ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecture with Computer Exercises:
Modelling and Simulating Social Systems with MATLAB

Project Report

Network Resilience of Interdependent Networks

Thierry Backes, Sichen Li, & Peng Zhou

Zürich
December 2016

Agreement for free-download

We hereby agree to make our source code for this project freely available for download from the web pages of the SOMS chair. Furthermore, we assure that all source code is written by ourselves and is not violating any copyright restrictions.

Thierry Backes Sichen Li Peng Zhou

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Network Resilience of Interdependent Networks

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Li	Sichen
Zhou	Peng
Backes	Thierry

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date

18.12.16 , Zurich

Signature(s)

Backes Thierry

Zhou Peng

Li Sichen

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

Abstract

In this project, we analyze the influence of node failure on a one-to-one interconnected network and the resulting propagation of failure. We simulate this cascade of failures using the NetworkX package [1] as a Python framework. We run the experiment on different network types, with different number of nodes to find a percolation phase transition, when this fraction reaches a threshold. We expand to model and study different connection schemes between two sub-networks, to find the best method to fully interconnect two initially disjoint networks.

Contents

[1] Available at `https://networkx.github.io/`

1 Individual contributions

Everyone contributed to writing the code and the report. Specifically, P.Z built the model, S.L. expanded the model to different types of networks, and T.B. expanded the model to different mapping schemes.

2 Introduction and Motivations

Networks are one of the most common mathematical models to mimic real-life situations, e.g. power grids, the Internet, social networks, transportation systems, etc. Due to the ever growing connectivity, a failure of a certain fraction may lead to a cascading and fatal failure of the entire network.

We analyze the network resilience, i.e. the stability of the network when it is subject to a cascade of failures. This scenario becomes more complicated and interesting, when we consider not only single network, but interconnected networks.

We ask ourselves, what the best strategy is to connect to separate networks together, to lower the risk of fracturing after a cascade of failure.

3 Description of the Model

The core of the application is the cascading model. This determines, how a failure can spread (cascade) through the network and sets restrictions and conditions to the components of the network. We use the cascading failure model proposed in [1].

The model requires two similar undirected sub-networks A and B. The two sub-networks have exactly the same number of nodes, and are generated using the same initial parameters. In our implementation we use an average degree $\langle k_A \rangle$, $\langle k_B \rangle$ of 4. To create our final interconnected graph G, we have a *strong one-to-one mapping*. This means, that every node in network A is connected to one and only one node in network B, such that every node has a direct link to the other network:

$$\forall v_a \in A, \exists! v_b \in B \mid \{v_a, v_b\} \in G \tag{1}$$

A failing node in the network G means that the node, and it's corresponding edges are all deleted. The strong one-to-one mapping enforces a dependency on a node and it's counterpart in the other sub-network. When a node in a sub-network fails, i.e. gets deleted, then the corresponding node, uniquely identifiable by the edge $\{v_a, v_b\}$, fails as well. The propagation model is the important part, which differentiates the attack of one, or several, nodes from a *static* failure model. After an initial attack of p randomly selected nodes (deleting $2p$ nodes due to the one-to-one dependency), the propagation model runs on the network G.

3.1 Propagation model

We use the terms network A and network B to identify the different components in the network G, and do not talk about the separate networks (without mapping).

The Model: In STEP 1 we remove for every node in network B, every internal edge (not considering the one-to-one mapping) in the network B , which connects to a separate cluster in the network in A. STEP 2 does the same process for each node in network A, and removes every internal edge, which connects to a separate B cluster. This steps are repeated until no further edges can be removed.

