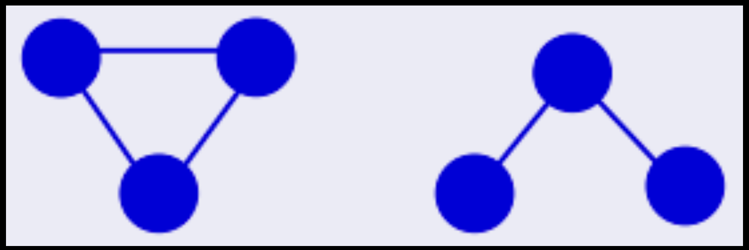
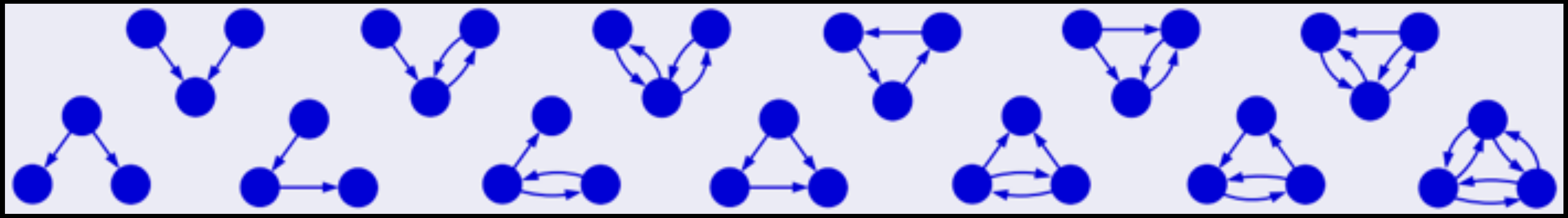
ROSS FLIEGER-ALLISON

EXTRACTING TRIAD MOTIF SIGNIFICANCE PROFILES FROM COMPLEX NETWORKS

NETWORK MOTIFS

- **motif**: a connected non-isomorphic induced subgraph with a predefined link configuration
- **triad motif**: motif consisting of 3 nodes
- 2 undirected, 13 directed:





Images taken from lecture slides.

WHY THEY MATTER

- Help determine **network properties**.
- Play a key role in **system performance** of certain tasks.
- **Triadic** > **Dyadic**

WHAT IS A SIGNIFICANCE PROFILE?

- **significance profile**: quantifier of motif over- or underrepresentation in a network
- Uses **normalized z-scores**.

$$SP_i = \frac{z_i}{\sqrt{\sum_j z_j^2}}$$

$$z_i = \frac{N_{\mathrm{original},i} - \langle N_{\mathrm{rand},i} \rangle}{\sigma_{\mathrm{rand},i}}$$

THE PROBLEM

FIND AN EFFICIENT MEANS OF EXTRACTING TRIAD MOTIF EXPRESSION FROM A NETWORK

MY SOLUTION (MOTIF COUNTING)

