immediate neighbours of
250     # vertex i, and such that neighRowIndex[k] and neighColIndex[k] are
251     # connected by an (undirected) edge.
252     neighRowIndex <- NULL
253     neighColIndex <- NULL
254     for (row in 1:length(neighIndex)) {
255         for (col in row:length(neighIndex)) {
256             if (aMat[neighIndex[row], neighIndex[col]] == 1) {
257                 neighRowIndex <- c(neighRowIndex, neighIndex[row])
258                 neighColIndex <- c(neighColIndex, neighIndex[col])
259             }
260         }
261     }
262
263     # If i is not an isolated vertex, then the length of the vector
264     # neighIndex is > 0. Let G_i be the subgraph of the neighbours of i. If
265     # all vertices of G_i are isolated, then each of the vectors
266     # neighRowIndex and neighColIndex has a length of zero. In this case,
267     # neighAMat is a 2 x 0 matrix.
268     neighAMat <- matrix(0, nrow = 2, ncol = length(neighRowIndex))
269     neighAMat[1,] <- neighRowIndex
270     neighAMat[2,] <- neighColIndex
271
272     # check if G_i is totally isolated
273     if (dim(neighAMat)[2] == 0) {
274         # If G_i is totally isolated, then return an n x n zero matrix, which
275         # is of the same dimensions as those of aMat.
276         neighAMat <- matrix(0, nrow = dim(aMat)[1], ncol = dim(aMat)[2])
277     }
278     # now we know that at least one pair of vertices in G_i are connected
279     else {
280         neighAdjMat <- matrix(0, nrow = dim(aMat)[1], ncol = dim(aMat)[2])
281         for (col in 1:dim(neighAMat)[2]) {
282             neighAdjMat[neighAMat[1, col], neighAMat[2, col]] <- 1
283             neighAdjMat[neighAMat[2, col], neighAMat[1, col]] <- 1
284         }
285         neighAMat <- neighAdjMat
286     }
287 }
288
289 neighAMat
290 }
291
292
293 # The local efficiency E_loc of a graph G, where the graph has N vertices
294 # and K edges. The notion of local efficiency of a graph is defined in the
295 # paper (Latora & Marchiori 2003). See also the function averageEfficiency,
296 # which defines the average efficiency of G. The measure E_loc is defined
297 # as
298 #
299 #  $E_{loc} = (1/N) \sum_{i \in G} E(G_i) / E(G^{\{ideal\}}_i)$ 
300 #
301 # where G_i is the subgraph of neighbours of vertex i, and G^{\{ideal\}}_i is
302 # the complete graph on N_i, which is the number of vertices in G_i. Note
303 # that G_i excludes the vertex i, and only considers the graph formed by

```

