CS425: Algorithms for Web-Scale Data

Community Detection in the Protein Interactome

Final Report

Muhammed Çavuşoğlu | Abdurrezzak Efe | Ayşegül Kütük | Mert İnan | Usama Saqib

{m.cavusoglu, abdurrezak.efe, aysegulkutuk, mert.inan, usama.saqib}@ug.bilkent.edu.tr

1. Overview

Community Detection is used to understand the structure of complex networks, and extracting useful information from them. We have used different Community Detection algorithms on Protein-Protein Interaction databases to discover protein communities.

1.1 Problem Statement

In this project, the problem is defined as: finding communities of proteins in the protein-protein interaction datasets using different community detection algorithms and comparing their results. In addition, an ensemble model is proposed to combine different algorithms and choose the best result of communities among them.

1.2 Background Information About Protein Interactomes

Proteins in organisms combine or interact together to form bigger complexes. These complexes regulate systemic functions such as DNA translation, cell transportation, and protein transcription. There are different biological methods to extract these protein-protein interactions (PPI) such as yeasts, and mass spectrometry [1]. Outputs of these methods are protein interactomes.

1.3 Dataset Description

There are two different datasets being used for algorithms. First and most commonly used dataset is the Harvard Worm Interactome [2]. This dataset is chosen as the interactome belongs to the *C. elegans* worm, which has all of its neurons mapped. Hence, protein analysis of the worm can give insights about its neurological functionalities as well. This dataset contains 3,551 edges.

The second dataset is BIOGRID where proteins of 64 different species are present. This dataset contains 65,500 interactions. [3]

2. Implemented Algorithms

Algorithms that we implemented are described in the following subsections.

2.1 Louvain Modularity

Louvain method extracts the community structure of large networks. It uses a heuristic method that is based on modularity optimization. Our implementation is based on the paper “Fast unfolding of communities in large networks”.

2.1.1 Modularity

Modularity is a scalar value between -1 and 1. It measures the density of links inside communities as compared to links between communities. It is used to compare quality of the partitions by different methods, but it is also an objective function. Formula below represents modularity where:

m = total link weight in the network
A_{ij} = weight of the edge between i and j
k_i = total link weight attached to node i
$\delta(c_i,c_j)$ = 1 when i and j are in the same community, 0 otherwise.

$$Q = \frac{1}{2m}\sum_{i,j}\left[A_{ij} - \frac{k_i k_j}{2m}\right]\delta(c_i, c_j)$$

2.1.2 Algorithm

Each pass of the algorithm has two phases. Phase 1 is the "greedy" assignment of nodes to communities, favoring local optimizations of modularity. Phase 2 is the aggregation of found communities in order to build a new network of communities. We repeat this pass until there is no increase in modularity [4].

2.1.3 Implementation

Louvain algorithm is implemented in Python and *C. elegans* (2007) data with 1496 nodes, 1816 edges is used.

2.2 k-Clique Percolation

k-Clique percolation method finds the communities through the means of unifying cliques. Community structure of nodes are based on their memberships to different sets and degrees of their overlappings in different communities.

2.2.1 Algorithm

The algorithm firstly finds all the cliques, then lays them on a clique-clique adjacency matrix to show the number of overlapping nodes. After this step, all the cliques with overlapping numbers less than k are eliminated. Then a connected component algorithm unifies all the remaining cliques.

2.2.1 Implementation

k-Clique method is implemented using the NetworkX library of Python, and its Graph data structure. Nodes and their cliques are stored in a dictionary structure to access in *O(1)* time to the adjacent nodes and find the cliques without going over each node.

2.3 Clustering Affinity Search Technique (CAST)

The CAST algorithm, proposed by Ben-Dor et al. 1999, is a greedy algorithm based on notion of genes close to a cluster C [5]. In Bioinformatics, CAST is used to determine which gene groups are close to each other.

2.3.1 Algorithm

Since the main core of the algorithm serves well to our goal in this project, we implemented the basic structure of CAST while adding some heuristics that provides a good fit for the features of our input and the expected output.

