## Genome neighborhood analysis using Bipartite Modularity

Conservation of gene order across long evolutionary distances is a strong indicator of functional relationships among genes. These orders are found in structures such as operons, gene clusters, and genomic islands. Notable examples include the tryptophan biosynthesis pathway and the str ribosomal protein operons, which are conserved in both bacteria and archaea. The amount of genomic sequence data available today provides valuable insights into the functions of uncharacterized proteins and the pathways they are involved in. Identifying conserved gene orders in genomes through high-throughput methods is promising for predicting protein roles in specific pathways or systems.

Various tools analyze gene neighborhood conservation and integrate this data with other metrics for functional association. The most famous example of this is the identification new CRISPR-Cas systems by first identifying all Cas9 proteins and then analyzing the closely related gene families. However, many of these tools are restricted to certain genomes or require local genome databases. Furthermore, most methods focus on identifying an anchor gene and then extend to adjacent flanking genes. They use downstream BLASTp or HMM searches with sets of homologous flanking genes and cluster these proteins to find the union of gene clusters between genomes. This approach has several limitations:

1. Misannotated anchor genes with many false positives, leading to unrelated genomic regions.
2. Lack of functional annotation and clustering in gene neighborhood analysis, causing the grouping of functionally unrelated sequences despite sequence homology.
3. Absence of tools to further subclassify and identify closely related gene modules for these anchor genes. For example, the str ribosomal protein could be in the same neighborhood as functionally distinct protein clusters involved in different functions.

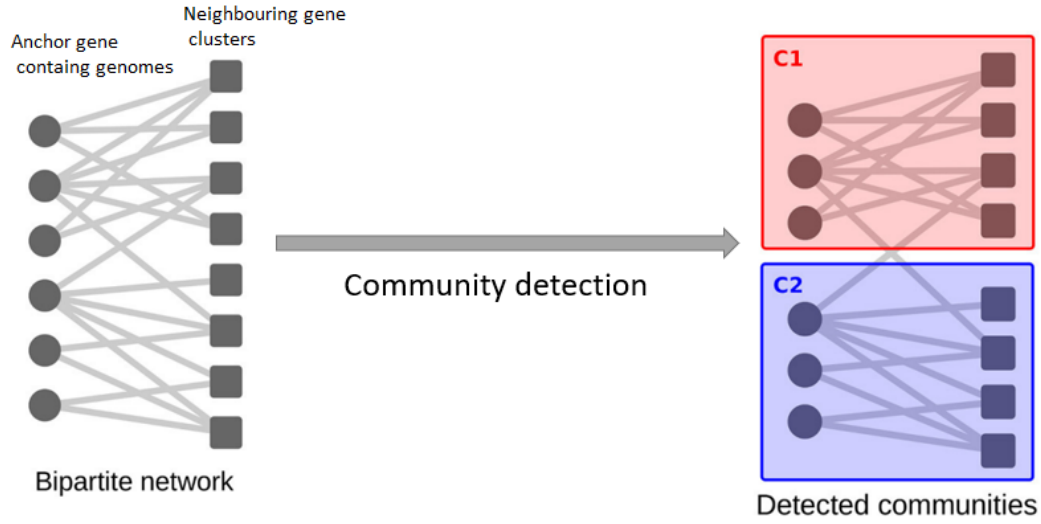
To address these limitations, I propose developing a novel tool that combines homology- and function-based approaches to identify proteins functionally related to a unique set of gene families (e.g., class A bla, CRISPR-Cas9 orthologs). This tool will help find other gene clusters or modules closely related to these homologous protein families.

My approach builds on methods previously used to reveal information about protein-protein interactions in protein complexes. Recent advances in protein sequencing technology have provided proteome-scale protein-protein interaction (PPI) data for many species. For global analysis of large-scale PPI data, the network-based approach is widely used. Since the modular nature of biological networks was demonstrated, detecting communities or modules—sets of nodes more densely interconnected among themselves than with the rest of the network—has become a popular analysis method. Many community-detection methods have been suggested in the past decade, with the modularity optimization approach being the most widely used for analyzing various networks. Modularity is a quality measure used to evaluate a given community structure and is defined as follows:

$$Q = \sum_c \left( \frac{l_c}{m} - \left( \frac{d_c}{2m} \right)^2 \right),$$

where $l_c$ is the number of intra-community edges of community $c$, $d_c$ is the sum of degrees of nodes in community $c$, and $m$ is the total number of edges in the network. Conceptually, the modularity indicates the difference between the fraction of intra-community edges observed and the expected value in a random network. Most community-detection methods have focused on maximizing the modularity of a given network based on the assumption that the community structure with the maximum $Q$ would correspond to the optimal community structure to extract hidden information.

In gene neighborhood analysis, I'll use a bipartite network comprising orthologous anchor gene-harboring bacteria (node type U) and neighboring gene clusters (node type V) as nodes. For community detection within this network, I used the Leiden algorithm, a popular method known for its speed and accuracy in identifying clusters of nodes that are more densely connected internally than with the rest of the network. This algorithm improves upon the Louvain method by delivering better results in terms of modularity and more effectively handling disconnected communities.