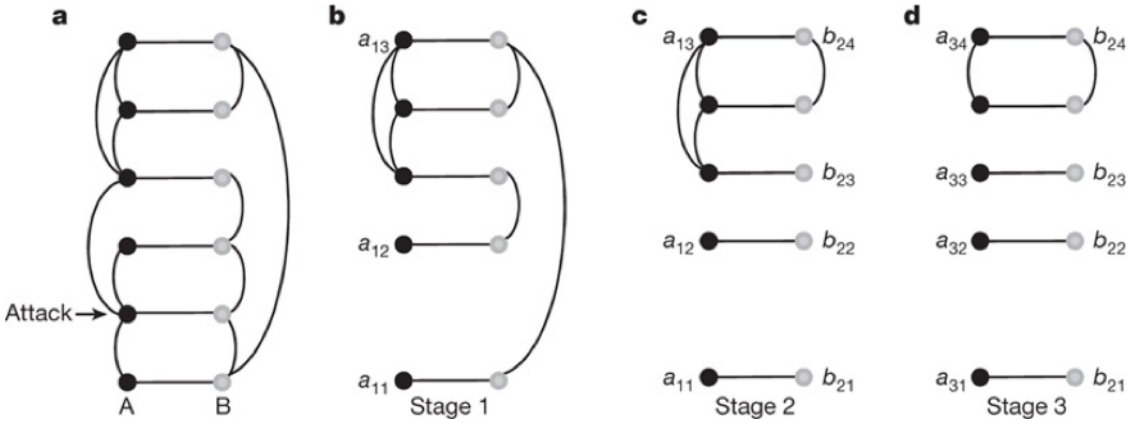

Figure 1: Basic model of cascading failure in interdependent network. The two sub-networks are of same sizes and have a one-to-one correspondence. In **a** we have an initial failure in the sub-network A. In **b** we remove these nodes and their corresponding nodes in B, and all the edges terminating at them. Then sub-networks A and B form into different clusters. In step **c** we focus on B clusters, and remove all those edges in B, whose start and end nodes do not link to the same clusters in A. We do this recursively, until we reach a situation where A clusters and B clusters are mutually connected, as in **d**. This figure is from [1]

Clustering: For an edge $\{v_{b1}, v_{b2}\} \in B$, linking $v_{b1} \in B$ to $v_{b2} \in B$ and the corresponding nodes $v_{a1} \in A$ and $v_{a2} \in A$, we call those four nodes mutually connected, if and only if, there exists an edge $\{v_{a1}, v_{a2}\} \in A$. Otherwise, the nodes v_{b1} and v_{b1} connect to different clusters in network A. With the removal of edges, each sub-network (might) fragment into separate clusters, separate connected components. We are interested in the size of the largest mutually connected cluster, $LMCC$, in the network G after this propagation model.

3.2 Mapping Schemes

As stated above, the network has a one-to-one mapping. We have four different schemes to generate a list of target nodes, and two connection strategies, namely high-to-high and high-to-low. This means, we generate a list of nodes from sub-network A, and a separate list from sub-network B. We then connect the two sub-networks either with the highest value to the highest value, or the highest value to the lowest value. In addition to the random mapping, we tried to chose strategies which link *important* nodes together.

Random Mapping Random mapping is the proposed way of the model from the paper. This does not privilege any node and the two sub-networks are connected randomly, making sure that every node has exactly one counterpart.

Closeness Centrality The closeness centrality of a given node v in a graph is the sum of the shortest paths between v and all the other nodes in the network. A node with a higher closeness centrality has therefore a *higher importance* in the graph.

$$C_c(v) = \frac{N-1}{\sum_w d(v,w)} \tag{2}$$

Where N is the total number of nodes in the graph, and $d(v,w)$ is the distance function from node v to w, based on the implementation of NetworkX [2].

Betweenness Centrality The betweenness centrality of a node v is the number of shortest paths from every other node to every other node, passing trough this specific node. Therefore, a node with a higher betweenness centrality is *more important* in a network.

$$C_b(v) = \sum_{s,v \in V} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{3}$$

Where σ_{st} is the number of shortest paths from s to t,and $\sigma_{st}(v)$ is the number of shortest paths from s to t. passing trough the node v. In our case, if $s = t$, $\sigma_{s,t} = 1$ and if $v \in s, t$ then $\sigma_{s,t}(v) = 0$ based on the implementation of NetworkX [3].

