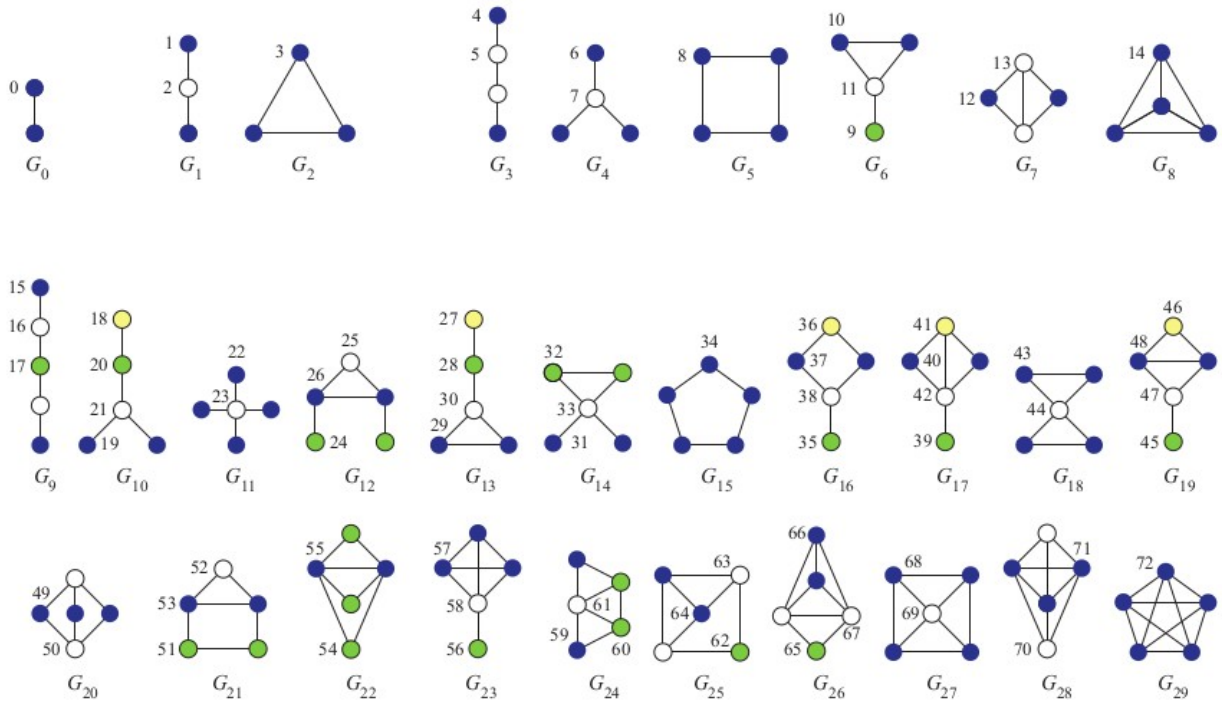
Higher-order connectivity of disease proteins in the PPI network

Aim: To count the orbits of each node in PPI network and find the significant orbits for a disease

We conduct permutation tests, comparing the median of the orbit distribution values for proteins associated with a given disease to the medians for 5,000 random samples of sets of proteins of the same size as the disease pathway. These values are used to calculate p-values for each disease at each orbit position. An orbit position for a disease is considered significant if there is an over-representation of counts in the disease proteins compared to the 99% of random samples (i.e., $\alpha = 0.01$).

Motif signatures of disease proteins.

The analysis of higher-order PPI network structure can be formalized by counting network motifs, which are subgraphs that recur within a larger network. There are 30 possible graphlets of size 2 to 5 nodes. The simplest graphlet is just two nodes connected by an edge, and the most complex graphlet is a clique of size 5. By taking into account the symmetries between nodes in a graphlet, there are 73 different positions or orbits for 2–5-node graphlets, numerated from 0 to 72. For each node in the PPI network we count the number of orbits that the node touches. Motif signature of a protein is thus a set of 73 numbers, hi (i = 0, 1, . . . , 72) representing the number of induced subgraphs the corresponding node is in, in which the node took the i-th orbital position. We use this signature to represent protein's higher-order connectivity in the PPI network.