The Leiden algorithm is a community detection algorithm that optimizes modularity. When applying the Leiden algorithm to bipartite graphs, we typically use the Modularity quality function, specifically adapted for bipartite graphs. The modularity for bipartite graphs is slightly different from that for unipartite graphs:

**Bipartite Modularity**

The bipartite modularity score is calculated using the formula below:

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

Where:

- $A_{ij}$ is the adjacency matrix.
- $k_i$ and $k_j$ are the degrees of nodes $i$ and $j$.
- $m$ is the total number of edges.
- $\delta(C_i, C_j)$ is 1 if nodes $i$ and $j$ are in the same community and 0 otherwise.

The Leiden algorithm seeks to maximize this modularity function through several steps:

1- Initialization: Each node starts in its own community.
2- Local Moving Phase: Nodes are moved between communities to maximize the increase in modularity. This is done iteratively until no further improvements can be made locally.
3- Refinement Phase: The algorithm refines the community structure to further optimize modularity.
4- Aggregation Phase: Communities are aggregated into super nodes, creating a new, smaller graph. The process is then repeated on this aggregated graph.

In the case of gene neighborhood analysis, the workflow for detecting communities of orthologous genes (anchor genes) with similar gene content is as follows:

1. **Identify orthologous genes in bacterial genomes:** Perform an HMM search with specific cutoffs to identify the orthologous gene of interest in GTDB bacterial genomes archive.
2. **Gene Neighborhood Acquisition (GNA):** Use the GNA program to acquire 20 genes upstream and downstream of the anchor gene of interest in all bacterial genomes containing the anchor gene. This step includes using Prodigal (Prokaryotic Dynamic Programming Gene-finding Algorithm), a software tool for predicting protein-coding genes in prokaryotic genomes (bacteria and archaea).
3. **Clustering with CD-HIT:** Use the CD-HIT program to cluster a FASTA file containing all the orthologous genes and their neighboring upstream and downstream genes.
4. **Functional Annotation with Kofamscan:** Use the Kofamscan program for functional annotation of the FASTA file by assigning KEGG Orthology (KO) identifiers to all protein sequences.

5. **Mapping and Similarity Calculation:** Map the Kofamscan KEGG KO IDs for all sequence clusters from the CD-HIT results. Calculate the cosine similarity score based on the mapped KEGG KOs, and combine CD-HIT clusters if they share similarity scores above 0.5. This results in sequence clusters with highly similar neighboring genes in both function (KOs) and homology (CD-HIT identity).
6. **Constructing the Bipartite Network:** Construct a bipartite network consisting of orthologous gene-harboring bacteria (node type U) and neighboring gene clusters (node type V) as nodes. Filter out clusters of neighboring genes with fewer than 9 members.
7. **Community Detection with Leiden Algorithm:** Use the Leiden modularity optimization to identify communities of anchor gene-harboring bacteria and their corresponding clusters of neighboring genes.
8. **Summary Output:** Write the summary of gene content for all identified communities and gene modules (highly similar and redundant neighboring gene repeats) to output files