```

304 # its immediate neighbours.
305 #
306 # INPUT:
307 #   aMat -- the adjacency matrix of the graph G. If G is an undirected
308 #          graph, then aMat is symmetric about the main diagonal.
309 #
310 # OUTPUT:
311 #   the local efficiency E_loc of the graph G.
312 #
313 # AUTHOR:
314 #   Minh Van Nguyen <nguyenminh2@gmail.com>
315 #
316 localEfficiency <- function(aMat, weightMat) {
317 # The number of vertices in the underlying graph G. Thus the column and
318 # row dimensions must be equal.
319 N <- dim(aMat)[1]
320
321 # summing the ratios (EGi / EIdealGi) for all vertices i
322 cumSum <- 0 # the cumulative sum
323 EG_i <- 0 # average efficiency of G_i
324 EIdealGi <- 0 # average efficiency of G^{ideal}_i
325 idealGi <- 0 # adjacency matrix of G^{ideal}_i
326 for (i in 1:N) {
327 # adjacency matrix of G_i
328 neighI <- neighboursAdjMat(aMat, i)
329 # matrix of weighted geodesics for G_i
330 weightedGeoMat <- weightedGeodesics(neighI, weightMat)
331 # average efficiency of G_i
332 EG_i <- averageEfficiency(neighI, weightedGeoMat)
333
334 # Construct the adjacency matrix of a complete graph on K_i vertices. By
335 # definition of complete graphs, a complete graph K_n and its geodesic
336 # matrix G_dist are equivalent, provided that K_n is unweighted.
337 idealGi <- matrix(0, nrow = N, ncol = N)
338 neighIndex <- immediateNeighbours(aMat, i)
339 for (j in 1:length(neighIndex)) {
340 for (k in j:length(neighIndex)) {
341 idealGi[neighIndex[j], neighIndex[k]] <- 1
342 idealGi[neighIndex[k], neighIndex[j]] <- 1
343 }
344 # do this since we want zeros along the main diagonal
345 idealGi[neighIndex[j], neighIndex[j]] <- 0
346 }
347
348 # matrix of weighted geodesics for G^{ideal}_i
349 weightedGeoMat <- weightedGeodesics(idealGi, weightMat)
350 # average efficiency of G^{ideal}_i
351 EIdealGi <- averageEfficiency(idealGi, weightedGeoMat)
352
353 # Prevent division by zero, which is possible when EIdealGi = 0. If
354 # both EG_i and EIdealGi are zero, then we get (0 / 0), which returns
355 # a NaN for "not a number". CAUTION: we need to consider four cases:
356 #
357 # EG_i  EIdealGi
358 # -----
359 # 0     0         <- (EG_i / EIdealGi) = 0 because we say so
360 # 0     y1        <- (EG_i / EIdealGi) = 0
361 # x1    0         <- Is it possible to get this case?
362 # x2    y2        <- (EG_i / EIdealGi) \in RR\{0}
363 #
364 # where x1, x2, y1, y2 \in RR are non-zero and RR is the set of
365 # real numbers.
366 if (EG_i == 0) {
367 cumSum <- cumSum + 0
368 }
369 else {

```

```

370     cumSum <- cumSum + (EGi / EIdealGi)
371   }
372 }
373
374 Eloc <- cumSum / N
375 }
376
377
378 # The cost of a network G with N vertices and K edges. For now, we assume
379 # that G is an undirected graph so that its adjacency matrix is symmetric
380 # about the main diagonal. The generalization of the Watts-Strogatz model
381 # contained in (Latora & Marchiori 2003) considers directed as well as
382 # undirected graphs.
383 #
384 # INPUT:
385 #   adjMat -- the adjacency matrix of G. This adjacency matrix must have
386 #             the same dimensions as the matrix of distances of G.
387 #   distMat -- the matrix of distances between pairs of vertices. This
388 #              distance matrix has the same dimensions as the adjacency matrix of
389 #              G.
390 #
391 # OUTPUT:
392 #   the cost of the network G.
393 #
394 # AUTHOR:
395 #   Minh Van Nguyen <nguyenminh2@gmail.com>
396 #
397 networkCost <- function(adjMat, distMat) {
398   netCost <- 0
399   numerator <- 0
400   denominator <- 0
401   N <- dim(adjMat)[1]
402
403   for (row in 2:N) {
404     for (col in 1:(row - 1)) {
405       numerator <- numerator + (adjMat[row, col] * distMat[row, col])
406       denominator <- denominator + (distMat[row, col])
407     }
408   }
409
410   netCost <- numerator / denominator
411 }
412
413
414 # The weight matrix of a ring lattice G that has N vertices. This weight is
415 # defined in terms of the Euclidean distance between pairs of nodes. If i
416 # and j are vertices of G, then the distance between i and j is
417 #
418 #  $l_{ij} = [2 * \sin(|i - j| \pi / N)] / [2 * \sin(\pi / N)]$ 
419 #  $= \sin(|i - j| \pi / N) / \sin(\pi / N)$ 
420 #
421 # The distance between each pair of neighbouring vertices is  $l_{ij} = 1$  and
422 # the distance from i to itself is trivially  $l_{ii} = 0$ . The weight matrix
423 # of G is denoted  $\{l_{ij}\}$ , which has zero along the main diagonal and is
424 # symmetric about this diagonal.
425 #
426 # INPUT:
427 #   n -- an integer > 0; this is the number of vertices of the ring
428 #        lattice G
429 #
430 # OUTPUT:
431 #   the weight matrix  $\{l_{ij}\}$  of G. If  $n \leq 0$ , then return NULL.
432 #
433 # AUTHOR:
434 #   Minh Van Nguyen <nguyenminh2@gmail.com>
435 #

```

