Text S1: Pseudocode for randomization algorithm.

Link Permutation algorithm

The notation |*S*| is the cardinality (or size) of the set *S*.

Pseudo-code for Link Permutation

```
1.  Begin
2.      Create vector linkSet and add all links from network to linkSet
3.      Create empty vector testedLinks
4.      while (|linkSet| >= 2  OR  |testedLinks| < |linkSet|):
5.          Pick a random link A-B from linkSet
6.          Pick a random link C-D from linkSet
7.          if (A-D and C-B not in network):
8.              Swap A-B and C-D to A-D and C-B
9.              Remove A-B and C-D from linkSet
10.         else if (A-C and B-D not in network)
11.             Swap A-B and C-D to A-C and B-D
12.             Remove A-B and C-D from linkSet
13.         else:
14.             Add link pair to testedLinks
15.     end while
16. End
```

Node Permutation algorithm

The *degNodesMap* is a map from an integer to a vector of nodes that have the same equivalence class established by the by the equivalence relation: $B[d] = \text{round}(\ln(d)+1)$, where d is each node's degree. A log scale is used so that scale free networks (such as observed in biological networks) will have approximately the same number of nodes in each bin. The notation *R.degree* is the degree of node *R* in the original network.

Pseudo-code for Node Permutation

```
1.  Begin
2.      Create mapping of nodes from binned degree: degNodesMap
3.      Create set of nodes paired with degree in original network: recordSet
4.      Randomly shuffle recordSet
5.      for (R in recordSet):
6.          Pick a random node K from degNodesMap[B[R.degree]]
7.          Swap the labels of R and K
8.      end for
9.  End
```

Link Assignment algorithm

Test condition for adding a link between *K* and *R*: *R* is not equal to *K*, link *R*-*K* is not already in the network, *R* AND *K* have not recovered their original degree yet. The notation *deg*(*K*) is the current degree of node *K* in the *network*. The notation *K.degree* is the degree of node *K* in the original network.

Pseudo-code for Link Assignment

```
1.  Begin
2.      Create set of nodes paired with degree in original network: recordSet
3.      Randomly shuffle recordSet
4.      Remove all links from network
5.      for (k = 1 to |recordSet|):
6.          K = recordSet[k]
7.          Generate list of indices of recordSet: recordIndexSet
8.          while (deg(K) < K.degree OR  |recordIndexSet| == 0 OR |recordSet| == 0 ):
9.              Pick a random index from recordIndexSet
10.             R = recordSet[index]
11.             if (K and R can be linked in the network):
12.                 Create R-K in network
13.                 if (deg(K) ==  K.degree):
14.                     Remove K from recordSet
15.                     k = 1
16.                     break while
17.                 if (deg(R) ==  R.degree):
18.                     Remove R from recordSet
19.                     k = 1
20.                     break while
21.             else:
22.                 Remove index from recordIndexSet
23.         end while
24.     end for
25. End
```

Link Assignment + Second-order modification

The algorithm for link assignment + second order is similar to the link assignment algorithm, but instead of choosing a random node from the whole network (represented by *recordSet*) for node *K* to connect to, it picks from a set of nodes that fall into the same log degree bin [established by the equivalence relation: $B[d] = \text{round}(\ln(d)+1)$], as the original neighbors of *K*. For example, say that node *K* was linked to a set of nodes that have the following degree sequence: {1, 30, 100, 400}. This can be termed the neighbor degree sequence of node *K*. By choosing nodes from the same *binned* neighbor degree sequence, the algorithm assures that node *K* has approximately the same neighbor degree sequence before and after randomization.

Fixing Degree Sequence Errors

For the two link assignment methods, a few nodes may not be able to conserve their degree due to conflicts in the test cases once the network becomes more densely connected. In that case the following procedure can be applied:

Pseudo-code for Fixing Degree Sequence Errors

```
1.  Begin
2.      #Fix odd errors first
3.      Create a set of nodes that did not conserve degree: errNodes
4.      Sort errNodes in increasing order of deltaDeg(N) = N.degree - deg(N)
5.      for (N1 in errNodes):
6.          if (deltaDeg(N1) odd):
7.              N2 = next node in errNodes with deltaDeg(N2) odd
8.              for (link L1-L2 in network):
9.                  if (testCase1):
10.                     Remove link L1-L2 from network
11.                     Create links (either form N1-L1 and N2-L2 or N1-L2 and N2-L1)
12.                     break for
13.             end for
14.     end for
15.     #Fix even errors next (errNodes should only have even errors at this point)
16.     for (N1 in errNodes):
17.         for (k=0 to deltaDeg(N1)/2):
18.             for (link L1-L2 in network):
19.                 if (testCase2):
20.                     Remove link L1-L2 from network
21.                     Create links N1-L1 and N1-L2
22.             end for
23.         end for
24.     end for
25. End
```

The following are the test cases used above.
testCase1: the nodes *L1* OR *L2* have an odd degree, AND [the nodes *L1* AND *L2* are not nodes *N1* OR *N2*,
AND [the network doesn't already have links *N1-L1* AND *N2-L2* OR links *N1-L2* AND *N2-L1*]].
testCase2: the nodes *L1* AND *L2* are not N1, AND the network doesn't already have links *N1-L1* AND *N1-L2*.