The input to the algorithm includes the pairwise closeness of the genes by edge information given with input and a threshold parameter T. The cluster set under construction is called C_o. When C_o is started, the initial affinity of all genes are 0 since the cluster set is empty at that moment. One heuristic is regarded by the authors in order to start a new cluster which is choosing the gene with the maximum number of neighbors. The affinity level of a gene G, A(G), is defined to be the total similarity value between G and the genes in the cluster set C_o. The similarity is defined to be the average closeness of the gene to the genes in the cluster set. G said to have high affinity if S(G) is less than T and also the smallest value among the values estimated for all the other genes outside the candidate cluster set C_o. Otherwise G is featured to be a gene having low affinity to the current cluster. The algorithm processes on adding the high affinity genes to C_o and ruling out the ones having low affinity. It iterates through all the genes that are assigned to cluster (or not) and the current cluster C_o is closed.

Once one pass on all nodes are completed and all nodes are tested under the constraints, the original CAST algorithm removes all the nodes that are included in the cluster. Since this will kill some considerable results due to the overlapping nodes, we remove only the edges between those nodes

and the nodes having no edge left. This is the first thing while we improve our heuristic approach.

The distances are represented by the edge weights in the original algorithm. Since our input graph is unweighted, we assigned the value of 1 to each and every edge weight.

When setting up a value for the threshold parameter, we did fine tuning since we have the results from other ensembled algorithms and have the chance to test whether the results are reasonable. Best results are captured when the threshold parameter is given to the algorithm with the value of between 3.2 and 3.5.

The algorithm requires us to calculate the average distance value from a gene to the candidate cluster set of genes. For this aim, instead of running shortest path algorithms like Floyd-Warshall and Dijkstra which will cost excessively in time, we established a statistical approach for the calculation of average distances. The details are stated in the upcoming section.

2.3.2 Approximate Distance

In our CAST implementation as it was not feasible to perform single source shortest path algorithms(Dijkstra etc.) or all-pair shortest paths(Floyd-Warshall etc.) we designed a function to return approximate distance of a node to a cluster. Our approach is mostly statistical and intuitive.

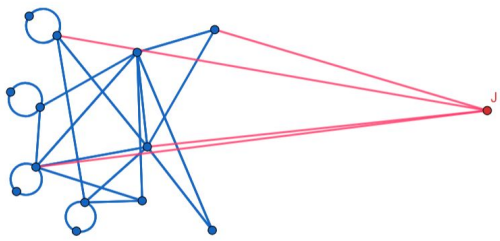

Figure 1 This is a theoretical network to explain the approximate distance.

In Figure 1, assuming blue nodes represent a cluster C and the red point J is the node outside of the cluster that we want to add or not to add to our cluster.

Let's say the distance of J to a point X is defined as $D(J, X)$. To move on we need to know the probability that a pair of two nodes inside the cluster, u and v, have an edge between them. Note that, u and v may be the same node.

$$p = \frac{e+s}{C(n,2)+\gamma n}$$

The probability p is kinda straightforward. Nevertheless, we need to make use of this probability. So, let's define probability that the point J has a distance of 1 to X;

$$P(D(J,X) = 1) = \frac{k}{n}$$

as J has k edges with the nodes that are inside the cluster. However, when we increase the distance(which can be n at most due to Graph Theory) the equation changes.

$$P(D(J,X) = 2) = kp$$

Why? Because X must be a neighbor of a node that J is also a neighbor.

$$P(D(J,X) = 3) = kp(1-p)$$

$$P(D(J,X) = 4) = kp(1-p)^2$$

.

.

.

$$P(D(J,X) = t) = kp(1-p)^{t-2}$$

So, if we want to calculate an approximate distance to any point X that is inside the cluster, we need to calculate the expected value of $D(J,X)$.

$$\overline{D}(J,X) = kp + \sum_{t=2}^{n} tkp(1-p)^{t-2}$$

In our implementation we found out that this gives us a value of 3.2-3.5 range which is very reasonable considering our dataset(Worm Interaction dataset) has only 3-cliques and 4-cliques at most.

