CPBS 7711 MODULE 2 DAY 3 ASSIGNMENT: Aishwarya Mandava

MOTIVATION:

Fanconi Anemia (FA) is a rare genetic disorder inherited in an autosomal recessive pattern and is characterized by physical abnormalities, bone marrow failure, and increased risk of malignancy. Around 90% of the individuals with FA have impaired bone marrow function leading to decrease in the production of Red blood cells, White blood cells, and Platelets [1]. Previous studies have found 12 genes associated with FA, including BRCA and FANC genes that play a role in DNA damage response and repair mechanisms. The construction of functional networks for FA-associated genes could offer valuable insights into novel molecular mechanisms and pathways.

COMPUTATIONAL PROBLEM:

We have a protein-protein interactions network representing various interactions between the nodes/genes and a set of loci associated with FA disease. The computational objective is to generate a population of random subnetworks, comprising 5000 subnetworks of disease associated loci and to subsequently compute the statistical significance of this final population of subnetworks. Furthermore, this involves generating a null population of 5000 subnetworks with non-informative collections of loci formed by randomly selecting genes such that the number of edges falls within the same range as that of the disease associated genes in the initial population.

SPECIFIC APPROACH:

Given that we have the disease (FA) associated genes and the protein-protein interactions (PPI), the approach is to first generate a population of 5000 random subnetworks of 12 genes by randomly selecting one gene per loci. Following that, generate another population of 5000 random subnetworks with non-informative collections of loci by replacing the FA genes in the initial subnetwork with the genes that have similar range of edges as them. Finally, compute the statistical significance using non-parametric permutation test with 10000 permutations.

SPECIFIC IMPLEMENTATION:

The protein-protein interactions (PPI) network was retrieved from the STRING database [2] in the tab-delimited format. This file has 1,972,248 interactions across various genes including both FA genes and non-FA genes. The FA disease genes were retrieved from the OMIM [3] database in the Gene Map Table (GMT) format. This file has 12 loci, and 584 FA disease genes. However, not all 584 interact with other genes, only 330 disease associated genes interact with other genes from the PPI networks. Table 1 shows disease associated genes across loci before and after filtering.

Locus	0	1	2	3	4	5	6	7	8	9	10	11
All FA genes	46	50	79	50	12	50	50	50	50	48	50	50
FA genes from STRING file	28	27	54	31	4	27	29	27	28	24	25	26

Table1: Number of disease associated genes

A) Generating 5000 subnetworks with FA genes:

This step involves selecting a random gene from each loci to form a subnetwork. This is repeated 5000 times resulting in generation of distinct subnetworks. There are 4.67e+16 potential ways of creating a subnetwork by randomly selecting one gene per locus (note - these 12 genes may or may not have interactions with the other genes in that subnetwork) and each subnetwork could have at most 12C2 or (12*11)/2 or 66 edges. Figure1 shows 4 loci and illustrates the 5000 random subnetworks generated from the 12 loci.

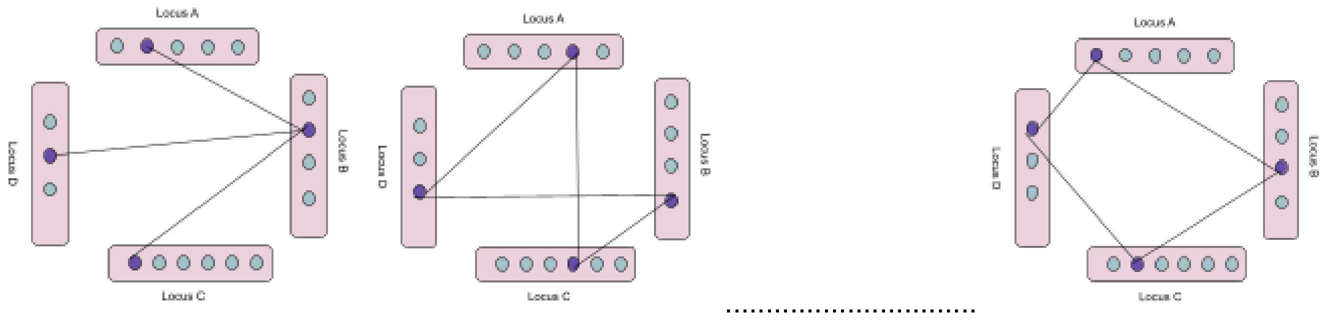


Figure1

B) Generating 5000 non-informative subnetworks:

In this step, a null distribution is created by substituting non-informative genes (those not associated with the disease) for disease-associated genes from the initial population of 5000 random subnetworks. All genes from the string network are sorted into 500 bins based on the number of edges they are associated with. The replacement of disease-associated genes from the original set of subnetworks is done by aligning the bin categories with those of the non-disease associated genes. Figure2 illustrates these bins categorized by the number of edges for both disease associated and non-disease associated genes, along with their respective randomly generated subnetworks.