```

436 weightMatrix <- function(n) {
437   weightMat <- NULL
438
439   if (n > 0) {
440     # construct an n x n matrix with zero everywhere
441     weightMat <- matrix(0, nrow = n, ncol = n)
442
443     # Calculate the Euclidean distances on the ring lattice. Perhaps we
444     # need only to consider either of the lower triangular or upper
445     # triangular matrices, excluding the main diagonal.
446     for (row in 1:n) {
447       for (col in 1:n) {
448         if (row != col) {
449           # numerator <- 2 * sin((abs(row - col) * pi) / n)
450           # denominator <- 2 * sin(pi / n)
451           numerator <- sin((abs(row - col) * pi) / n)
452           denominator <- sin(pi / n)
453           weightMat[row, col] <- numerator / denominator
454         }
455       }
456     }
457   } else {
458     weightMat <- NULL
459   }
460
461   weightMat
462 }
463
464
465 # A matrix of weighted shortest paths for a weighted ring lattice G. The
466 # lattice G has N vertices and a degree of k per vertex.
467 #
468 # INPUT:
469 #   adjMat -- the adjacency matrix of G. If G is undirected, then adjMat
470 #   is symmetric about the main diagonal. The adjacency matrix of G
471 #   must have the same dimensions as the weight matrix of G.
472 #   weightMat -- a matrix of edge weights. This is an N x N matrix,
473 #   where N is the number of vertices in G. If G is undirected, then
474 #   weightMat is symmetric about the main diagonal. The adjacency
475 #   matrix of G must have the same dimensions as the weight matrix of G.
476 #
477 # OUTPUT:
478 #   an N x N matrix of weighted shortest paths. If G is totally isolated,
479 #   then return an N x N matrix with +oo everywhere, and zero along the
480 #   main diagonal. Else G has an (undirected) edge connecting a pair of
481 #   its vertices, so we return an N x N matrix of weighted shortest paths.
482 #
483 # AUTHOR:
484 #   Minh Van Nguyen <nguyenminh2@gmail.com>
485 #
486 weightedGeodesics <- function(adjMat, weightMat) {
487   N <- dim(adjMat)[1]
488   weightedGeo <- NULL
489
490   # Check for totally isolated graphs. The graph G represented by adjMat is
491   # totally isolated if all its vertices are isolated from each other. For
492   # a totally isolated graph G of dimensions N x N, its corresponding
493   # matrix of weighted geodesics is an N x N matrix with +oo (positive
494   # infinity) everywhere, and zero along the main diagonal.
495   if (sum(adjMat) == 0) {
496     weightedGeo <- matrix(Inf, nrow = N, ncol = N)
497     for (i in 1:N) {
498       weightedGeo[i, i] <- 0
499     }
500   }
501   # Now we know that G is not totally isolated, so at least one pair of

```



