IDC 407, Network Science

Information Sharing and Epidemic Modelling in Ecological Networks

Dhruva Sambrani (MS18163)
Aalhad Bhatt (MS18118)
Dhananjay Joshi (MS18065)

21st March 2022

Contents

---

1 Introduction

Social and contact networks are receiving increased attention as powerful tools to investigate disease spread in animal populations. In this term paper we carry out a comparative analysis of social networks of about 400 animal species from different taxa: mammals, birds, reptiles and insects in the context of some common disease spread models and try to draw meaningful conclusions. We also consider some information sharing models and compare the responses generated by different networks depending on their varying network parameters.

2 Network Parameters

We used the following set of network parameters to characterize the current database:

- **Number of Nodes and Edges**: For the present analysis, we only considered undirected animal networks having a minimum of 50 nodes and 100 edges. This filtered out the excessively sparse networks.
- **Average Degree:** $\frac{1}{N}\sum_i^N \sum_j A_{ij}$: The mean degree of a node in the network
- **Connectance** : Refers to the proportion of the realized links of the total number of possible links:
$$\frac{\sum_{i,j} A_{i,j}}{N(N-1)}$$
- **Assortativity** : Assortativity is a number between -1 and 1 that gives us a measure of mixing in the network. A degree of assortativity of 1 means that the network is perfectly mixed.
$$A = \frac{\sum_{ij}\left(A_{ij} - k_i k_j/2m\right)\delta\left(c_i, c_j\right)}{2m - \sum_{ij}\left(k_i k_j/2m\right)\delta\left(c_i, c_j\right)}$$
- **Betweeness Centrality**: The betweeness centrality of a node gives us an idea about how many times that node lies between the geodesic path between two randomly selected vertices of the network. Here we consider the average of betweeness centrality measure of all vertices.
$$X = \frac{\sum_i \sum_{ts} n^i}{N}$$
- **Eigenvector Centrality**: Eigenvector Centrality awards a vertex a centrality score based upon the number of neighbors and how important a neighbour is. The average Eigenvector Centrality here is given by:
$$X = \frac{\sum_i \kappa_1^{-1} \sum_j A_{ij} x_i}{N}$$
- **Clustering Coefficient**: The clustering coefficient gives us a measure of transitivity in the network. It is given by :
$$C = \frac{\text{No. of triangles} \times 6}{\text{Total number of paths of length 2}}$$

- **Mean Path Length**: The average value of shortest paths between all the nodes:

$$M = \frac{\sum_{i,j} \mu_{i,j}}{2N}$$

where $\mu_{i,j}$ is the shortest path between nodes i and j.

3 Models

We consider 5 different epidemic models, all of which are stochastic and discrete time. These are types of compartment models where each node belongs to any one compartment. Transitions, or how a node goes from one compartment to another are based on which compartment it and its neighbours belong to. The compartments under consideration in the models are Susceptible (S), Infected (I), Removed (R), Exposed (E).

1. SI
2. SIR
3. SIS
4. SEIS
5. SEIR

Since many of the models have the same transitions, the mechanism for each transition and the associated parameters are mentioned below.

- S→I: Each S node becomes I with a probability of transition β at each step. The transition requires that at least one neighbour be I. Only occurs if the model does not have E class.
- S→E: Each S node becomes E with a probability of transition β at each step. The transition requires that at least one neighbour be I.
- E→I: Each E node becomes I with a probability of transition α at each step.
- I→R: Each I node becomes R with a probability of transition γ at each step.
- I→S: Each I node becomes S with a probability of transition λ t each step. Only happens in SIS and SEIS.

Each model was initialised with 0.05 of the total nodes being randomly chosen and changed to I. Rest were S. Parameter values chosen were $\beta = 0.1$, $\alpha = 0.05$, $\gamma = 0.005$, and $\lambda = 0.005$.

4 Methods

4.1 Extracting Data

We used the animal network repository compiled by the authors of this [1] paper. They have collected data from various ecology publications and and converted into a python ready format using NetworkX.