2.3.3 Implementation

CAST is implemented using NetworkX library of Python. For the input graph, the dictionary structure is preferred for better efficiency.

2.4 Markov Clustering (MCL)

The Markov Clustering Algorithm is an unsupervised cluster algorithm based on simulation of stochastic flow in graphs, presented in the paper "Van Dongen,S. (2000) Graph clustering by flow simulation". This flow is simulated by using the random walks model, which uses two operators to transform one set of probabilities into another. These two operators known as the Expansion operator and the Inflation Operator.

2.4.1 Expansion

Expansion corresponds to computing random walks of higher length, which means random walks with many steps. It associates new probabilities with all pairs of nodes, where one node is the point of departure and the other is the destination. This is Mathematically equivalent to taking the power of a stochastic matrix using normal matrix product.

Another way to interpret this is as successive linear combinations of the column in a matrix. [ab + cd + ef] value is then the probability of a certain random walk from a single node.

2.4.2 Inflation

Inflation corresponds to taking the Hadamard power of the matrix followed by normalisation, into a stochastic matrix.

The effect of inflation is to boost the higher probabilities and reduce the lower probabilities. This results in a finer granularity of the emergent clusters.

$$\Gamma_r(M_{ij}) = M_{ij}^r / \Sigma_{r,j}(M)$$

Given such a matrix M and a real number r > 1, the column stochastic matrix resulting from inflating each of the columns of M with power coefficient r is written $\Gamma_r(M)$, and Γ_r is called the inflation operator with power coefficient r. Write $\Sigma_{r,j}(M)$ for the summation of all the entries in column j of M raised to the power r (sum after taking powers).

2.4.3 Convergence

The convergence of this method is only guaranteed under very specific conditions, which might not be met by most real world clusters. Therefore, convergent is not mathematically guaranteed. However, in the implementation convergence is enforced by bringing the change in the set of probabilities below a certain threshold. In our implementation this threshold was the one used by numpy.allclose function.

absolute(a - b) <= ($atol$ + $rtol$ * absolute(b))

Here 'a' and 'b' are the matrices and atol is the absolute tolerance parameter and rtol is the relative tolerance parameter.

2.4.4 Interpretation

The clusters correspond to the nonzero values in one row. Therefore for row, r, all nonzero values form one cluster. This accounts for overlapping clusters as well.

2.4.5 Implementation

The algorithm was implemented in Python. In order to speed up implementation a pruning methodology was used. Pruning was used to remove values that were below a certain threshold, since they could then be assumed to not affect the change much.

3 Ensemble Model

Ensembling is a method mostly used in Machine Learning. The core idea of ensembling is to make use of more than one model to solve the same problem and determine a general output according to outputs of those models.

In our project, we used a similar approach to determine best clusters of a given graph.

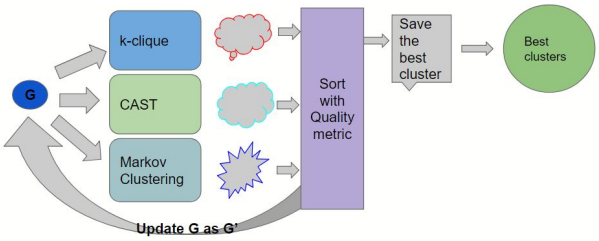

Figure 2 This is a visual depiction of the ensemble model.

As shown in Figure 2, we have an input graph which is undirected and unweighted. We send the graph to all three applicable algorithms; k-clique, CAST and MCL, and get the clusters they output.

Using a metric, which will be explained in detail in 3.1 Metric of Quality section, we choose the best cluster and add it to our current list of clusters up to that point. When our algorithms do not return any other cluster, we end the procedure.

