Lab activity: analysis of transcriptional regulatory networks

1. Session 3: network motifs analysis

1.1. Introduction

In this session, we will explore the concept of network motifs (small, recurring patterns of interconnections) in the context of transcriptional regulatory networks (TRNs). Given a small number of interconnected nodes in a network (e.g. 3 nodes), some interconnection patterns may occur more than expected by chance. We call these patterns *network motifs*. Network motifs are fundamental building blocks of complex networks and are thought to play key roles in the function and evolution of biological systems.

In TRNs, nodes represent genes or operons, and directed edges represent regulatory interactions, typically from a transcription factor (TF) to its target gene (TG). Certain motifs, such as the feed-forward loop (FFL) and the bi-fan (BF), are found more frequently in biological networks than in randomized counterparts, suggesting they confer specific functional advantages.

The **Feed-Forward Loop (FFL)** is a three-node motif where a TF X regulates a second TF Y, and both TFs also regulate a target gene Z. This motif implement dynamic behaviors in gene regulation that may be advantageous to the cell.

The **Bi-Fan (BF)** is a four-node motif where two TFs (X and Y) jointly regulate two target genes (Z and W). This motif is associated with combinatorial regulation and integration of multiple signals.

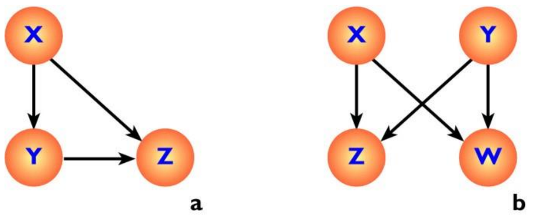

Figure 1 – Schematic representation of the **(a)** feed-forward loop (FFL) and **(b)** the bi-fan (BF) network motif.

In this activity, you will implement algorithms to detect these motifs in the two versions of the *E. coli* TRN obtained in session 1. You will then analyze and interpret the biological significance of your findings.

1.2. Network motif analysis

In this third session, we will use the graphs obtained from the RegulonDB *E. coli* data (`Ecoli_TRN.graphml`, `Ecoli_operon_TRN.graphml`) and study their network motifs.

1.2.1. Feed-Forward Loop (FFL) Motif Detection

1. **(1.5 points)** Implement a function that, given a directed graph, detects all feed-forward loop (FFL) motifs. The function should return a list of tuples, each representing a motif as `(X, Y, Z)`, where X regulates Y, X regulates Z, and Y regulates Z.

   Important: Your implementation must be efficient (e.g., using adjacency matrices or optimized data structures) to ensure that motif detection completes in a reasonable amount of time when executed over the *E. coli* networks.

```python
def find_ffl_motifs(G: nx.DiGraph) -> List[Tuple[str, str, str]]:
    """
    Detect feed-forward loops (FFLs) in the directed graph given as input.

    Parameters
    ----------
    - param: G : Networkx digraph
        Graph to analyze
    - return: list
        List of detected motifs, each represented as a tuple (X, Y, Z).
    """
    # ------- IMPLEMENT HERE THE BODY OF THE FUNCTION ------- #
    pass
    # ----------------- END OF FUNCTION --------------------- #
```

1.2.2. Bi-fan Motif Detection

2. **(1.5 points)** Implement a function that, given a directed graph, detects all bi-fan motifs. The function should return a list of tuples, each representing a motif as `(X, Y, Z, W)`, where X and Y regulate both Z and `W`.

   Important: Recall that the implementation must be efficient (should be able to process *E. coli* networks in a reasonable amount of time).

```python
def find_bf_motifs(G: nx.DiGraph) -> List[Tuple[str, str, str, str]]:
    """
    Detect bi-fan (BF) motifs in the directed graph given as input.

    Parameters
    ----------
    - param: G : Networkx digraph
        Graph to analyze
    - return: list
        List of detected motifs, each represented as a tuple (X, Y, Z).
    """
    # ------- IMPLEMENT HERE THE BODY OF THE FUNCTION ------- #
    pass
    # ----------------- END OF FUNCTION --------------------- #
```

1.2.3. Motif clustering

3. **(0.5 points)** Implement a function that, given a graph and a list of motifs (FFL or bi-fan), constructs the subgraph formed by all edges participating in at least one motif, and returns the list of weakly connected components (motif clusters) in this subgraph.

```python
def get_motifs_clusters(G: nx.DiGraph, motifs: list, \
                        mtype: str="ffl") -> List[Set[str]]:
    """
    Analyse the subgraph created from edges belonging to motifs.

    Parameters
    ----------
    - param: G : Networkx digraph
        Graph to analyze
    - param: motifs : list
        List of motifs
    - param: mtype : str
        Motif type ("ffl" or "bf")
    - return : list
        List of weakly connected components in the motifs subgraph
    """
    # ------- IMPLEMENT HERE THE BODY OF THE FUNCTION ------- #
    pass
    # ----------------- END OF FUNCTION --------------------- #
```

1.2.4. Comparison with Random Graph Models

4. **(1 point)** Implement a function that, given an original directed graph and a number of random graphs to generate, creates random graphs with the same degree distribution as the original directed graph using the (directed) configuration model.