Degree The degree of a given node v is the number of edges it has. The higher the degree, the more nodes are connected directly to it, resulting in a (potentially) more severe outcome if it fails.

[2] `https://networkx.github.io/documentation/development/reference/generated/networkx.algorithms.centrality.closeness_centrality.html`

[3] `https://networkx.github.io/documentation/development/reference/generated/networkx.algorithms.centrality.betweenness_centrality.html`

3.3 Measurement

For a graph G we randomly remove a fraction of $1-p$ randomly selected nodes and their counterparts. We suppose that only the LMCC stays functional after the initial failure and the resulting cascade. Therefore, we measure the portion of remaining nodes in the LMCC. We use the same initial graph to remove different fraction of nodes, to find the transition from a functional graph to a clustered graph. For a given $1-p$, we repeat the experiment several times for statistical significance.

3.4 Graph Types

We focus on three different types of random graphs.

Erős-Rényi A graph $G(n,p)$ of n nodes is constructed by randomly adding edges. For each pairs of nodes (v,w), there is a probability p of having an edge $\{v,w\}$. [4]

Random regular network In a k-regular graph, every node has the same number of neighbors (the same degree); namely k. A random k-regular graph is simply a random selection from all possible k-regular graphs for a given k.[5][6]

Scale free network A scale-free network has degree distribution that follows a power law. The fraction $P(k)$ of nodes in the network having degree a of k follows for large values of k:

$$P(k) \sim k^{-\lambda} \tag{4}$$

where λ is a parameter typically in the range $2 < \lambda < 3$. This results in several nodes with a really high degree, whereas most nodes have a degree of 1 or 2. The high-degree nodes, or hubs, can be seen as *having a specific purpose*.[7] Removing one of those hubs, may certainly result in serious clustering, compared to removing random nodes.

4 Implementation

We are using the python library NetworkX [8] for our model. This gives us the advantage to rely on previously created, reviewed, and efficient code.

[4] https://jeremykun.com/2013/08/22/the-erdos-renyi-random-graph/
[5] `https://www.math.cmu.edu/~af1p/MAA2005/L5.pdf`
[6] `https://en.wikipedia.org/wiki/Random_regular_graph`
[7] `https://en.wikipedia.org/wiki/Scale-free_network`
[8] `https://networkx.github.io`

Simulation: The main function starts by generating the selected network G and storing it. We then continue with the loop for the different p values, i.e., the fraction of nodes to be removed before the propagation model runs. For each p value, we run the propagation model, as well as the node removal i times on a copy of the network G. The number i is chosen according to the time complexity of this particular combination, and varies heavily on the graph type, the number of nodes, and the connection scheme.

For every combination of graph type, and number of nodes, we get a seperate file: *N{#nodes}_kavg{value}_rep{#repetitions}_choice{RR or ER}_type{Centrylity, Degree, Random}_order{High to High OR High to Low}.dat* The data can be found in our github repository. [9]

4.1 Graph creation

For the Erős-Rényi graph and the Random Regular graph, we're using the NetworkX functions FAST_GNP_RANDOM_GRAPH [10] and RANDOM_REGULAR_GRAPH. [11] For the scale free network, we generate a degree sequence using POWERLAW_SEQUENCE(N, LAM) [12] and then use the function CONFIGURATION_MODEL [13]to generate the graph. This function returns a graph, which might contain parallel edges and self loops. To have a scale free network close to the optimal solution, we added a threshold to only accept networks with less than 0.01% of self-loops. Additionally, as stated on the documentation, for the larger the number of nodes, the closer the graph is to a perfect scale-free network. We therefore try to only use large scale-free networks.