4.2 Implementing the Network

We used the python libraries **Networkx** to filter suitable networks from the database and calculate the various network parameters. The epidemic and information sharing models were implemented using the python **ND.Lib** library.

4.3 Output

We found out the normalised total infection duration after running the model for 5000 time steps. This was calculated by dividing summing over the duration each nodes was in the I state and dividing by the size of the network. This gives a measure of the severity of the infection both in terms of proportion of the nodes infected and the duration. This was plotted against the different network parameters for each network and infection model.

4.4 Correlation Measures

We calculated two types of correlation measures for each of the Network Parameter vs the total infected plot for each model.

- **Pearson Correlation Coefficient**: This is a measure of linear correlation between two independent variables : X and Y and is given by the relation:

$$\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y}$$

  Where $\text{cov}(X,Y)$ is the covariance between X and Y, and $\sigma_X \sigma_Y$ are the respective standard deviations. The value of ρ varies between -1 to 1 where -1 represents negative correlation, 0 means no correlation and 1 refers to complete correlation.

- **Distance Correlation**: This measure checks for both linear and non linear correlation and returns a value between 0 (non correlated and independent) to 1 (maximally correlated and linear). Distance correlation does not compare the distribution of the two random variables around their respective means. Instead, it is based off of the pairwise Euclidean distances between two points.

$$a_{ij} = ||X_i - X_j||, b_{ij} = ||Y_i - Y_j||$$

$$A_{ij} = a_{ij} - \bar{a}_{i\cdot} - \bar{a}_{\cdot j} + \bar{a}_{\cdot\cdot}, B_{ij} = b_{ij} - \bar{b}_{i\cdot} - \bar{b}_{\cdot j} + \bar{b}_{\cdot\cdot}$$

$$\text{dcov}(X,Y) = \frac{1}{n^2} \sum_{i,j} A_{ij} B_{i,j}$$

$$\text{dcor}(X,Y) = \frac{\text{dcov}(X,Y)}{\sqrt{\text{dcov}(X,X)\,\text{dcov}(Y,Y)}}$$