   Then, for both the original graph and the generated random graphs, the function must print (for FFL and BF motifs):

   a. the number of motifs

   b. the number of nodes participating in motifs

   c. the number of motif clusters

   d. the distribution of motif cluster sizes

   The function should analyze both the original graph and the random graphs (reporting mean and standard deviation for the random models) and print all results.

```python
def compare_with_random_graphs(G: nx.DiGraph, n_random: int):
    """
    Compare motif statistics of the original graph with random graphs generated by
the directed configuration model.
    For each motif type (ffl and bf), prints:
        - Number of motifs
        - Number of nodes participating in motifs
        - Number of motif clusters
        - Distribution of motif cluster sizes
    Parameters
    ----------
    - param: G : Networkx digraph
        Original graph to analyze
    - param: n_random : int
        Number of random graphs to generate
    """
    # ------- IMPLEMENT HERE THE BODY OF THE FUNCTION ------- #
    pass
    # ----------------- END OF FUNCTION --------------------- #
```

1.2.5. Bi-fan overlap analysis

5. **(0.5 points)** Implement a function that, given a list of bi-fan motifs obtained with `find_bf_motifs` and two node names (`locus_tag` identifiers), returns a tuple with: the number of bi-fan motifs in which the first TF participates, the number of bi-fan motifs in which the second TF participates, and the total number of bi-fan motifs in which both motifs participate.

```
def compute_bf_overlap(bifan_list : list, TF1_locus_tag : str, \
                       TF2_locus_tag : str) -> Tuple[int, int, int]:
    """
    Compute the overlap between bi-fan motifs in which two TFs participate.
    Returns a tuple containing:
        - Number of bi-fan motifs in which the first TF participates
        - Number of bi-fan motifs in which the second TF participates
        - Number bi-fan motifs in which both TFs participate

    Parameters
    ----------
    - param: bifan_list : list
        List of all bi-fan motifs identified in a network
    - param: TF1_locus_tag: str
        Locus tag of motif 1
    - param: TF1_locus_tag: str
        Locus tag of motif 1
    - return : tuple
        Tuple containing:
            - TF1 number of BF motifs
            - TF2 number of BF motifs
            - Number of shared BF motifs
    """
    # ------- IMPLEMENT HERE THE BODY OF THE FUNCTION ------- #
    pass
    # ----------------- END OF FUNCTION --------------------- #
```

1.3. Questions to answer in the report

1. **(0.5 points)** Compare the abundance of FFL and BF motifs between the gene-level and operon-level networks. Which network has more motifs of each type? What might explain the differences observed?

2. **(0.5 points)** Report on the distribution of motif cluster sizes for both motif types and both networks. What do you observe? Are motif clusters typically small or large? Does this align with the results you obtained when analyzing the overall properties of the *E. coli* transcriptional regulatory network in session 2?

3. **(0.5 points)** Are there any coincidences between any secondary clusters you obtain for FFL and BF in the *E. coli* (no operon) network? Look up the genes involved to see if you can explain why these genes are set apart from the rest of the network.

4. **(1 point)** Compare the motif statistics of the *E. coli* networks to those of a randomized network with the same degree sequence (e.g., using the configuration model). Are motifs still overrepresented? What does this tell us about the evolutionary design of the TRN? How do operons contribute to this?

5. **(0.5 points)** The *araB* ('b0063') gene is the lead gene of the *araBAD* operon, which encodes three proteins involved in the import and degradation of arabinose (a sugar used by *E. coli* as a secondary energy source (the primary one being glucose)). As in the case of the *lac* operon, the *araBAD* operon is maximally expressed when there is there is arabinose, but no glucose, around. That is, its promoter implements a `(NOT(glucose) AND arabinose)` logic. Analyze the interactions of the *araB* gene in the no-operon *E. coli* network. What TFs interact to regulate *araB*? Look them up. Do any of them sense presence/absence of glucose/arabinose? If so, how are they connected (what connections and what type (check the original RegulonDB file to see if they are activation/repression))? Do they make up a feed-forward loop with *araB*?

6. **(1 point)** When TFs sense an environmental input, their reaction is very fast and in a matter of seconds they start activating their target genes in response. As they do so, transcription of target genes is irrevocably initiated, leading to translation and synthesis of the resulting proteins after 10-20 minutes. Protein synthesis has an obvious energetic cost for the cell. As they swim around, *E. coli* cells may encounter sudden, temporary shifts in glucose levels. Explain how a feed-forward loop regulating the *araBAD* operon might prevent *E. coli* from spending energy by synthesizing arabinose degradation genes when arabinose and glucose are around and the cell experiences a short, downward shift in the concentration of glucose. How does that compare to a situation in which both TFs independently regulate *araB*AD?

7. **(1 point)** Look at the overlap between the bi-fan motifs in which the three TFs with the highest out-degree participate. Do the values obtained, together with network metrics inferred in session 2, indicate that the *E. coli* TRN is a shallow network (with few levels and most of the regulatory logic implemented through combinatorial logic gates at promoters) or a deep network (with regulatory logic implemented through multiple cascades of sequential interactions)? Based on what you know about what the *E. coli* transcriptional regulatory network does, why do you think this is so?