4.2 Mapping

For the mapping we generate a list for each sub-network, having as key the node, and as value the chosen metric (such as degree, or centrality). The entry n of the list has the form of $[node : value]$. The lists are then sorted ascending based on the value, such that an index n holds the $n-$th node. The lists are generated using the following implementations provided by NetworkX:

[9] https://github.com/pzhou137/network_resilience

[10] https://networkx.github.io/documentation/development/reference/generated/networkx.generators.random_graphs.fast_gnp_random_graph.html

[11] https://networkx.github.io/documentation/development/reference/generated/networkx.generators.random_graphs.random_regular_graph.html

[12] https://networkx.github.io/documentation/networkx-1.9/reference/generated/networkx.utils.random_sequence.powerlaw_sequence.html

[13] https://networkx.github.io/documentation/networkx-1.9/reference/generated/networkx.generators.degree_seq.directed_configuration_model.html

Closeness NX.CLOSENESS_CENTRALITY(GRAPH,NORMALIZED=TRUE) [14]

Betweenness NX.BETWEENNESS_CENTRALITY(GRAPH,NORMALIZED=TRUE) [15]

Degree G.DEGREE() [16]

For the high-to-high order, we map the nodes at index n of both lists. The high-to-low matches the element at index n and $N - n$, where N is the total number of nodes in a given sub-network (note that both sub-networks have the same number of nodes). For the random mapping, we simply fill the lists randomly with unique numbers and can therefore use the same code for all the mapping schemes.

4.3 Propagation Model

Initial Random Failure The propagation model starts by generating the lists of corresponding nodes for both sub-networks, and then removing a fraction $1 - p$ of random nodes. We remove those nodes (with all their edges) from sub-network A, and remove the corresponding nodes in sub-network B, taking the connectivity model into account.

Cascading The cascading function, which removes the edges, is executed for both sub-networks as long as we remove at least one edge in one of the networks is removed. We launch the function for each sub-network once during each iteration. The cascading function iterates over each edge $\{v, w\} \in A$ in the sub-network (iteration over edges makes sure to not double-check a node), looks up the corresponding nodes of v and w and their clusters. If v and w are not mutually connected, we remove the edge.

4.4 Limitations

During the simulations, we noticed running times ranging from several minutes up to a week. Even for small networks (such as 4000 nodes), it took us 24 hours to get a data-set of minimal repetitions. The implementation has several bottlenecks, next to the expensive cascading-failure function.

[14] `https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.centrality.closeness_centrality.html`

[15] `https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.algorithms.centrality.betweenness_centrality.html`

[16] `https://networkx.github.io/documentation/networkx-1.10/reference/generated/networkx.DiGraph.degree.html`

Deepcopy The deep copy function G = COPY.DEEPCOPY(G0) of python has a worst case running time of $\mathcal{O}(n^2)$. Calling this function $100 * k$ times, where k is the number of repetitions, and the 100 is the number of different p values we remove, with a growing number of nodes, we spend a lot of time waiting for an object to work on. [17]

Betweenness Centrality The betweenness centrality implementation of NetworkX uses Brandes' algorithm [18] having a running time of $\mathcal{O}(|V|\,|E|)$ for a single node, where V is the total number of nodes, and E is the total number of edges. Doing this operation for $2*N$ nodes, we spend around 90% of the time on this function. [19]

5 Simulation Results and Discussion

One of our goals was to implement the model proposed in [1] and [2] and to reproduce their results. We want to extend the model, by using different mapping schemes, and compare the resilience of the networks based on different connectivity models.

5.1 First order Phase Transition

Figure 2 shows the resilience of interdependent networks of different sizes and the appearance of percolation phase transition. We examine the ER network of different sizes with the same average degree. We plot the fraction of the remaining largest mutually connected cluster v.s. the fraction of nodes that we remove. According to the result, with increasing size of the network, the transition becomes more and more abrupt. This means in the thermodynamic limit, the percolation phase transition is first order. This result is very different from that of single network, where the transitions are second order.