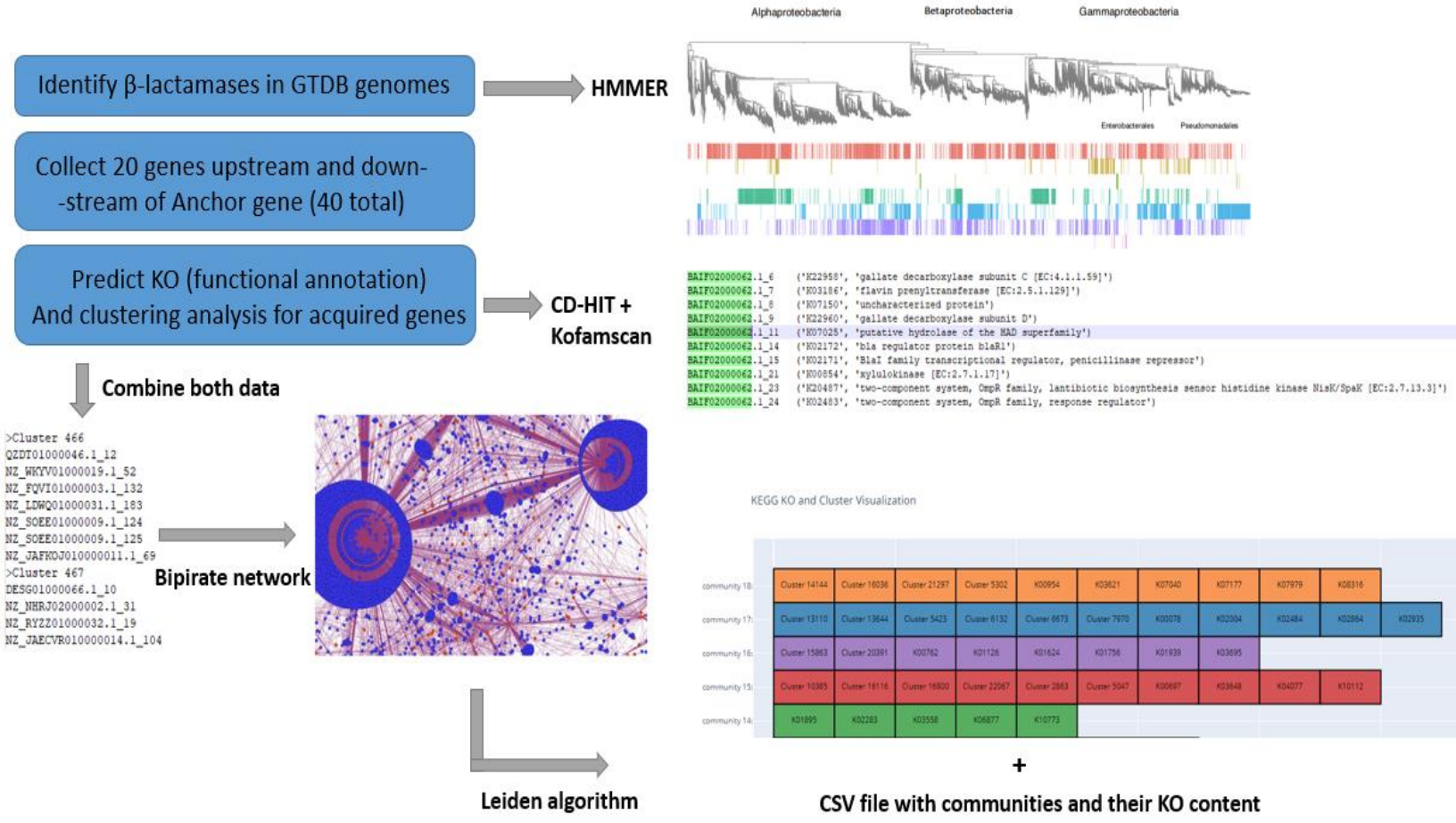


**Figure 1. Genome neighborhood analysis using Bipartite Modularity**

**Detailed summary:**

1. The GNA code takes a FASTA file containing all identified orthologs for a gene of interest in the Prodigal tool gene format. For example:

   EQ973338.1_555 # 662064 # 662654 # 1 # ID=17_555;partial=00;start_type=ATG;rbs_motif=GGxGG;rbs_spacer=5-10bp;gc_cont=0.538

   Here, EQ973338.1_555 represents the gene of interest. The first two numbers, 662064 and 662654, indicate the starting and ending nucleotide positions of our gene of interest. This information is used to map the genomic region of our gene and to collect 20 genes upstream and downstream with lengths above 100 amino acids.

2. The following commands predict the functional KO and sequence clustering based on identity for all collected genes upstream and downstream of the anchor ortholog genes of interest.

**GNA Command:**

python GNA.py -a Class_A_bla.faa -d N:\Naoki_project\prodigal_result -o Class_A_bla_10_genes.txt

- -a: Specifies the input anchor sequence file containing the identified orthologs FASTA file.
- -d: Specifies the path to the directory containing the complete genome files.
- -o: Specifies the output file where the results of collected upstream and downstream gene sequences will be saved.

**CD-HIT Command:**

cd-hit -I input.txt -o output.txt -c 0.4 -n 2 -d 0 -T 20 -M 55000 -s 0.7

- -I: Input file for all collected genes upstream and downstream of anchor ortholog genes of interest.
- -o: Output clustering file for the collected genes.
- -c: Sequence identity threshold for clustering, set to 40%.
- -n: Word length for the similarity search, set to 2 for higher sensitivity.
- -d: Length of the description in the output FASTA header, set to 0 for full headers.
- -T: Number of threads (CPUs) to use for clustering.
- -M: Memory limit in megabytes.

- -s: Minimal length coverage of the shorter sequence relative to the longer sequence, set to 70%.

**Kofamscan Command:**

./exec_annotation -o output.txt -p /mnt/n/Kofam_scan/profiles -k /mnt/n/Kofam_scan/ko_list --cpu 4 collected_gene.txt

- -o: Specifies the output file for the annotation results.
- -p: Path to the directory containing the KO HMM profiles.
- -k: Path to the KO list file with KEGG Orthology identifiers.
- --cpu: Number of CPU threads to use for the annotation process.
- collected_gene.txt: Input file containing the protein sequences to be annotated.

3. The GBA code performs community detection and module identification related to the identified ortholog gene of interest (Figure 1).

**GBA Command:**

python GBA.py -a inputs/collected_sequences.fasta -o QnR_40_genes --kofamscan inputs/kofamscan_predictions.txt --cdhit inputs/cdhit_clustering.txt

- -a: Specifies the input collected sequence FASTA file from the GNA code output.
- --kofamscan: Specifies the input Kofamscan prediction file from the collected sequence FASTA file.
- --cdhit: Specifies the input CD-HIT clustering file from the collected sequence FASTA file.