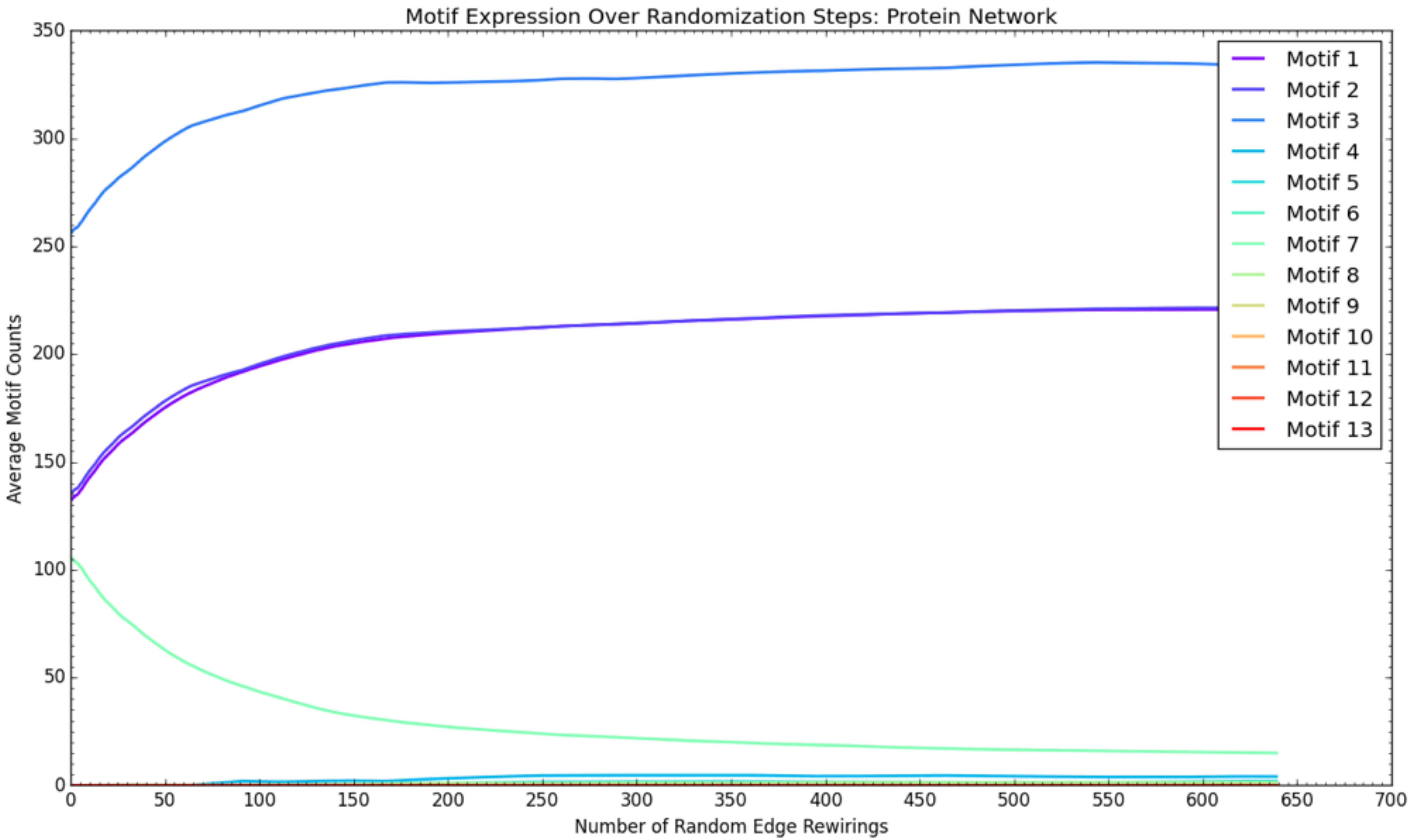
- Iterate through edges (a, b).
- Find all unique neighbors of a and b (call them c).
- Search for motifs in (a, b, c) triads such that every connected node triplet is searched exactly once.
- Return aggregated counts.

MY SOLUTION (Z-SCORES)