5.2 Influence of Different Graph Types

We would like to compare the turning point of p for each type of interdependent network, so we generate ER, RR and SF (with the power-law coefficient λ equals 3, 2.7 and 2.3) networks of at least 30000 number of nodes, because the first-order transition would be much more apparent for larger enough size. Figure 3 shows the

[17] `http://stackoverflow.com/a/11815286`

[18] `https://networkx.github.io/documentation/development/reference/generated/networkx.generators.random_graphs.random_regular_graph.html`

[19] cProfiling the application, results are in the github repository. Improvements can be done by pre-computation.

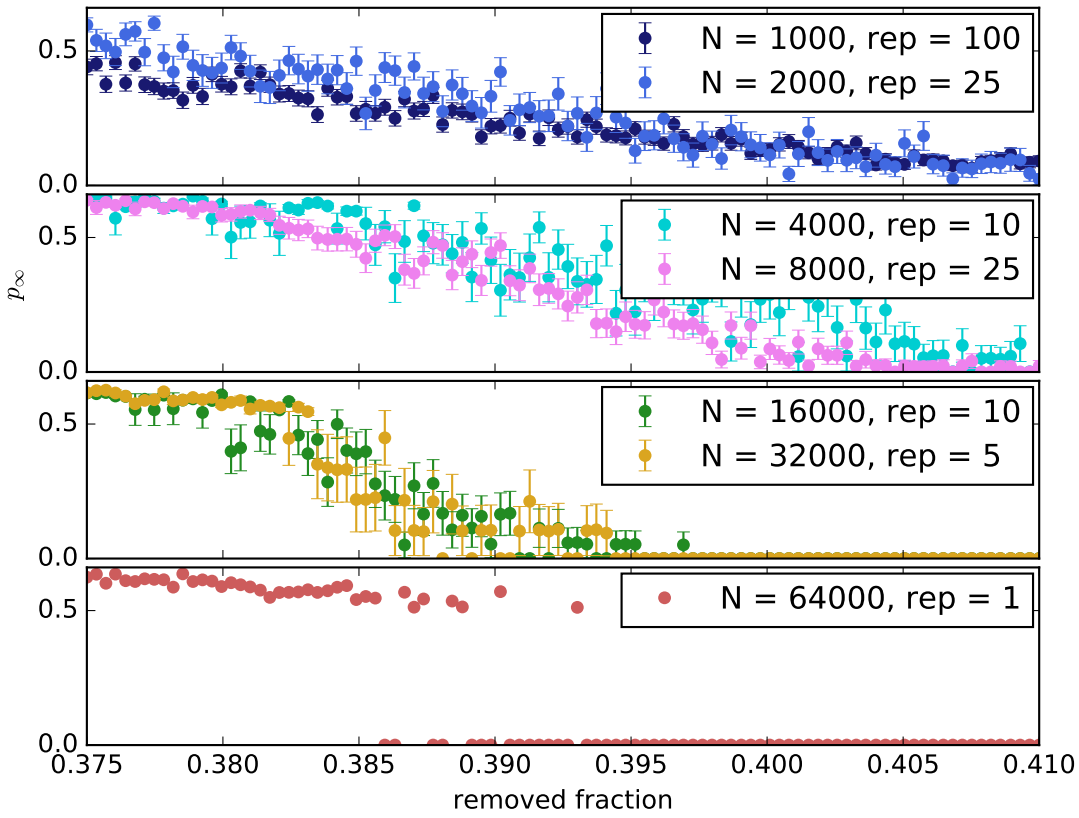

Figure 2: The resilience of interconnected ER networks of different sizes, when a certain number of nodes are removed randomly. From upper panel to lower panel, the size increases, and the percolation transition becomes more appearent and abrupt.

result for all types, and the order of stability is: SF with $\lambda = 2.7 <$ SF with $\lambda = 3 <$ ER $<$ RR.

5.3 Influence of Mapping Schemes

We expect the best behaviour (least clustering for growing initial failure) from the high-to-low mapping, because when one *important* (definition depends on the connectivity model) node fails, the other sub-network is initially keeping many edges. We therefore expect the high-to-high mapping to be the weakest, because a failure in one network has dramatic consequences, and produces the same result in the other sub-network.