(a) Node orbits.

Orbit Counting Algorithm (ORCA)

For counting the orbits of graphlets,we used a C++ package ORCA.Orca is an efficient algorithm for counting graphlets in networks. It computes node- and edge-orbits (of 4- and 5-node graphlets) for each node in the network.

Algorithm
The algorithm consists of pre-computation of some data, followed by computation of orbit counts for each node or edge.
1. Pre-computation:
• Count the complete graphlets touched by each node or edge.
• Count the common neighbors of each pair and each connected triplet of vertices.
2. For each node or edge:
• Compute the right-hand sides by enumeration of k − 1 node graphs using the pre-computed data above.
• Solve the system of linear equations.

1. Pre-computation:
• For each node, count the number of complete graphlets in which the node or edge participates. We build cliques of size k from cliques of size k − 1 by maintaining a set of candidate nodes that are adjacent to all nodes in the smaller clique. This procedure is similar to the Bron-Kerbosch algorithm with the difference that we are not interested in maximal cliques but in all cliques of a given size.
Although the theoretical upper bound of the time complexity of this step is O(ed k−2), where d is the maximal node degree, the actual contribution of this step to the total running time is negligible since complete subgraphs (cliques) in sparse networks are rare.
• Compute and store the number of common neighbors for each pair of adjacent vertices. This takes O(ed) time and O(e) space. For computing the orbits in five-node graphlet, we also compute the number of paths of length 2 between each pair of nodes for which such a path exists, and the number of common neighbors for all triplets of connected nodes. This takes O(ed 2) time and O(ed) space.

2. For each node or edge:
• Compute the right-hand sides of the system of the linear equations. Its general form is shown in Equation 6. For four-node graphlets, the sums run over three-node paths or triangles in which the node appears. For five-node graphlets, they run over four-node graphlets that the node touches.
Right-hand sides of equations that sum over the same graphlet can be computed simultaneously. The following code chunk illustrates the computation of the right-hand sides of equations for orbits 13, 16 and 20 for an edge (x, y),

$$e_{13} + 2e_{22} + 2e_{28} + e_{31} + e_{40} + 2e_{44} + 2e_{54} = \sum_{\substack{a,b:\, G[\{x,y,a,b\}]\cong G_3 \\ a\in N(x)\ \wedge\ b\in N(y)}} (c(a) + c(b) - 2),$$

$$2e_{16} + 2e_{20} + 2e_{22} + e_{31} + 2e_{40} + e_{44} + 2e_{54} = \sum_{\substack{a,b:\, G[\{x,y,a,b\}]\cong G_3 \\ a\in N(x)\ \wedge\ b\in N(y)}} (c(x) + c(y) - 4),$$

$$e_{20} + e_{40} + e_{54} = \sum_{\substack{a,b:\, G[\{x,y,a,b\}]\cong G_3 \\ a\in N(x)\ \wedge\ b\in N(y)}} c(x, y).$$

The code for computation of the right-hand sides is as follows.

```
for (int nx = 0; nx < deg[x]; nx++) {
int const &a = adj[x][nx];
if (a == y || adjacent(y, a))
continue;
```

```
for (int ny = 0; ny < deg[y]; ny++) {
int const &b = adj[y][ny];
if (b == x || adjacent(x,b) || adjacent(a,b))
continue;
EORBIT(3)++;
f_13 += (deg[a] - 1) + (deg[b] – 1);
f_16 += (deg[x] - 2) + (deg[y] - 2);
f_20 += tri[xy];
}
}
```

Here, deg[x] and deg[y] are degrees of nodes x and y, and adj[x] and adj[y] are arrays with indices of their neighbors. Function adjacent(u, t) checks whether nodes u and t are adjacent (with a time complexity O(1) or O(log d), depending on whether we construct an adjacency matrix or not) and tri[xy] is the number of triangles spanning over the edge between x and y. Variables f_13, f_16 and f_20 contain the right-hand sides of equations for O 13 , O 16 and O 20 .