```

502 # vertices in G is connected by an (undirected) edge. Then proceed to
503 # find the matrix of weighted geodesics corresponding to G.
504 else {
505     collimit <- 0
506     colStart <- 1
507     rowStart <- 2
508     startVertex <- c()
509     endVertex <- c()
510     edgeWeight <- c()
511
512     # As weightMat is symmetric about the main diagonal, we only need to
513     # consider its lower (or upper) triangular matrix, excluding entries
514     # along the main diagonal.
515     for (row in rowStart:N) {
516         collimit <- collimit + 1
517         for (col in colStart:collimit) {
518             if (adjMat[row, col] == 1) {
519                 startVertex <- c(startVertex, row)
520                 endVertex <- c(endVertex, col)
521                 edgeWeight <- c(edgeWeight, weightMat[row, col])
522             }
523         }
524     }
525
526     e <- c()
527     for (i in 1:length(startVertex)) {
528         e <- c(e, startVertex[i], endVertex[i], edgeWeight[i])
529     }
530     emat <- matrix(nc = 3, byrow = TRUE, e)
531     for (row in 1:dim(emat)[1]) {
532         emat[row, 1] <- emat[row, 1] - 1
533         emat[row, 2] <- emat[row, 2] - 1
534     }
535     g <- add.edges(graph.empty(N, directed = FALSE),
536                   t(emat[, 1:2]), weight = emat[, 3])
537     weightedGeo <- shortest.paths(g)
538 }
539
540 weightedGeo
541 }
542
543
544 # The main routine. This is where the network metrics are calculated. For
545 # the Watts-Strogatz model, the network metrics is comprised of the
546 # characteristic path length L and the clustering coefficient C. As regards
547 # the generalization of Watts-Strogatz contained in the paper
548 # (Latora & Marchiori 2003), the network metrics are the local efficiency
549 # E_loc, the global efficiency E_glob, and the network cost C_G.
550 #
551 # INPUT:
552 #   regmat -- a regular matrix.
553 #   probabilities -- a set of re-wiring probabilities.
554 #   weightMat -- a matrix of edge weights.
555 #   experiment -- n-th experiment
556 #
557 # OUTPUT:
558 #   Network metrics using the measures described in Latora & Marchiori [1].
559 #
560 # AUTHOR:
561 #   Minh Van Nguyen <nguyenminh2@gmail.com>
562 #
563 calculateNetworks <- function(regmat, probabilities, weightMat, experiment) {
564     # set of adjacency matrices
565     nets <- array(NA, dim = c(length(probabilities) + 1, n, n))
566     # first matrix is regular matrix
567     nets[1,,] <- regmat[,]

```

```

568
569 # read in rewired networks for the specified experiment number
570 print("read in rewired networks")
571 counter <- 1
572 for (p in 1:length(probabilities)) {
573   # Read in rewired network into memory and the resulting object is named
574   # "mat".
575   source(paste("networks-r/graph-", experiment, "-", p, ".r", sep = ""))
576   nets[counter + 1,,] <- mat[,]
577   counter <- counter + 1
578 }
579
580 # This section is for the Latora-Marchiori generalization.
581 # Global and local efficiencies, and network cost
582 Eglob <- NULL
583 Eloc <- NULL
584 netCost <- NULL
585 for (counter in 1:(length(probabilities) + 1)) {
586   print(c("metrics for n-th rewiring probability ", counter))
587   Eglob[counter] <- globalEfficiency(nets[counter,,], weightMat)
588   Eloc[counter] <- localEfficiency(nets[counter,,], weightMat)
589   netCost[counter] <- networkCost(nets[counter,,],
590                                   weightedGeodesics(nets[counter,,],
591                                                         weightMat))
592 }
593
594 # the structure to return
595 result <- cbind(Eglob[1] / Eglob,
596                Eloc / Eloc[1],
597                netCost / netCost[1])
598 }
599
600
601 # Simulates a circular list. We are only interested in the index i of a
602 # member of this list, which has n members. One way to conceptualize this
603 # list is to visualize all n members as arranged in a cycle graph, in
604 # which each member i has an edge connecting it to i + 1, and an edge
605 # connecting it to i - 1. Another way to think about this function is to
606 # interpret it as a simple implementation of the group  $Z/Z_n$ , where only
607 # the index of each i in  $Z/Z_n$  is returned.
608 #
609 # INPUT:
610 #   index -- integer; index of an element in this circular list.
611 #   length -- integer > 0; the number of elements in this circular list.
612 #   FIXME: Maybe it's a good idea to implement the case where
613 #   length < 0, or provide some sanity checking to take care of that
614 #   possibility.
615 #
616 # OUTPUT:
617 #   If  $0 < \text{index} \leq \text{length}$ , then return index. If  $\text{index} > \text{length}$ , then
618 #   return  $\text{index} \bmod \text{length}$ . Else  $\text{index} < 0$ , so return  $\text{index} \bmod \text{length}$ .
619 #   If length is 0, then return NaN (not a number).
620 #
621 # AUTHOR:
622 #   Rodolfo Garcia-Flores
623 #   Documentation by Minh Van Nguyen <nguyenminh2@gmail.com>
624 #
625 returnIndex <- function(index, length) {
626   if ((index > 0) && (index <= length)) {
627     index
628   }
629   else if (index > length) {
630     index - length * floor(index / length)
631   }
632   else {
633     (length + index) + length * floor(-1 * index / length)

```