3.1 Metric of Quality

Determining a metric to see how good a cluster is is not trivial. There are many methods to score a cluster according to its density of edges, number of nodes, incoming edges from non-clustered nodes etc. The metric we used is designed for protein interactomes only. Designing the metric, not only did we take self loops into consideration, but also paid attention to how close a cluster to being a clique.

$$q = \sqrt{n}\,\frac{e + (1-\gamma)s}{C(n,2) + \gamma n}$$

Where

q : *quality*

e : *number of edges inside cluster*

γ : *probability that a node has a self loop*

$C(n,2)$: *n choose*

n : *number of nodes inside the cluster*

s : *number of nodes with self loop in the cluster*

What makes it a good metric is that in our datasets we realized there are many proteins who have interactions with themselves, which made us take self loops into consideration.

In the formula we designed the denominator represents the maximum possible number of edges that a cluster with n nodes including probable self loops. Meanwhile, our nominator is the sum of number of edges between nodes and weighted number of self loops in that cluster.

Also, when γ becomes 0, which is when there is no self loops in our dataset, having a self loop gets utmost importance. On the other hand, when γ is 1, which is when every single node has a self loop, we give simply no importance to self loops as it is too unimportant and common.

However, besides all the points mentioned, a little downside may be in play as

our quality metric do not care about the size of the cluster it scores. After all, the number of required edges for a clique increases with the square of the number of nodes. Thus, for a bigger cluster to be a clique it becomes harder.

4. Results

4.1 Louvain Results

Modularity value for *C. elegans* (2007) data is converged to 0.79816826, and the algorithm has made 7 passes. We have detected 197 communities. The algorithm did not return random results, we obtained same number of passes and modularity values for every run.

4.2 k-Clique Results

Two different datasets were used with the k-clique algorithm. BIOGRID data was used to test for all the possible clusterings of all available protein interactions in all organisms while Harvard *C. elegans* data was used to compare with the other algorithms.

When k is chosen to be 4, then there are only five communities as the output of the k-clique. However, as it is also mentioned by Palla et al., five communities are not biologically feasible [6]. Yet, when k is chosen to be 3, the results are biologically feasible as in Table 1.

4.3 CAST Results

Due to the issues remained unresolved during the implementation, considerable results couldn't be generated. Also since a complete implementation of the algorithm could not be found as a re-use source to assess, the problems could not be eliminated, and so the testing with the CAST results is unfortunately not a subject matter.

Table 1 This table shows the top 5 clusters and the proteins inside them found by the k-clique algorithm.

Protein Names in Found Communities
Biggest cluster is omitted due to lack of space. Shown in Appendix A.
'Y69H2.3', 'R02F11.1', 'T28A11.11', 'C05G6.1', 'Y39B6A.1', 'C23H3.3', 'F30H5.3', 'ZK849.2', 'R05F9.10', 'F54B11.7', 'M02G9.1', 'T21D12.11', 'B0507.1', 'T23F1.6', 'T01D1.6', 'F29G6.3', 'T01B7.8', 'Y65B4A.7', 'F48G7.10', 'DH11.4', 'C03A7.4', 'K08E7.5', 'AC3.3', 'H04J21.3', 'B0024.14', 'T22H2.5', 'Y22F5A.4', 'F43G9.11', 'C17G10.5', 'C03A7.14', 'C37C3.6', 'B0205.3', 'Y69H2.10', 'ZK1067.7', 'Y17G7B.14', 'F35A5.3', 'F53A9.6', 'R09B5.5', 'T21B6.3', 'ZK1053.5'
'F44G3.9', 'F38A6.1', 'T01G9.6', 'F59C12.3', 'C49A1.4', 'M04G12.1', 'F57C9.4', 'K08F8.2', 'F54D10.7', 'T11B7.1', 'T04H1.2', 'C38D9.1', 'C06A5.9'
'C05D9.1', 'C06E1.1', 'D2013.2', 'Y38C1AA.7', 'F17E9.5', 'W06A7.3'
'F32D1.1', 'Y57A10A.8', 'W10D5.3', 'F29B9.6', 'K12C11.2', 'W04D2.1'

4.4 MCL Results

The data on which MCL was tested on C.elegans dataset. The top 5 clusters are shown in Appendix B. As can be seen in the appendix there is significant overlap between the two largest clusters.