A random seed is used to reproduce the index value for generation of subnetworks. Setting the initial seed1 to 123 and seed2 to 345, the random index is generated by using (seed1 * seed2) % len(list) which ensures that the random index generated is within the range of this list indices.

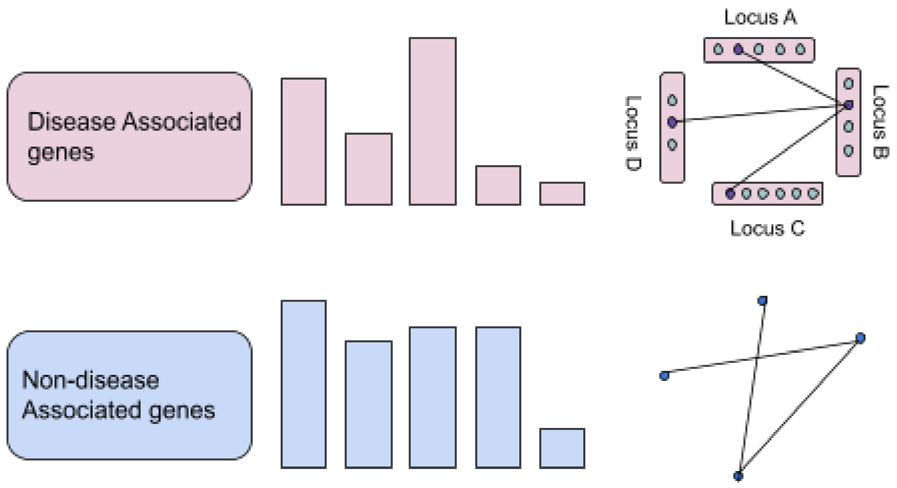


Figure2

C) Computing statistical significance of a population of FA subnetworks using non-parametric Permutation test and generating empirical p-value

To test the statistical significance of the initial population of FA subnetworks, a non-parametric test (two tailed permutation test) is applied.

Null hypothesis (H0): mean = 0 (no significant difference between the FA subnetwork and nonFA subnetwork distributions and any observed difference between the two groups of subnetworks is due to random chance alone

Alternative hypothesis (H1): mean $\neq$ 0 (there is a significant difference between the FA subnetwork and nonFA subnetwork distributions and there is a true difference between the two groups (not by random chance))

For the permutation test, the true observed absolute mean difference in the number of edges between the 5000 random FA subnetworks and non-informative subnetworks is computed. The subnetworks are then combined and a permuted datasets are drawn without replacement. Alternatively, the current approach randomly swaps the labels, followed by which, the permuted absolute mean difference between the two groups is calculated as the permutation statistic. The permutation steps are repeated 10000 times generating a distribution of permutation statistics. The empirical p-value is the probability of observing as extreme or more extreme value than the observed statistic. The p-value is determined by the proportion of permutation statistics that are greater than (more extreme) or equal to (as extreme) the true observed mean difference in the number of edges.

$$empirical\, p\, value = number\ of\ extreme\ and\ more\ extreme\ value / total\ number\ of\ permutations$$

For randomization, similar random seed approach is used as in B)

Figure3 illustrates the steps in permutation test. The first column displays the true labels of subnetworks while the subsequent columns show randomly swapped labels, representing different permutations.