Figure 4 shows us a clear distinction between the different mapping schemes

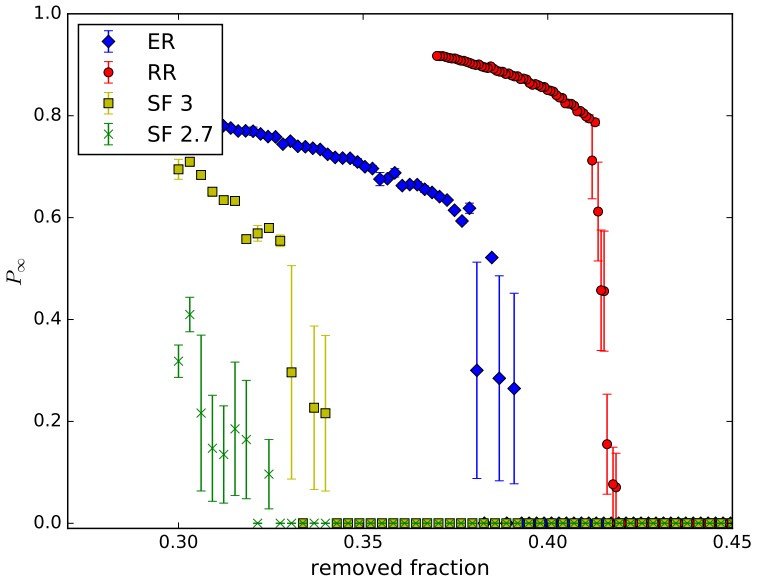

Figure 3: Result for all networks for large nodes. All are of distince first order transition.

for ER graphs. The high-to-high mapping scheme is the most resilient, and the high-to-low mapping scheme is the least. This is at first counter-intuitive. We have to emphasize, that we used a random selection of initial nodes. This random selection has higher chance that a low connected node (holds for all mapping schemes) was targeted, than one of the few highly connected ones. As the *weak* nodes were connected with the *strong* nodes, this resulted in a fatal failure in the corresponding sub-network. For the high-to-high connection model, there was a very small chance to randomly select a *strong* node, compared to the total number of nodes, and thus a failing node in one sub-network led to a minor failure in the corresponding network.

Figure 5.3 shows us that there is no difference in RR graphs for different mapping schemes. In a random regular graph, every node has the same degree, and we can't prioritize one node over another. Every failure has the same impact on the other sub-network and is therefore resulting in the same outcome.

6 Outlook

The high-to-high connectivity model is the strongest for ER graphs in case of random failure. It would be interesting to see the behaviour for targeted attacks (i.e. shutting selected nodes off), compared to the random failure. Here, we expect the random connections to be the most resilient, because the other mapping easily expose their