4.4.1 Jaccard Similarity between clusters

sim(Cluster1, Cluster2) = 0.9322

4.5 Ensembling Results

The ensembling algorithm gave results for only k-clique and MCL algorithms due to CAST algorithm being incapable of generating any results and Louvain being incompatible with ensembling.

Score of the communities are the sum of the scores of the clusters that our algorithm returned. We observed that ensembling allowed us to get a higher community score than what we get from only k-clique or MCL alone.

4.5.1 Score of Communities

A sample result table can be seen as below.

1st CLUSTER: [3, 42, 43, 15, 943, 17, 18, 19, 20, 21, 22, 1087]FROM: k_clique
SCORE: 4.897435897435896

2nd CLUSTER: [264, 74, 235, 209, 242]FROM: k_clique
SCORE: 3.297872340425532

3rd CLUSTER: [199, 209, 180, 244, 251]FROM: mcl
SCORE: 3.234042553191489

4th CLUSTER: [147, 748, 501, 622]FROM: k_clique
SCORE: 3.0270270270270268

5th CLUSTER: [200, 889, 34, 95]FROM: mcl
SCORE: 2.8648648648648645

The sum for a sample run of ensemble is:
97.317423445623424
Where;
k-clique's score:
64.843243249823286
MCL's score:
45.273248645832462

This result presents some possibilities. One of them is that our metric to score a cluster favors k-clique; meanwhile, the other possibility is that k-clique literally gives us better results than MCL algorithm.

Nevertheless, as our sources imply, there is no obvious way to say that a cluster is good or bad. It only depends on ground truth and the metric used.

5 Comparing Algorithms

Comparison of algorithms in a biological network is still problematic in the literature [7]. As labeling the clusters is a biological duty rather than a computational one. Hence, certain benchmarks and methods are mentioned here, yet the overall quality of the implemented methods are still susceptible to the data itself.

5.1 Methods & Benchmarks

Several methods exist in the literature. Silhouette indexing is one of the most used as it is an internal metric [8]. The algorithms can also be compared according to the GN and LFR benchmarks. We have used the ensembling algorithm to choose the best method.

5.2 Results of Comparison

Louvain method and CAST cannot be compared to other two as their outputs are not compatible with them. Meanwhile, our metric say that k-clique gives better results when it comes to clustering. However, as mentioned before, it may be case that our metric favors k-clique algorithm.

6. Challenges & Solutions

One of the challenges we faced was that comparing our output clusters with ground truth would be sheer time consuming. We solved this problem by deciding on a metric that can score our outputs.

Moreover, dataset obtained from different sources tend not to have the same structure. Therefore preprocessing of the dataset would be required for comparison This is quite difficult, since understanding the curated datasets require technical biological background.

Clusters do not correspond directly to biological families and superfamilies and the connection can only be meaningfully drawn with the appropriate biological background. Therefore, it was not possible for us to use our results to predict structural modules which the requisite biological precision.

7. Conclusion

All in all, in this project, different community detection algorithms have been implemented to detect protein communities in the protein interactome.

References

1. T. Berggård, S. Linse, and P. James, "Methods for the detection and analysis of protein–protein interactions," Proteomics, vol. 7, no. 16, pp. 2833–2842, 2007.

2. "Worm Interactome version 8," Worm Interactome Version 8. [Online]. Available: http://interactome.dfci.harvard.edu/C_elegans/index.php. [Accessed: 09-May-2018].

3. M. T. Lab, "BioGRID Interaction Database," BioGRID Search for Protein Interactions, Chemical Interactions, and Genetic Interactions. [Online]. Available: https://thebiogrid.org/. [Accessed: 09-May-2018].

4. "Fast unfolding of communities in large networks". https://arxiv.org/abs/0803.0476. [Accessed: May 4, 2018].

5. "Details of the Clustering Algorithms Supplement to the Paper "Validating Clustering in Gene Expression Data"(to appear in Bioinformatics)" http://faculty.washington.edu/kayee/cluster/algs.pdf [Accessed: May 5, 2018].