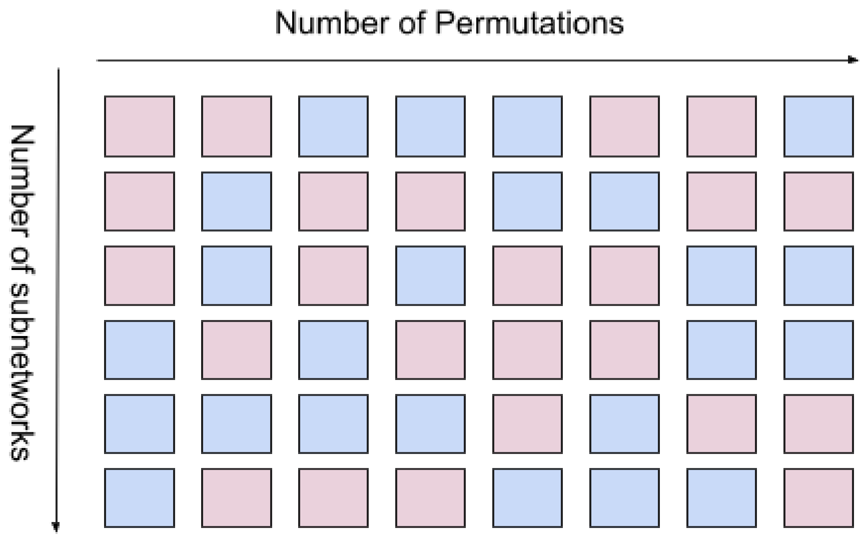


Figure 3

RESULTS:

Distribution of gene frequencies:

Figure 4a shows a histogram of gene frequencies for a total of 16845 genes. The gene *UBC* stands out as a non-FA gene with highest number of edges, totaling 10101 edges. Following *UBC*, *UBB* and *ZFAND4* have 2824 and 2594 edges respectively. These genes are categorized into bins where no FA genes share the same range of frequency of edges. As a result, these genes have been filtered out prior to subnetwork generation. Figure 4b shows a histogram after filtering. Specifically, genes with frequencies exceeding 1000 are targeted for filtering. In addition, these genes do not share bins with FA genes.

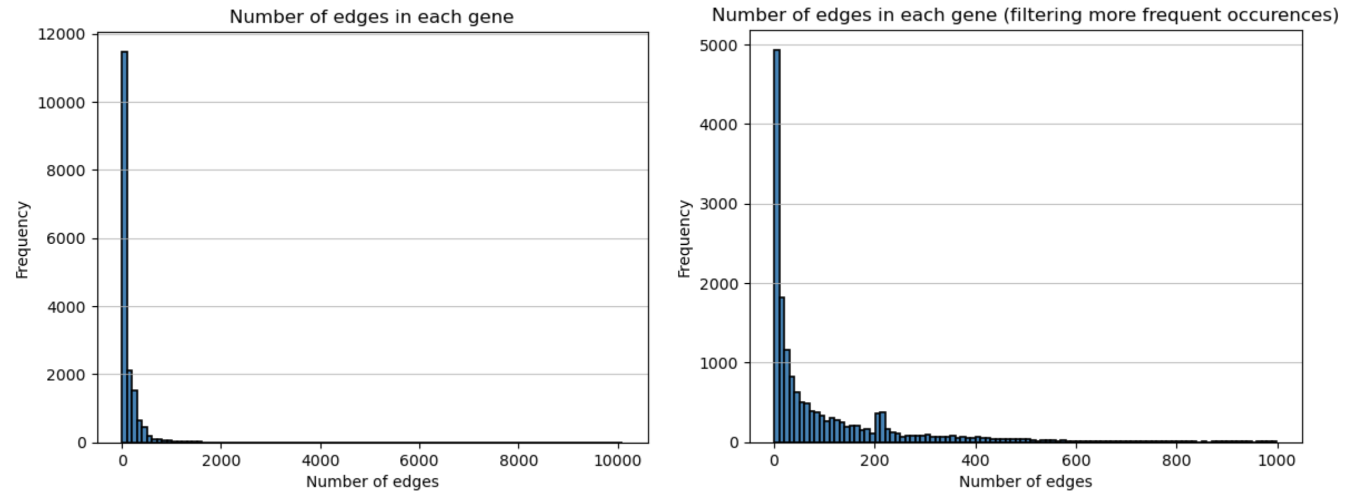


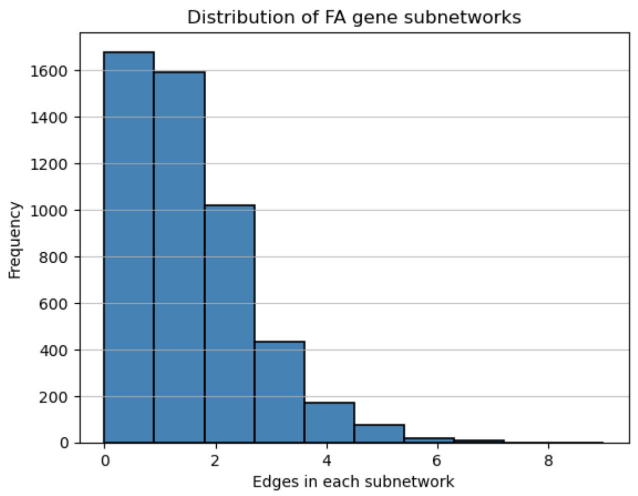
Figure 4a and 4b

Random subnetworks:

Figures 5a and 5b show the distributions of the 5000 random FA subnetworks and nonFA subnetworks (Null) respectively. Table 2 shows the frequency of the number of edges in the two distributions. Mean of the FA subnetworks and null distribution are calculated as 1.2358 and 1.3362 respectively. The true observed mean is 0.1004 (absolute)

	0	1	2	3	4	5	6	7	8	9
FA subnetwork distribution	1679	1590	1021	434	172	77	19	7	--	1
Null	1792	1467	787	484	274	122	47	19	5	3

Table2



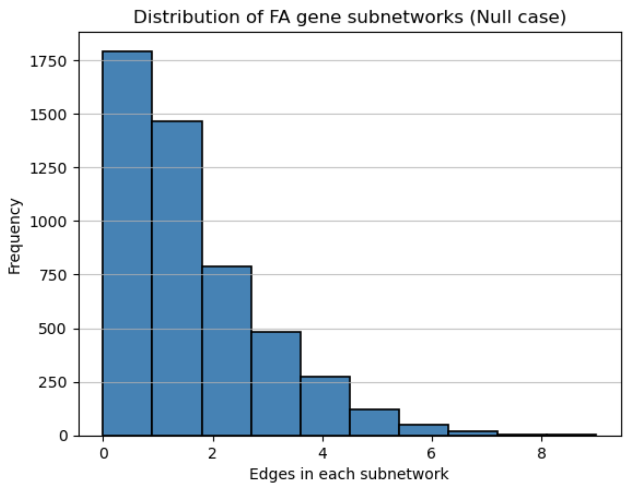


Figure 5a and 5b

Permutation test:
Figure 6 shows the distribution of permutation statistic (mean difference in the permuted groups). The red dotted line is the true observed mean difference (0.1004) between the two groups of subnetworks. Of the 10000 permutations, none of the permutation statistic was as extreme or more extreme than the observed mean. The empirical p-value was computed as 0.

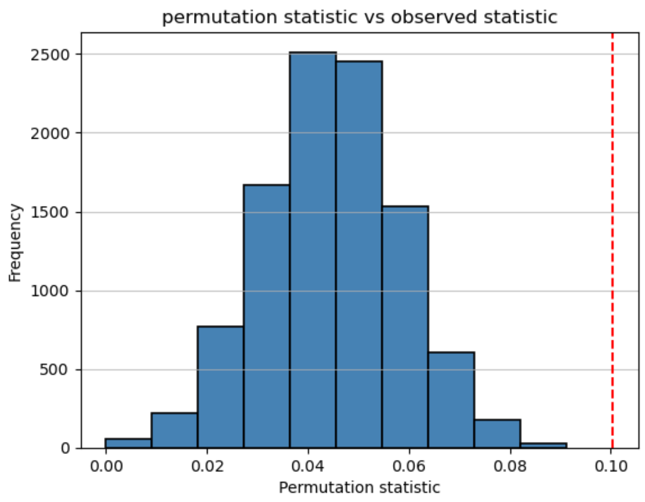


Figure 6

DISCUSSION:
The empirical p-value is computed as 0 using the permutation test, suggesting that we reject the null hypothesis that there is no true difference between the two groups of subnetworks and the observed difference is by random chance. These results suggest a significant difference between the two sets of subnetworks. This could be attributed by the strong interactions between the FA genes and such strong interactions are not seen in non-informative/non-disease associated gene networks.

LIMITATIONS:

Computational - The results are not completely reproducible because the current python script uses sets and dictionaries that are unordered. The randomization resulting in a random index, however, is reproducible.

The current runtime is high - approximately 576.066 seconds (~9 to 10 minutes) - which could be brought down by further filtering out the network STRING file.

The current approach uses edge counts and does not use the edge weights. Using normalized edge weights to derive densities and comparing the distributions might be more intuitive, but isunexplored in this workflow.