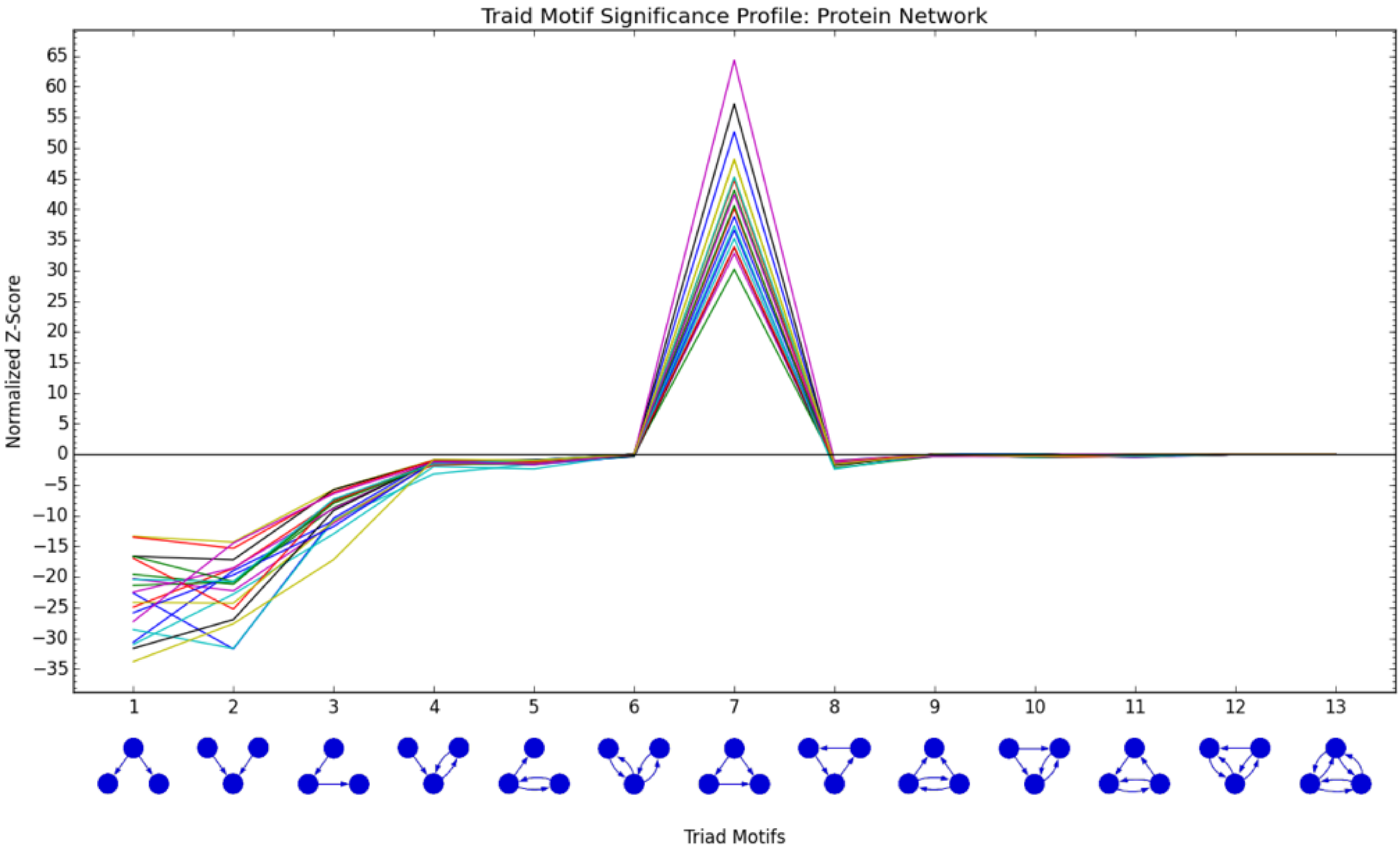
- Count the motifs in the original network.
- Randomize the network such that its degree sequence is maintained.
- Count the motif occurrences in randomized instances.
- Average the random network motif counts (expected value).
- Compute the standard deviation of randomized counts.
- Use this to compute the z-score.

$$z_i = \frac{N_{\mathbf{original},i} - \langle N_{\mathbf{rand},i} \rangle}{\sigma_{\mathbf{rand},i}}$$