6. G. Palla, I. Derényi, I. Farkas, and T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society," Nature, vol. 435, no. 7043, pp. 814–818, 2005.

7. P. Chejara and W. W. Godfrey, "Comparative analysis of community detection algorithms," 2017 Conference on Information and Communication Technology (CICT), 2017.

8. P. J. Rousseeuw, "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis," Journal of Computational and Applied Mathematics, vol. 20, pp. 53–65, 1987.

APPENDIX A

Biggest community is shown below as found by the k-clique algorithm.

'W05H7.4', 'F59C12.3', 'Y49E10.14', 'C37A2.5', 'T22B2.4', 'ZK666.2', 'R11E3.6', 'C30F12.4', 'Y55F3AM.3', 'Y53C12B.3', 'M01E11.7', 'K09B11.9', 'Y73B6BL.33', 'C06G1.5', 'Y37D8A.21', 'F36D1.4', 'F11G11.5', 'T01D1.2', 'H06I04.1', 'F54C9.11', 'DH11.4', 'ZK121.2', 'Y57G11C.24', 'T21G5.5', 'Y119C1A.1', 'C06A8.5', 'Y77E11A.7', 'ZK1067.6', 'F26B1.2', 'R74.5', 'T11B7.4', 'B0336.6', 'T11B7.1', 'D2089.4', 'ZK849.2', 'F17A2.5', 'F42A6.9', 'Y39B6A.11', 'F52D2.4', 'Y46G5A.31', 'W01A11.4', 'W04D2.1', 'C06C3.1', 'Y65B4BR.4', 'D1046.1', 'K07D4.7', 'C06E7.4', 'C47G2.2', 'F09F7.8', 'F54D5.15', 'Y37H2A.1', 'F44D12.1', 'Y80D3A.3', 'F14F3.1', 'W02D3.11', 'T26A8.4', 'C27B7.4', 'C52B11.2', 'T04H1.2', 'R05F9.1', 'ZK863.7', 'H28G03.2', 'C18H9.7', 'T17H7.4', 'T22A3.3', 'Y44E3A.6', 'C49A1.4', 'F42H10.7', 'W06F12.1', 'F38B2.1', 'F35H8.5', 'Y54H5A.3', 'F46A9.6', 'Y79H2A.11', 'F01G10.2', 'W10C8.2', 'F28F5.3', 'R01E6.5', 'F52D10.3', 'F32B4.4', 'T05C12.6', 'M142.6', 'C09G1.4', 'Y54E2A.3', 'R02F2.5', 'Y42H9AR.1', 'T09A5.12', 'Y63D3A.5', 'C16B8.3', 'Y53H1A.1', 'F59E12.9', 'F10C1.7', 'C15C7.5', 'Y63D3A.4', 'Y40C5A.1', 'F15C11.1'

APPENDIX B

Cluster1 size: 68

'B0024.14', 'B0024.14', 'T21D12.11', 'B0024.14', 'F12F6.7', 'B0205.3', 'B0280.3', 'B0304.1', 'B0348.6', 'C13F10.2', 'M03B6.1', 'C24G6.4', 'C02F5.9', 'Y59A8B.7', 'C03C10.4', 'C05C10.5', 'C05D9.1', 'R03D7.4', 'C06G1.5', 'C31H1.6', 'W02D3.9', 'C09G1.4', 'ZC8.4', 'K07F5.11', 'C12D8.1', 'C13F10.2', 'W10C8.2', 'C17D12.7', 'Y41E3.7', 'T24H10.6', 'C18B2.2', 'F52E4.7', 'C18D4.8', 'E02H4.2', 'C18E3.9', 'C18E9.8', 'K10C9.7', 'T20D4.17', 'C18H9.7', 'F01G10.2', 'Y37H2A.1', 'C25A1.4', 'Y59A8B.10', 'C33G8.8', 'Y69H2.3', 'F42H10.5', 'C47C12.3', 'W10C8.2', 'T23G5.3', 'C47E12.9', 'C50F4.1', 'F54G8.4', 'C50F4.1', 'C52B11.2', 'W05H7.4', 'C52B11.2', 'W10C8.1', 'C52B11.2', 'Y37D8A.21', 'C52B11.2', 'Y54H5A.3', 'C52E4.3', 'F17E9.5', 'F01F1.9', 'F01G10.2', 'F17E9.5', 'F07B7.2', 'K09B11.9'