5 Results

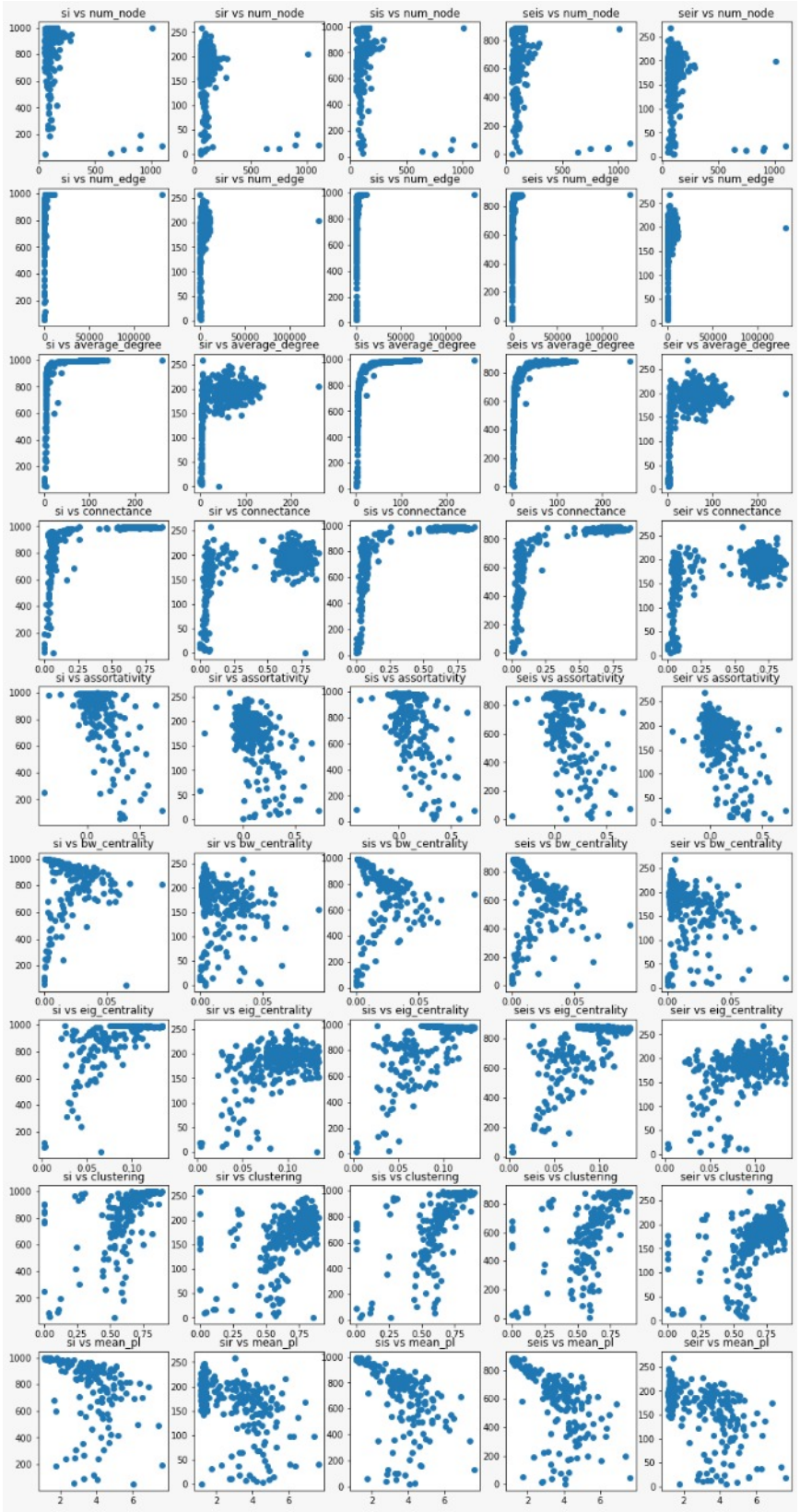
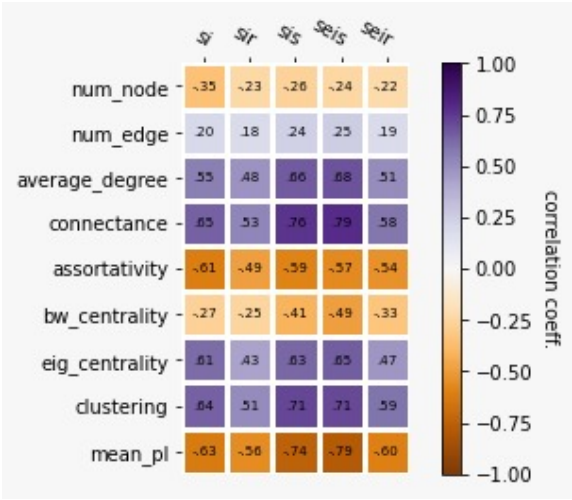
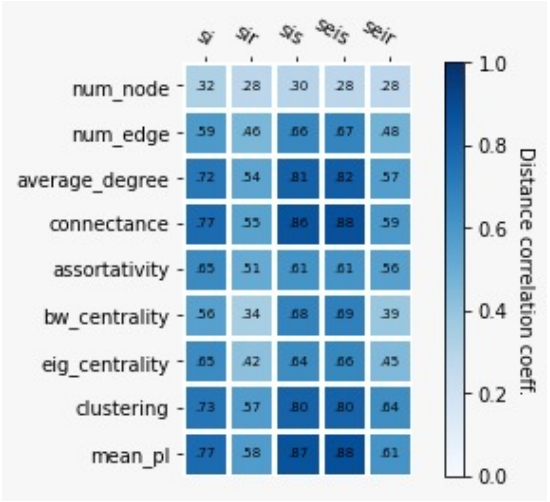


Figure 1: Net infection vs various network parameters



(a) Pearson Coefficient for each network parameter and epidemic model



(b) Distance Correlation for each network parameter and epidemic model

Figure 2: The correlation coefficients reveal which network parameters affect the rate of spread of infection the most

NOTE The SI model can be used as a proxy for information sharing as it has 2 states, 0 corresponding to someone who doesn't know the information and 1 corresponding to someone who does. Once you know the information, you stay in that state forever.