RANDOMIZATION STEPS

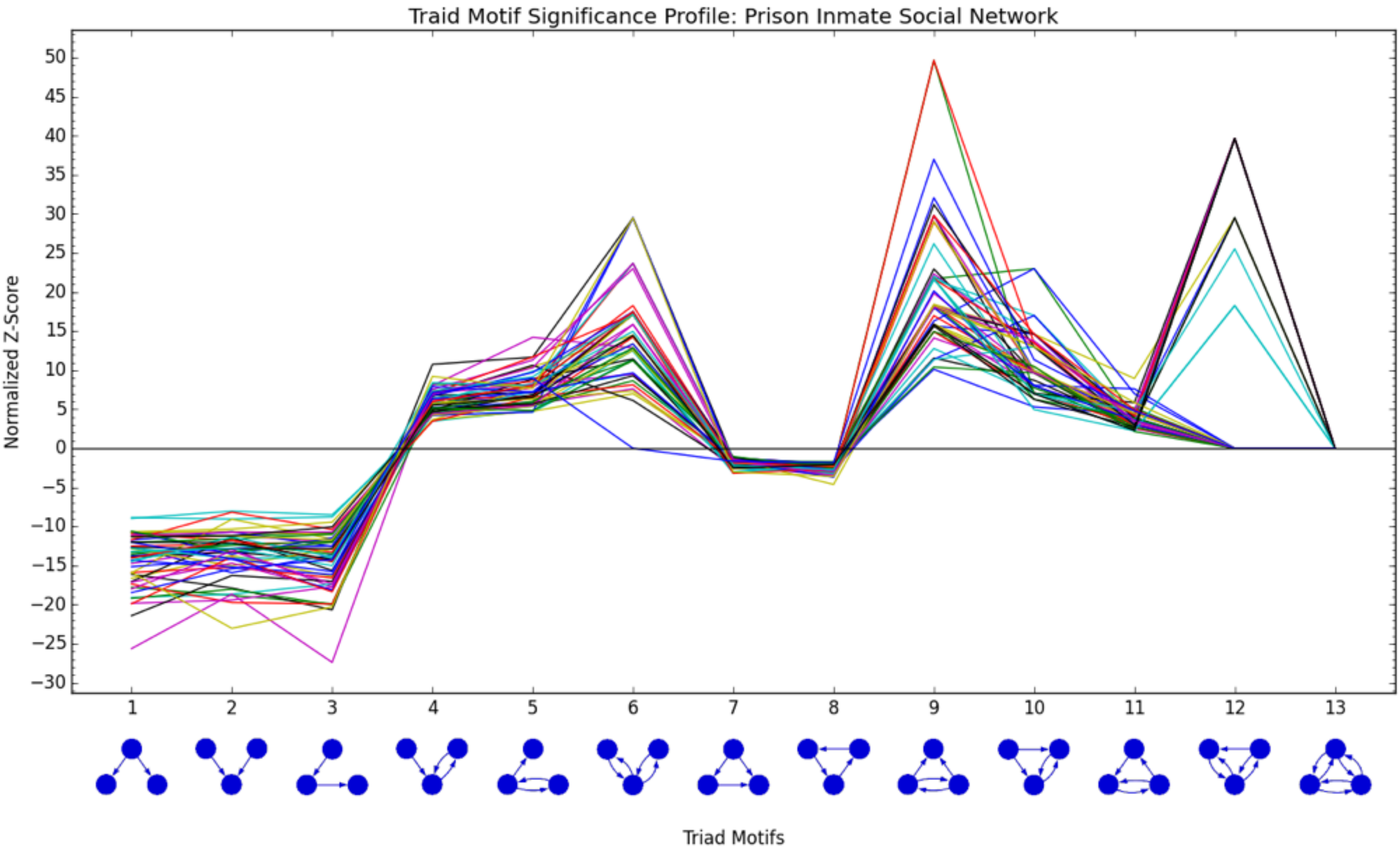


THE RESULT

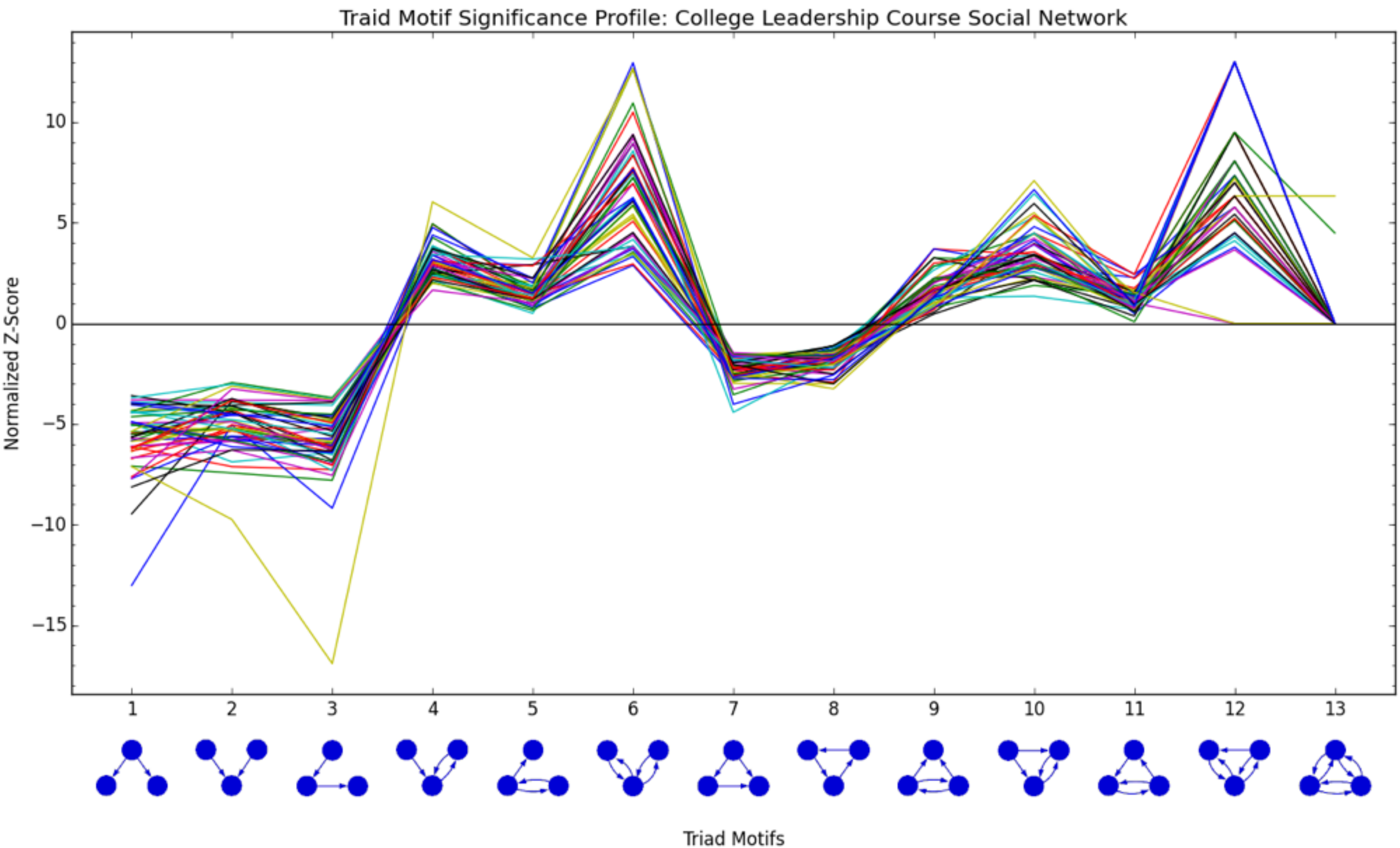


Traid Motif Significance Profile: Protein Network
Normalized Z-Score
65
60
55
50
45
40
35
30
25
20
15
10
5
0
−5
−10
−15
−20
−25
−30
−35
1
2
3
4
5
6
7
8
9
10
11
12
13
Triad Motifs