REFERENCES:

1. https://www.ncbi.nlm.nih.gov/medgen/325420
2. Szklarczyk D, Gable AL, Lyon D, Junge A, Wyder S, Huerta-Cepas J, Simonovic M, Doncheva NT, Morris JH, Bork P, Jensen LJ, Mering CV. STRING v11: protein-protein association networks with increased coverage, supporting functional discovery in genome-wide experimental datasets. Nucleic Acids Res. 2019 Jan 8;47(D1):D607-D613. doi: 10.1093/nar/gky1131. PMID: 30476243; PMCID: PMC6323986.
3. McKusick, V.A.: Mendelian Inheritance in Man. A Catalog of Human Genes and Genetic Disorders. Baltimore: Johns Hopkins University Press, 1998 (12th edition)
4. Gustavsen JA, Pai S, Isserlin R et al. RCy3: Network biology using Cytoscape from within R [version 3; peer review: 3 approved]. F1000Research 2019, 8:1774 (https://doi.org/10.12688/f1000research.20887.3)

PSUEDOCODE:

```
// Function to create a dictionary and a set of all genes from the STRING file

Function string_to_diction(string_path):
    INITIALIZE an empty dictionary string_diction
INITIALIZE an empty set string_gene_set
    OPEN string_path AS string_file
    FOR each edge in string file:
            SORT columns of each row
            UPDATE genes in string_gene_set
IF gene in first column is in string_diction keys:
            IF gene in second column and edge value are not inner dictionary of string_diction:
                    UPDATE the inner dictionary of string_diction by appending the edge
ELIF gene in first column is not in string_diction keys:
```

```
                ADD a new key for this row
        RETURN string_diction

// Function to convert input file into loci dictionary, loci set, filtered FA dictionary from string dictionary
Function string_dictionary(input_path,string_path):
        OPEN input_path AS input
                CREATE loci_diction with genes as values

        CREATE fa_set with all genes
        INITIALIZE an empty dictionary fa_diction
        INITIALIZE an empty set fa_string_set
        INITIALIZE an empty dictionary filtered_loci_diction
        OPEN string_path AS string_file
                FOR each edge in string file:
                SORT columns of each row
                IF gene in first column and second column are in fa_set
                        UPDATE genes to fa_string_set
                        IF genes in first column is in fa_diction.keys()
                                IF genes in two columns not in fa_diction inner keys
                                        UPDATE inner dictionary to the key
                        ELIF first column not in fa_diction.keys()
                                ADD inner dictionary to the key

        FOR k,v in loci_diction.items()
FOR element in v
IF element in fa_string_set
Filtered_loci_diction = {k:element}
RETURN loci_diction, filtered_loci_diction,fa_set,fa_string_set,fa_diction

//Function to a dictionary of gene frequencies
Function gene_frequency(string_diction,string_gene_set)
INITIALIZE gene_freq dictionary
FOR outer_key,inner_diction in string_diction.items()
FOR inner_key,inner_value in inner_diction.items()
IF outer_key is in string_gene_set
        INCREMENT gene_freq[outer_key] by 1
IF inner_key is in string_gene_set
        INCREMENT gene_freq[inner_key] by 1
RETURN gene_freq


//Function to creare a nested dictionary of bin sizes for fa genes and non fa genes
Function bin_dictionary(gene_freq,num_bins,filter_bin_data)
        Bin_width = (maximum(gene_freq.values)-minimum(gene_freq.values))/num_bins
```

```
    INITIALIZE an empty dictionary bin_diction
    FOR i in range of num_bins
        Lower_bound = minimum(gene_freq.values)+i*bin_width
        Upper_bound=maximum(gene_freq.values)+i*bin_width
        INITIALIZE outer_keys in bin_diction
        INITIALIZE inner_keys in bin_diction
        FOR gene,freq in gene_freq.items
            IF freq >= lower_bound and freq <= upper_bound
                IF gene is in fa_set
                    APPEND gene to bin_diction fa_genes
                ELSE
                    APPEND gene to bin_diction non_fa_genes

// Function to match the bins for a given FA gene
Function match_fa_nonfa_by_bucket(nested_dict,disease_gene)
    FOR out_g, inner_dict in nested_dict.items()