6 Conclusions

Across all the models, the correlation coefficients for normalised total infection duration with different network measures were mostly uniform. SIR and SEIR showed slightly different values from the other 3 while being fairly similar to each other. In the other 3, SI was slightly different from the others. There was no large scale difference, and if one model had a strongly positive correlation, then all the others did as well. As a result, we can speak of he correlation coefficients without specific reference to any model.

For Pearson's Correlation Coefficient, we see that connectance and clustering have the strongest positive correlations while average shortest path length has the strongest negative one. This makes sense, as longer paths will slow disease spread down. High clustering and connectance mean that there are multiple paths for the disease to spread, and therefore it can spread faster. Betweenness centrality, number of odes and number of edges seem to play the smallest role.

Pearson's Correlation Coefficient only looks at linear relationships however, and it might not detect certain important relationships. The direction of the correlation matches what we would expect. More nodes and higher degree assortativity imply weaker disease spread, while the number of edges implies faster spread.

To solve this problem, we also compute the distance correlation. Distance correlation does not specify whether the relationship is positive or negative, but gives a way to calculate the strength independent of the form of the data. Here, the main changes are for betweenness centrality and number of edges, which both now have stronger correlations.

References

[1] Sah, P., Méndez, J.D. & Bansal, S. *A multi-species repository of social networks* (link)

[2] Wikipedia *Pearson Correlation Coefficient* (link)

[3] Peter Gleeson *Finding Correlations in Non-Linear Data* (link)

[4] Mark Newmann *Networks* (link)

main

April 20, 2022

[101]:
```
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np
import scipy as sc
import ndlib.models.epidemics as ep
import ndlib.models.ModelConfig as mc
from joblib import Parallel, delayed
import matplotlib
from scipy.spatial.distance import pdist, squareform
from numba import jit, float32
```

[99]:
```
def average_degree(g):
    return np.average([v for k, v in g.degree()])

def connectance(g):
    m = len(g.edges())
    n = len(g.nodes())
    return 2*m/(n*(n-1))

def assortativity(g):
    return nx.degree_assortativity_coefficient(g)

def bw_centrality(g):
    j = nx.betweenness_centrality(g)
    return np.average([j[k] for k in j])

def eig_centrality(g):
    j = nx.eigenvector_centrality(g)
    return np.average([j[k] for k in j])

def clustering(g):
    return nx.average_clustering(g)

def mean_pl(g):
    largest=g.subgraph(max(nx.connected_components(g), key=len)).copy()
    return nx.average_shortest_path_length(largest)
```

```
def num_node(g):
    return len(g.nodes())

def num_edge(g):
    return len(g.edges())
```

[100]:
```
def sir(g):
    model = ep.SIRModel(g)
    config = mc.Configuration()
    config.add_model_parameter('beta', 0.01)
    config.add_model_parameter('gamma', 0.005)
    config.add_model_parameter("fraction_infected", 0.05)
    model.set_initial_status(config)
    iterations = model.iteration_bunch(5000)
    trends = model.build_trends(iterations)
    return np.sum(trends[0]['trends']['node_count'][1])/len(g.nodes())

def si(g):
    model = ep.SIModel(g)
    config = mc.Configuration()
    config.add_model_parameter('beta', 0.01)
    config.add_model_parameter("fraction_infected", 0.05)
    model.set_initial_status(config)
    iterations = model.iteration_bunch(5000)
    trends = model.build_trends(iterations)
    return np.sum(trends[0]['trends']['node_count'][1])/len(g.nodes())

def sis(g):
    model = ep.SISModel(g)
    config = mc.Configuration()
    config.add_model_parameter('beta', 0.01)
    config.add_model_parameter('lambda', 0.005)
    config.add_model_parameter("fraction_infected", 0.05)
    model.set_initial_status(config)
    iterations = model.iteration_bunch(5000)
    trends = model.build_trends(iterations)
    return np.sum(trends[0]['trends']['node_count'][1])/len(g.nodes())

def seir(g):
    model = ep.SEIRModel(g)
    config = mc.Configuration()
    config.add_model_parameter('beta', 0.01)
    config.add_model_parameter('gamma', 0.005)
    config.add_model_parameter('alpha', 0.05)
    config.add_model_parameter("fraction_infected", 0.05)
    model.set_initial_status(config)
    iterations = model.iteration_bunch(5000)
```

```
    trends = model.build_trends(iterations)
    return np.sum(trends[0]['trends']['node_count'][1])/len(g.nodes())

def seis(g):
    model = ep.SEISModel(g)
    config = mc.Configuration()
    config.add_model_parameter('beta', 0.01)
    config.add_model_parameter('lambda', 0.005)
    config.add_model_parameter('alpha', 0.05)
    config.add_model_parameter("fraction_infected", 0.05)
    model.set_initial_status(config)
    iterations = model.iteration_bunch(5000)
    trends = model.build_trends(iterations)
    return np.sum(trends[0]['trends']['node_count'][1])/len(g.nodes())
```