```

634 }
635 }
636
637
638 ### Main script
639
640
641 # Actual values should be n = 1000, k = 10. When run with these values,
642 # the script should take a few hours to complete. Test values can be
643 # n = 20, k = 4
644 n <- 1000           # vertices
645 k <- 6             # edges per vertex, MUST BE EVEN.
646 nTimes <- 20      # should be 20
647
648 # data for a logarithmically-scaled probability vector
649 nPoints <- 37
650 minProb <- 1e-4
651 maxProb <- 1
652
653 # output file name
654 summaryFileName <- "small-world-summary.txt"
655
656 # connectivity matrix of a regular network with no loops
657 regularMatrix <- matrix(0, nrow = n, ncol = n)
658 for (i in 1:n) {
659   for (index in (k/2):1) {
660     # Get right the indexes.
661     jplus <- returnIndex(i + index, n)
662     jminus <- returnIndex(i - index, n)
663     regularMatrix[i, jplus] <- regularMatrix[i, jminus] <- 1
664   }
665 }
666
667 # logarithmically-scaled probability vector
668 factor <- (maxProb / minProb)^(1 / (nPoints - 1))
669 probs <- NULL
670 for (pt in 1:nPoints) {
671   probs[pt] <- minProb * factor^(pt - 1)
672 }
673
674 # variable aliases
675 numMeasures <- 3 # how many measures
676 GE <- 1          # global efficiency
677 LE <- 2          # local efficiency
678 NC <- 3          # network cost
679
680 # weight matrix of the ring lattice
681 weightMat <- weightMatrix(n)
682
683 # first probability is zero, i.e. the regular matrix
684 results <- array(NA, dim = c(nTimes, length(probs) + 1, numMeasures))
685
686 for (experiment in 1:4) {
687   print(c("network metrics for experiment ", experiment))
688   results[experiment, ,] <- calculateNetworks(regularMatrix,
689                                               probs,
690                                               weightMat,
691                                               experiment)
692 }
693
694 # averages
695 GEmeans <- colSums(results[, , GE]) / nTimes
696 LEmeans <- colSums(results[, , LE]) / nTimes
697 NCmeans <- colSums(results[, , NC]) / nTimes
698
699 # write table of results

```

```
700 summary <- cbind(c(0, probs), GMeans, LMeans, NCmeans)
701 write.table(summary, file = summaryFileName, sep = "\t")
702
703 # plot variables
704 # insert plotting code here
```

References

- [1] B. Bollobás. *Random Graphs*. Cambridge University Press, 2nd edition, 2001.
- [2] R. D. Castro and J. W. Grossman. Famous trails to Paul Erdős. *Mathematical Intelligencer*, 21:51–63, 1999.
- [3] P. Erdős and A. Rényi. On random graphs. *Publicationes Mathematicae*, 6:290–297, 1959.
- [4] A. A. Hagberg, D. A. Schult, and P. J. Swart. Exploring network structure, dynamics, and functions using NetworkX. In G. Varoquaux, T. Vaught, and J. Millman, editors, *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA, USA, 2008. <http://networkx.lanl.gov>.
- [5] M. D. Intriligator. *Mathematical Optimization and Economic Theory*. Prentice-Hall, Englewood Cliffs, NJ, USA, 1971.
- [6] T. Kaihara. Multi-agent based supply chain modelling with dynamic environment. *International Journal of Production Economics*, 85:263–269, 2003.
- [7] V. Latora and M. Marchiori. Economic small-world behavior in weighted networks. *The European Physical Journal B*, 32(2):249–263, 2003.
- [8] S. Milgram. The small world problem. *Psychology Today*, 2:60–67, 1967.
- [9] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2009. ISBN 3-900051-07-0, <http://www.r-project.org>.
- [10] W. A. Stein et al. *Sage Mathematics Software (Version 3.2.3)*. The Sage Development Team, 2009. <http://www.sagemath.org>.
- [11] D. J. Watts and S. H. Strogatz. Collective dynamics of ‘small-world’ networks. *Nature*, 393:440–442, 1998.