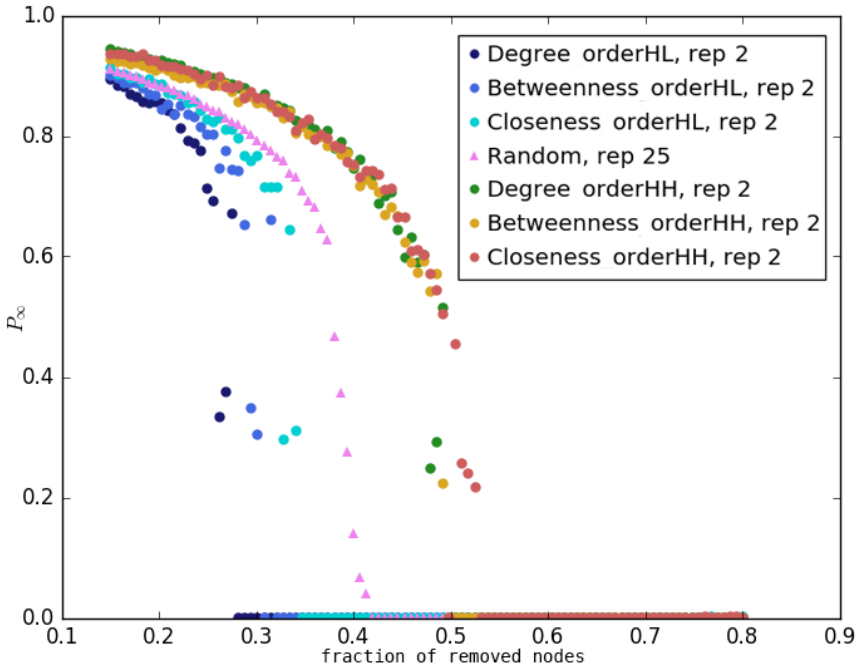

Figure 4: Influence of different mapping schemes on the resilience of interdependent ER networks. We first have random connection (rectangular), then we change the mapping schemes based on nodes' degree, betweenness and closeness. For each mapping scheme, we consider high-to-low mapping, and low-to-high mapping.

strongest components. We also expect the high-to-low to be a bit more resilient than the high-to-high, because by targeting one node, we can always only shut down a single *strong* node compared to the high-to-high.

The used propagation model is very rigid. It would be interesting to compare the results with different cascading models. There are always more ways to interconnect both networks. Additionally, we can include autonomous nodes, this means, nodes that stay functional when their counterpart fails.

References

[1] Sergey V Buldyrev, Roni Parshani, Gerald Paul, H Eugene Stanley, and Shlomo Havlin. Catastrophic cascade of failures in interdependent networks. *Nature*, 464(7291):1025–1028, 2010.

[2] Christian M Schneider, Nuri Yazdani, Nuno AM Araújo, Shlomo Havlin, and

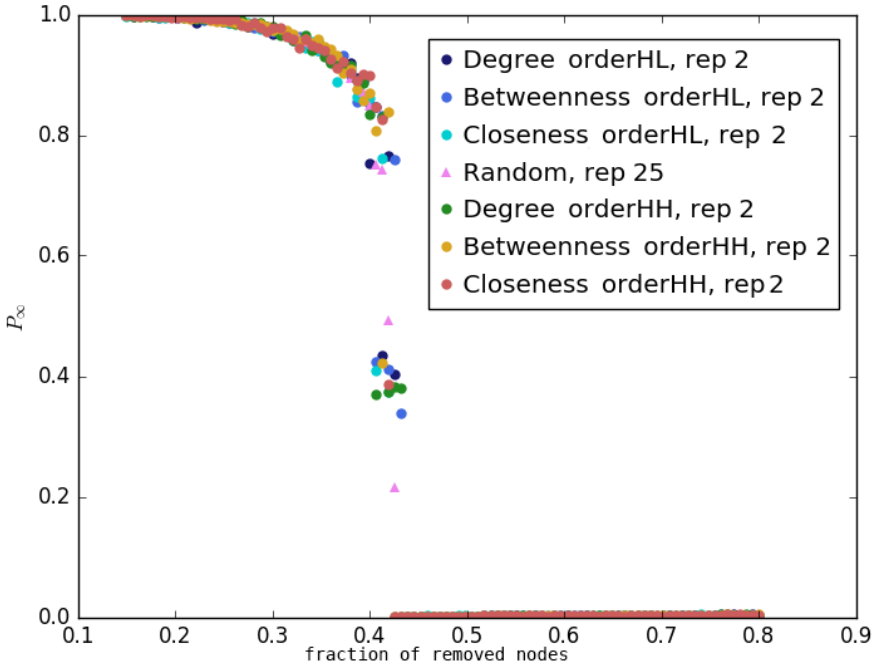

Figure 5: Influence of different mapping schemes on resilience of interdependent random regular (RR) networks.

Hans J Herrmann. Towards designing robust coupled networks. *Scientific reports*, 3, 2013.

Acknowledgement

We would like to give our special thanks to Dr. Evangelos Pournaras and Dr. Olivia Woolley Meza for their support.