[11]:
```
props = [num_node, num_edge, average_degree, connectance, assortativity,␣
 ↪bw_centrality, eig_centrality, clustering, mean_pl]
models = [si, sir, sis, seis, seir]
```

[26]:
```
def get_properties(g):
    props_dict = {}
    for prop in props:
        print("finding", prop.__name__)
        try:
            props_dict[prop.__name__] = prop(g)
        except:
            props_dict[prop.__name__] = np.na
    return props_dict
```

[27]:
```
def run_sims(g):
    sims = {}
    for sim_type in [si, sir, sis, seis, seir]:
        print("running model", sim_type.__name__)
        sims[sim_type.__name__] = sim_type(g)
    return sims
```

[28]:
```
def run_everything(g, i=0, total=0):
    print("Running for graph", i, "of", total)
    props = get_properties(g)
    sim_vals = run_sims(g)
    return props, sim_vals
```

[107]:
```
def make_correlation_plots(data):
    figure, axis = plt.subplots(len(props), len(models), figsize=(15,30))
    pearson_correl = np.zeros((len(props), len(models)))
    distance_correl = np.zeros((len(props), len(models)))
    for i, prop in enumerate(props):
```

```
        for j, model in enumerate(models):
            xvals = []
            yvals = []
            for datum in data:
                xvals.append(datum[0][prop.__name__])
                yvals.append(datum[1][model.__name__])
            axis[i, j].scatter(xvals, yvals)
            axis[i, j].set_title(model.__name__ + " vs " + (prop.__name__))
            distance_correl[i, j] = distcorr(np.array(xvals)[~np.isnan(xvals)],␣
↪np.array(yvals)[~np.isnan(xvals)])
            pearson_correl[i, j] = np.corrcoef(np.array(xvals)[~np.
↪isnan(xvals)], np.array(yvals)[~np.isnan(xvals)])[0, 1]

    plt.show()
    return pearson_correl, distance_correl
```

[86]:
```
def heatmap(data, row_labels, col_labels, ax=None,
            cbar_kw={}, cbarlabel="", **kwargs):
    if not ax:
        ax = plt.gca()
    im = ax.imshow(data, **kwargs)
    cbar = ax.figure.colorbar(im, ax=ax, **cbar_kw)
    cbar.ax.set_ylabel(cbarlabel, rotation=-90, va="bottom")
    ax.set_xticks(np.arange(data.shape[1]), labels=col_labels)
    ax.set_yticks(np.arange(data.shape[0]), labels=row_labels)
    ax.tick_params(top=True, bottom=False,
                   labeltop=True, labelbottom=False)
    plt.setp(ax.get_xticklabels(), rotation=-30, ha="right",
             rotation_mode="anchor")
    ax.spines[:].set_visible(False)
    ax.set_xticks(np.arange(data.shape[1]+1)-.5, minor=True)
    ax.set_yticks(np.arange(data.shape[0]+1)-.5, minor=True)
    ax.grid(which="minor", color="w", linestyle='-', linewidth=3)
    ax.tick_params(which="minor", bottom=False, left=False)
    return im, cbar

def annotate_heatmap(im, data=None, valfmt="{x:.2f}",
                     textcolors=("black", "black"),
                     threshold=None, **textkw):
    if not isinstance(data, (list, np.ndarray)):
        data = im.get_array()
    if threshold is not None:
        threshold = im.norm(threshold)
    else:
        threshold = im.norm(data.max())/2.
    kw = dict(horizontalalignment="center",
              verticalalignment="center")
```

```
    kw.update(textkw)
    if isinstance(valfmt, str):
        valfmt = matplotlib.ticker.StrMethodFormatter(valfmt)
    texts = []
    for i in range(data.shape[0]):
        for j in range(data.shape[1]):
            kw.update(color=textcolors[int(im.norm(data[i, j]) > threshold)])
            text = im.axes.text(j, i, valfmt(data[i, j], None), **kw)
            texts.append(text)
    return texts
def func(x, pos):
    return "{:.2f}".format(x).replace("0.", ".").replace("1.00", "")
```