Cluster2 size: 65

'B0024.14', 'B0024.14', 'T21D12.11', 'B0024.14', 'F12F6.7', 'B0205.3', 'B0280.3', 'B0304.1', 'B0348.6', 'C13F10.2', 'M03B6.1', 'C24G6.4', 'C02F5.9', 'Y59A8B.7', 'C03C10.4', 'C05C10.5', 'C05D9.1', 'R03D7.4', 'C06G1.5', 'C31H1.6', 'C09G1.4', 'ZC8.4', 'K07F5.11', 'C12D8.1', 'C13F10.2', 'Y41E3.7', 'T24H10.6', 'C18B2.2', 'F52E4.7', 'C18G1.2', 'C18D4.8', 'E02H4.2', 'C18E3.9', 'C18E9.8', 'K10C9.7', 'C18H9.7', 'F01G10.2', 'Y37H2A.1', 'C25A1.4', 'Y59A8B.10', 'C33G8.8', 'Y69H2.3', 'F42H10.5', 'C47C12.3', 'W10C8.2', 'T23G5.3', 'C47E12.9', 'C50F4.1', 'F54G8.4', 'C50F4.1', 'C52B11.2', 'W05H7.4', 'C52B11.2', 'W10C8.1', 'C52B11.2', 'Y37D8A.21', 'C52B11.2', 'Y54H5A.3', 'C52E4.3', 'F17E9.5', 'F01F1.9', 'F01G10.2', 'F17E9.5', 'F07B7.2', 'K09B11.9'

Cluster3 size: 49

'Y69H2.3', 'B0034.1', 'Y23H5A.4', 'Y38C1AA.7', 'B0035.1', 'B0205.7', 'B0393.4', 'F08C6.3', 'B0554.4', 'T25B6.4', 'T28D9.10', 'C03C10.4', 'F32B4.4', 'R102.5', 'C04F1.3', 'C04F12.3', 'C05G6.1', 'F54D10.7', 'C06A5.9', 'F55H2.7', 'C06A5.9', 'F59B8.1', 'C06A5.9', 'C06A5.9', 'R07G3.7', 'C06A5.9', 'R01E6.5', 'W09C2.1', 'R05F9.10', 'C36B1.4', 'M04G12.1', 'C40H1.3', 'C52B11.2', 'C50E3.12', 'W09C2.1', 'C50E3.5', 'ZC455.1', 'C50F4.1', 'ZK849.2', 'C52E4.3', 'F01F1.1', 'ZK849.2', 'F01F1.4', 'Y55F3AM.15', 'F01G10.2', 'H06I04.1', 'F01G10.2', 'K02E7.9', 'W10D5.3'

Cluster4 size: 27

'B0024.9', 'B0205.3', 'B0280.3', 'F08C6.3', 'B0554.4', 'T07G12.6', 'C01G10.11', 'F46C8.5', 'T01G9.6', 'C06G1.5', 'C06G1.5', 'R05F9.10', 'C29E4.1', 'Y40B10A.2', 'C29E4.5', 'F44G3.9', 'K09H11.1', 'C44B9.2', 'C49A1.4', 'C52B11.2', 'C53C7.3', 'D2085.3', 'K12C11.2', 'D2089.4', 'D2089.4', 'EEED8.12', 'W10D5.3'

Cluster5 size: 19

'B0035.1', 'T25B6.4', 'K12C11.2', 'C03C10.4', 'F32B4.4', 'R01E6.5', 'C12D8.1', 'W09C2.1', 'C44F1.5', 'C13F10.2', 'F22B7.13', 'C38D9.1', 'C18H9.7', 'C50E3.12', 'W09C2.1', 'C50E3.5', 'R166.1', 'W10D5.3', 'F07C6.4'