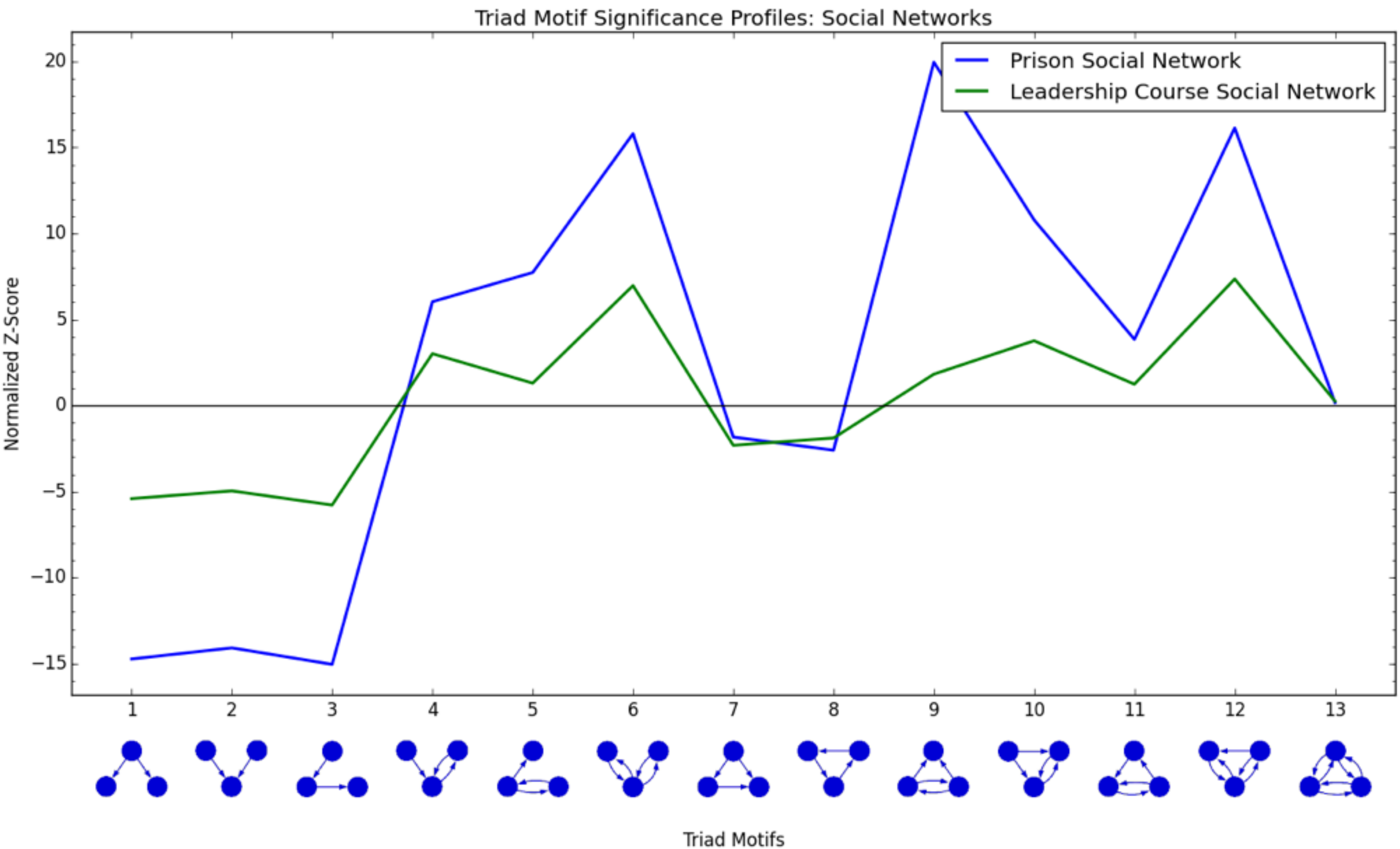
PRISON INMATE SOCIAL NETWORK



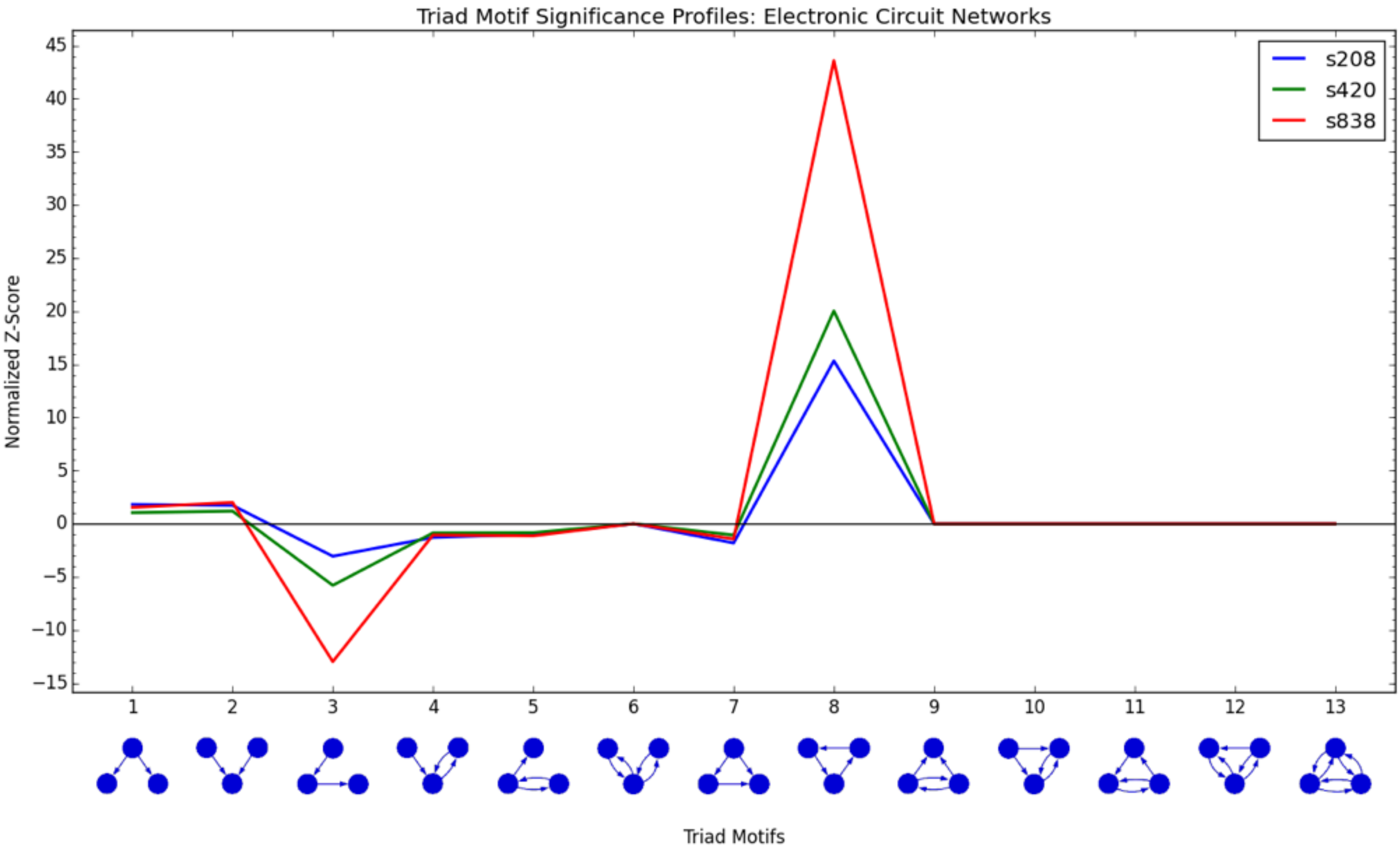
LEADERSHIP COURSE SOCIAL NETWORK



SOCIAL NETWORKS

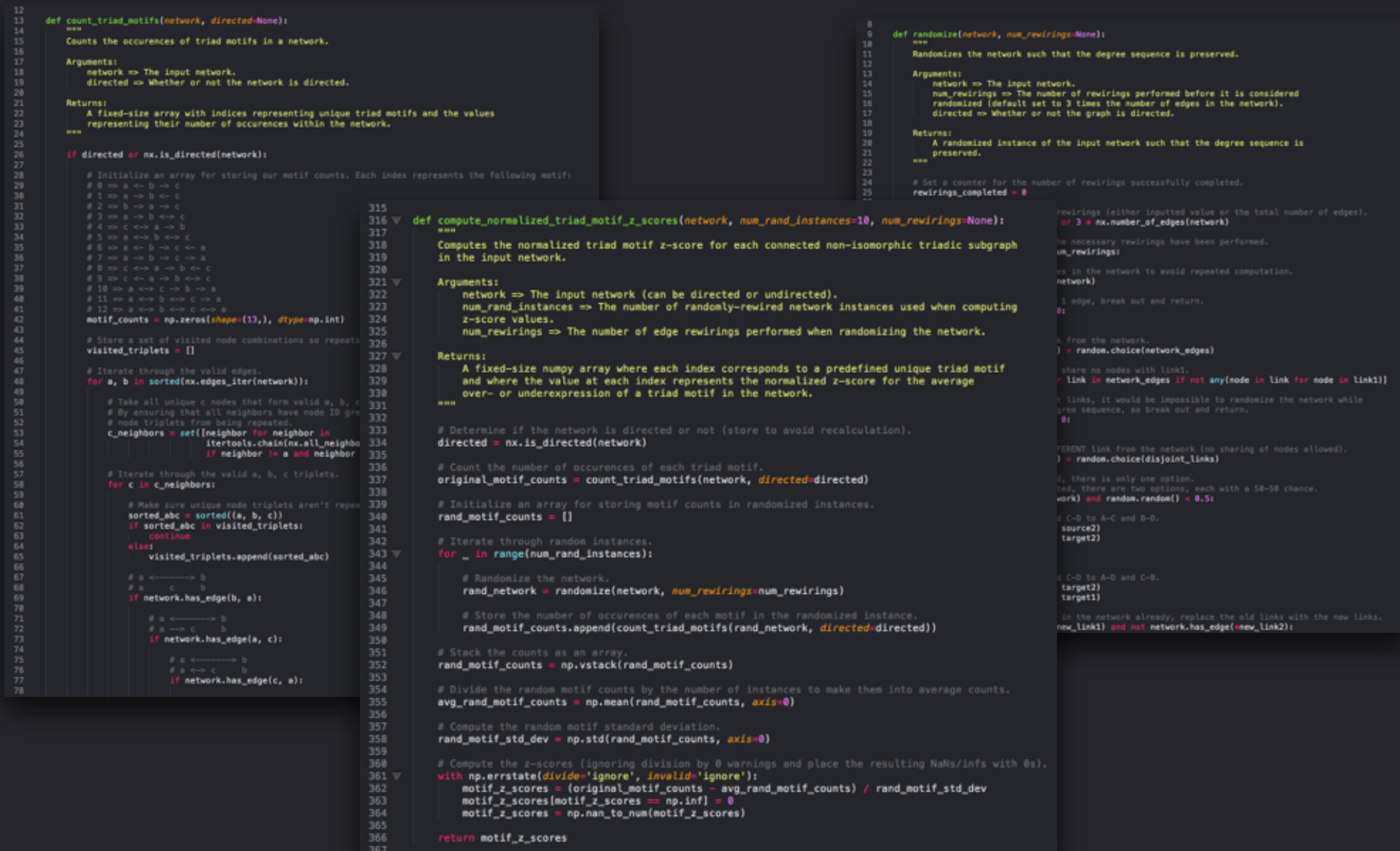


ELECTRONIC NETWORKS



REFERENCES

- Prison dataset: J. P. Eckmann, E. Moses, Proc. Natl. Acad. Sci. U.S.A. 99, 5825 (2002).
- Leadership course dataset: M. A. J. Van Duijn, M. Huisman, F. N. Stokman, F. W. Wasseur, E. P. H. Zeggelink, J. Math. Sociol. 27 (2003).
- Protein dataset: from lab
- Electronic circuit datasets: R.F. Cancho, C. Janssen, R.V. Sole, Phys. Rev. E 6404 (2001) 046119.


def compute_normalized_triad_motif_z_scores(network, num_rand_instances=10, num_rewirings=None):
"""
Computes the normalized triad motif z-score for each connected non-isomorphic triadic subgraph
in the input network.
Arguments:
network => The input network (can be directed or undirected).