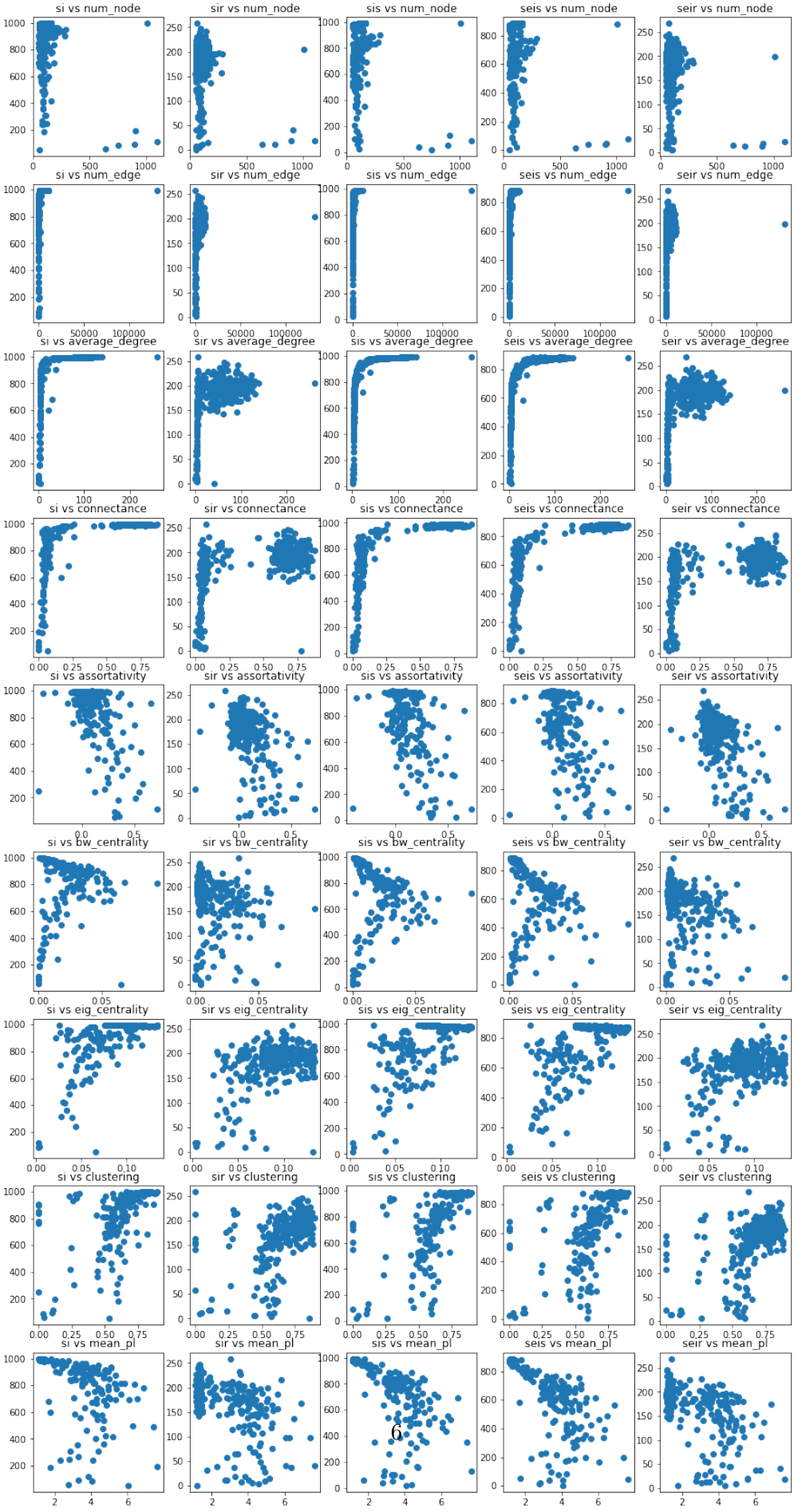
[102]:
```
def distcorr(X, Y):
    X = np.atleast_1d(X)
    Y = np.atleast_1d(Y)
    if np.prod(X.shape) == len(X):
        X = X[:, None]
    if np.prod(Y.shape) == len(Y):
        Y = Y[:, None]
    X = np.atleast_2d(X)
    Y = np.atleast_2d(Y)
    n = X.shape[0]
    if Y.shape[0] != X.shape[0]:
        raise ValueError('Number of samples must match')
    a = squareform(pdist(X))
    b = squareform(pdist(Y))
    A = a - a.mean(axis=0)[None, :] - a.mean(axis=1)[:, None] + a.mean()
    B = b - b.mean(axis=0)[None, :] - b.mean(axis=1)[:, None] + b.mean()

    dcov2_xy = (A * B).sum()/float(n * n)
    dcov2_xx = (A * A).sum()/float(n * n)
    dcov2_yy = (B * B).sum()/float(n * n)
    dcor = np.sqrt(dcov2_xy)/np.sqrt(np.sqrt(dcov2_xx) * np.sqrt(dcov2_yy))
    return dcor
```