        ASSIGN out_g to bin_cat
            IF inner_dict is dictionary
                CALL match_fa_nonfa_by_bucket(inner_gene,disease_gene)
                IF result is not NONE
                    RETURN out_g
            ELIF disease_gene in inner_dict
                RETURN bin_cat

// Function to generate two populations of 5000 random subnetworks
Function subnetworks(filtered_loci_dictionary,nnum_iterations,print_subnetwork)
    ASSIGN seed1
    ASSIGN seed2
    INITIALIZE an empty dictionary subnetwork_diction
    INITIALIZE an empty dictionary null_subnetwork_diction
    INITIALIZE an empty dictionary fa_subnetwork
    INITIALIZE an empty dictionary null_subnetwork

    FOR i in range(num_iteratiobs)
        INITIALIZE an empty set iteration_fa_genes
        INITIALIZE an empty set iteration_nonfa_genes
        FOR key,value in filtered_loci_dictionary.items()
            //random index for FA genes
            random_index=(seed1*seed2)%len(value)
            ADD value[random_index] to iteration_fa_genes

            // bin match
            CALL match_fa_nonfa_by_bucket(bin_diction,value[random_index])
```

```
Non_fa_values = bin_diction[bin][‘non_fa_genes’]
random_index_nonfa=(seed1*seed2)%len(non_fa_values)
ADD non_fa_values[random_index_nonfa] to iteration_nonfa_genes
INCREMENT seed1
INCREMENT seed2


INITIALIZE an empty list temp_list
FOR outer_g,inner_d in fa_dictionary.items()
    FOR inner_g,edge_value in inner_d.items()
        IF outer_g in iteration_fa_genes and inner_g in iteration_fa_genes
            APPEND row to temp_list
    ADD len(temp_list) to subnetwork_diction
    IF print_subnetwork is true
        ADD temp_list to fa_subnetwork

INITIALIZE an empty list temp_list_2
FOR outer_g_2,inner_d_2 in string_dictionary.items()
    FOR inner_g_2,edge_value_2 in inner_d_2.items()
        IF outer_g_2 in iteration_nonfa_genes and inner_g_2 in iteration_nonfa_genes
            APPEND row to temp_list_2
    ADD len(temp_list_2) to null_subnetwork_diction
    IF print_subnetwork is true
        ADD temp_list_2 to null_subnetwork

IF print_subnetwork is true
    RETURN subnetwork_diction,fa_subnetwork,null_subnetwork_diction,null_subnetwork
ELSE
    RETURN subnetwork_diction,null_subnetwork_diction

// Function for permutation test
Function permutation_test(subnetwork_diction,null_subnetwork_diction,num_permutations)
    // Calculate mean edges in 5000 FA subnetworks (observed mean difference)
        Mean_fa_sub = mean(subnetwork_diction.values())/len(subnetwork_diction.values())
        Mean_nonfa_sub                                                                  =
mean(null_subnetwork_diction.values())/len(null_subnetwork_diction.values())
        Mean_diff_observed = abs(mean_non_fa_sub - mean_fa_sub)

        MERGE dictionaries subnetwork_diction, null_subnetwork_diction
        INITIALIZE extreme to 0
        INTIALIZE perm_stat_list

        INITIALIZE seed1
        INITIALIZE seed2
        FOR i in range(num_permutations)
            INITIALIZE an empty set
            MERGE subnetwork_diction.keys() and null_subnetwork_diction.keys()
            CREATE a list of all keys swap_labels

            FOR index in range(length(all_network_keys)
                random_index=(seed1*seed2)%length(all_network_keys)
                INCREMENT seed1
                INCREMENT seed2
                ADD random_index to random_index_set

                // swapping without replacement
                Swap_labels[index],swap_labels[random_index] =
swap_labels[random_index],swap_labels[index]

            INCREMENT seed1
            INCREMENT seed2

            Perm_mean1 = sum(all_networks[subnet] for subnet in
swap_labels[0:length(subnetwork_diction.keys())]])/len(subnetwork_diction.keys())
            Perm_mean2 = sum(all_networks[subnet] for subnet in
swap_labels[length(subnetwork_diction.keys()):])/len(subnetwork_diction.keys())

            Perm_stat = as(perm_mean1-perm_mean2)
            APPEND perm_stat to perm_stat_list

            IF perm_stat >= mean_diff_observed
                INCREMENT extreme

        Empirical_p_value = extreme/num_permutations

    RETURN perm_stat_list, extreme, empirical_p_value
```