num_rand_instances => The number of randomly-rewired network instances used when computing
z-score values.
num_rewirings => The number of edge rewirings performed when randomizing the network.
Returns:
A fixed-size numpy array where each index corresponds to a predefined unique triad motif
and where the value at each index represents the normalized z-score for the average
over- or underexpression of a triad motif in the network.
"""
# Determine if the network is directed or not (store to avoid recalculation).
directed = nx.is_directed(network)
# Count the number of occurences of each triad motif.
original_motif_counts = count_triad_motifs(network, directed=directed)
# Initialize an array for storing motif counts in randomized instances.
rand_motif_counts = []
# Iterate through random instances.
for _ in range(num_rand_instances):
# Randomize the network.
rand_network = randomize(network, num_rewirings=num_rewirings)
# Store the number of occurences of each motif in the randomized instance.
rand_motif_counts.append(count_triad_motifs(rand_network, directed=directed))
# Stack the counts as an array.
rand_motif_counts = np.vstack(rand_motif_counts)
# Divide the random motif counts by the number of instances to make them into average counts.
avg_rand_motif_counts = np.mean(rand_motif_counts, axis=0)
# Compute the random motif standard deviation.
rand_motif_std_dev = np.std(rand_motif_counts, axis=0)
# Compute the z-scores (ignoring division by 0 warnings and place the resulting NaNs/infs with 0s).
with np.errstate(divide='ignore', invalid='ignore'):
motif_z_scores = (original_motif_counts - avg_rand_motif_counts) / rand_motif_std_dev
motif_z_scores[motif_z_scores == np.inf] = 0
motif_z_scores = np.nan_to_num(motif_z_scores)
return motif_z_scores

Questions?