[90]:
```
graph_paths = ["../"+i.strip() for i in open("../good_ones").readlines()]
all_graphs = []
for i, path in enumerate(graph_paths):
    #print("Reading network data", i+1, "of", len(graph_paths))
    all_graphs.append(nx.read_graphml(path))
```

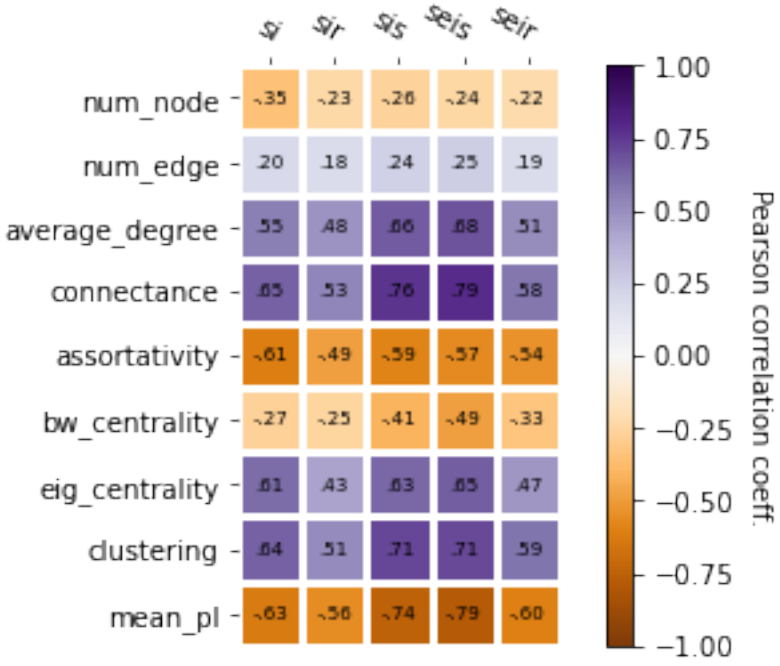
[ ]:
```
data = Parallel(n_jobs=8)(delayed(run_everything)(i) for i in all_graphs)
```