The if-clauses check that the edge belongs to E 3 and impose additional constraints as needed. The computation is sped up by using the pre-computed data from the first two steps. In the above case, the right-hand sides of equations for orbits 13, 16 and 20 are (c(a)−1)+(c(b)−1), (c(x)−2)+(c(y)−2) and c(x, y), respectively. The former two are trivial to compute from the graph, and the latter is pre-computed in the second step above.

Note that the orbits for k − 1-node graphlets (as the orbit 3, above) are computed directly.

The time complexity of this step is O(ed k−3).

• Solve the system of equations to obtain orbit counts. Since the system is triangular and the coefficients are fixed, this does not require decomposing or inverting a matrix; the orbits are computed in order, from the higher towards the lower indices, starting with the orbit belonging to the complete graphlet, as for instance, in the following code snippet from the computation of edge orbits.

```
EORBIT(67) = C5[e];
EORBIT(66) = (f_66 - 6 * EORBIT(67)) / 2;
EORBIT(65) = (f_65 - 6 * EORBIT(67));
EORBIT(64) = (f_64 - 2 * EORBIT(66));
EORBIT(63) = (f_63 - 2 * EORBIT(65)) / 2;
EORBIT(62) = (f_62 - 2 * EORBIT(66) - 3 * EORBIT(67));
EORBIT(61) = (f_61 - 2 * EORBIT(65) - 4 * EORBIT(66)
  - 12 * EORBIT(67));
EORBIT(60) = (f_60 - 1 * EORBIT(65) - 3 * EORBIT(67));
```

The system of equations is also rather sparse, with each equation having at most (but usually much less than) eight variables. These nice properties – sparse triangular shape – do not make the algorithm faster since the coefficients are fixed. Even a more general matrix could be inverted in advance and hard-coded into the program. The advantage of triangularity, besides the interpretability of the program, is the numerical accuracy since the entire computation stays in the realm of whole numbers. 4

The system is solved once for each node (or edge), so the time complexity is O(n) (O(e) for edge-orbits).

The total time complexity for all four steps is O(ed k−2 + ed k−3 + ed k−3 + n) for nodes and O(ed k−2 + ed k−3 + ed k−3 + e) for edges. The theoretical complexity is thus O(ed k−2), which is the same as for direct enumeration algorithms. Since large networks are typically sparse, the actual contribution of the first term, which comes from enumerating the cliques with k nodes,

is negligible in practice. Empirical measurements indeed show that the time complexity is proportional to ed k−3 , that is, O(ed) for four-node graphlets and O(ed 2) for five-node graphlets.

USAGE

Preprocessing:
First relabel the edgelist fileshave node ids between 0 to n-1.

Script:
The utility takes four command-line arguments: orca.exe node 5 graph.in orbit-counts.out

1.Orbit type is either node or edge(In this project,we only took the orbit counting of nodes).
2.Graphlet size indicates the size of graphlets that you wish to count and should be either 4 or 5.
3.Input file describes the network in a simple text format. The first line contains two integers n and e - the number of nodes and edges. The following e lines describe undirected edges with space-separated ids of their endpoints. Node ids should be between 0 and n-1. See graph.in as an example.

4.Output file will consist of n lines, one for each node in a graph from 0 to n-1. Every line will contain space-separated orbit counts depending on the specified graphlet size and orbit type.

Computation:

We conduct permutation tests, comparing the median of the orbit distribution values for proteins associated with a given disease to the medians for 5,000 random samples of sets of proteins of the same size as the disease pathway. These values are used to calculate p-values for each disease at each orbit position. An orbit position for a disease is considered significant if there is an over-representation of counts in the disease proteins compared to the 99% of random samples (i.e., $\alpha = 0.01$).

REFRENCES:

1.Hočevar, Tomaž, and Janez Demšar. "Computation of graphlet orbits for nodes and edges in sparse graphs."*Journ. Stat. Soft*71 (2016).

2.http://snap.stanford.edu/pathways/

3.https://github.com/thocevar/orca