[112]:
```
p_c, d_c = make_correlation_plots(data)
```

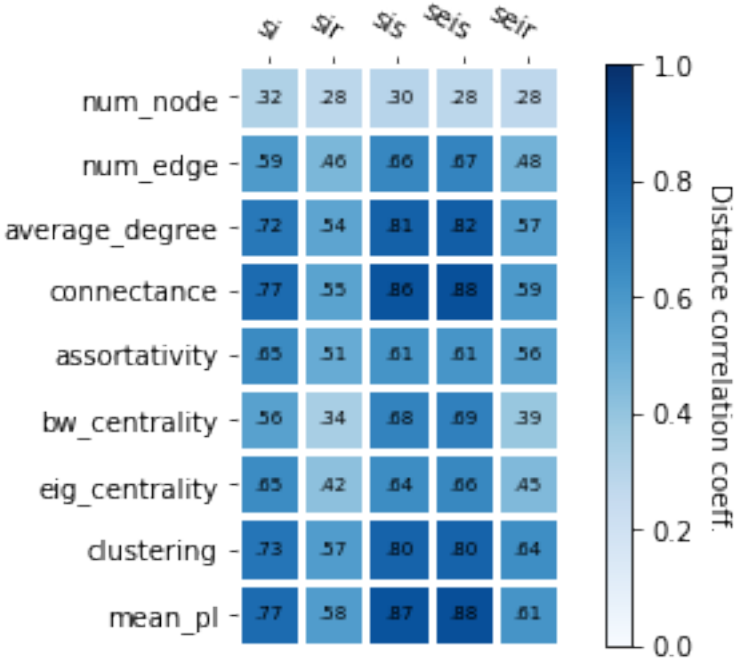

si vs num_node
sir vs num_node
sis vs num_node
seis vs num_node
seir vs num_node
si vs num_edge
sir vs num_edge
sis vs num_edge
seis vs num_edge
seir vs num_edge
si vs average_degree
sir vs average_degree
sis vs average_degree
seis vs average_degree
seir vs average_degree
si vs connectance
sir vs connectance
sis vs connectance
seis vs connectance
seir vs connectance
si vs assortativity
sir vs assortativity
sis vs assortativity
seis vs assortativity
seir vs assortativity
si vs bw_centrality
sir vs bw_centrality
sis vs bw_centrality
seis vs bw_centrality
seir vs bw_centrality
si vs eig_centrality
sir vs eig_centrality
sis vs eig_centrality
seis vs eig_centrality
seir vs eig_centrality
si vs clustering
sir vs clustering
sis vs clustering
seis vs clustering
seir vs clustering
si vs mean_pl
sir vs mean_pl
sis vs mean_pl
seis vs mean_pl
seir vs mean_pl

[111]:
```
im2, f2 = heatmap(p_c, [prop.__name__ for prop in props], [model.__name__ for␣
 ↪model in models],
                  cmap="PuOr", vmin=-1, vmax=1,
                  cbarlabel="Pearson correlation coeff.")

annotate_heatmap(im2, valfmt=matplotlib.ticker.FuncFormatter(func), size=7);
```



[113]:
```
im1, f1 = heatmap(d_c, [prop.__name__ for prop in props], [model.__name__ for␣
 ↪model in models],
                  cmap="Blues", vmin=0, vmax=1,
                  cbarlabel="Distance correlation coeff.")
annotate_heatmap(im1, valfmt=matplotlib.ticker.FuncFormatter(func), size=7);
```


si
sir
sis
seis
seir
num_node
num_edge
average_degree
connectance
assortativity
bw_centrality
eig_centrality
clustering
mean_pl
.32
.28
.30
.28
.28
.59
.46
.66
.67
.48
.72
.54
.81
.82
.57
.77
.55
.86
.88
.59
.65
.51
.61
.61
.56
.56
.34
.68
.69
.39
.65
.42
.64
.66
.45
.73
.57
.80
.80
.64
.77
.58
.87
.88
.61
1.0
0.8
0.6
0.4
0.2
0.0
Distance correlation coeff.

[ ]: