Lab_8_BIO247

March 24, 2025

1 "Lab 8: Finding Motifs in Transcription Factor Networks"

1.1 RK Azhigulova

2 Introduction

In today's lab, we'll be taking a Systems Modeling approach to the motifs we've learned out during lecture. In this lab we'l be looking at clever ways to identify loops within a model network, looking at the difference between simple regulation and autoregulation, implementing a feed-forward loop, and learn about repressilators (yes that's a real word).

[26]:
```
'''
Preamble
'''

from igraph import *
from network_loader import *
import matplotlib.pyplot as plt
import random
```

3 Module 1 - Looking for Loops

In this module, you'll be using an iPy notebook to analyze the *E. Coli* transcription factor network. In addition to the coding chunk below, you're also going to need to download 'network_tf_tf_clean.txt' from the class github

3.0.1 Task 1.1 Before running the coding chunk below, look at the data text file, and try to figure out what the data is saying. How is it organized? What information is held? [2 Points]

3.0.2 Response 1.1

Before running the code, I looked through the data file and noticed that it shows interactions between transcription factors and their target genes in E. coli. Each line seems to represent one interaction, starting with the name of the transcription factor, then the target gene, followed by a symbol showing whether it's activating or repressing (like +, -, or +-). There's also a list of sources in brackets that looks like experimental evidence, and at the end, it says whether the interaction is "Strong" or "Weak." The data is separated by tabs and is pretty structured, which makes it easier to work with. Overall, it looks like the file is mapping out how genes regulate each other.

[27]:
```
txt_file = 'network_tf_tf_clean.txt'

network, vertex_names = open_network(txt_file)

# how many nodes & edges
print("Number of nodes: ", len(network.vs))
print("Number of edges: ", len(network.es))
print("Number of self-loops: ", sum(Graph.is_loop(network)))
```

```
Number of nodes:  197
Number of edges:  477
Number of self-loops:  130
```

3.0.3 Task 1.2 Run the code chunk below to generate a randomized Erdos Renyi network of the *E. Coli* transcription network. Compare it to the data-informed network. What differences do you see between the network statistics? What does this mean? [2 points]

3.0.4 Reponse 1.2

When I compared the real E. coli transcription network to the randomized Erdos-Renyi network, I noticed some key differences. Even though both networks have the same number of nodes and edges, the structure is completely different. The real network had more self-loops and likely includes specific patterns like feedback and feedforward loops, which make sense biologically since genes often regulate themselves or work in coordinated ways. The random network, on the other hand, just connects nodes without any logic, so it doesn't have the same kind of structure or meaningful motifs. This shows that the biological network is not random at all — it's organized in a way that supports regulation, control, and function in the cell.

[28]:
```
random.seed(42)
g = Graph.Erdos_Renyi(197,m=477,directed=True, loops=True)
# how many nodes & edges
print("Number of nodes: ", len(g.vs))
print("Number of edges: ", len(g.es))
print("Number of self-loops: ", sum(Graph.is_loop(g)))
```

```
Number of nodes:  197
Number of edges:  477
Number of self-loops:  5
```

4 Module 2 - Visualization

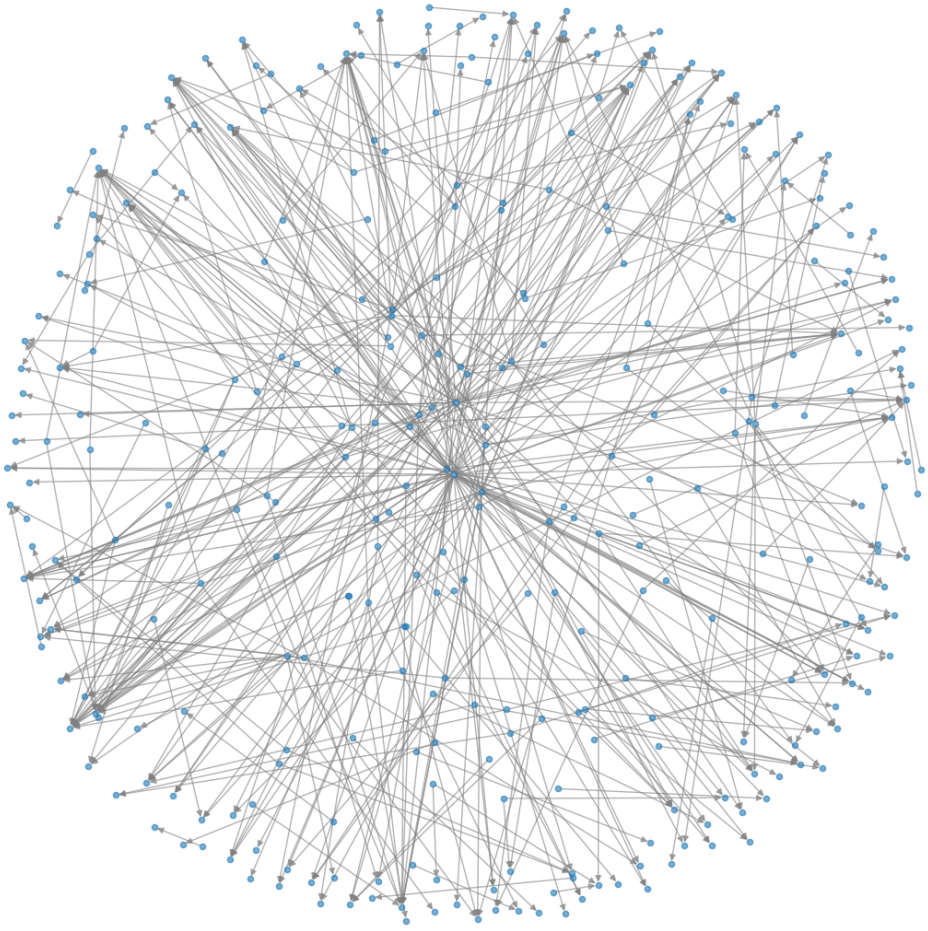
Okay, so we can calculate some stats for a given network. But it would be nice if we had a visualization to go along with that. Look online to try and find a way to graph the networks defined in Module 1. Try at least 2 methods. If they don't work, try to figure out why not.

[29]:
```
'''
Module 2. First method
'''
```

```python
import networkx as nx
import matplotlib.pyplot as plt

# Load the network from the text file
def open_network_nx(filepath):
    G = nx.DiGraph()
    with open(filepath, 'r') as f:
        for line in f:
            parts = line.strip().split('\t')
            if len(parts) >= 2:
                source, target = parts[0], parts[1]
                G.add_edge(source, target)
    return G

# Path to your data file
txt_file = 'network_tf_tf_clean.txt'

# Load the transcription factor network
network = open_network_nx(txt_file)

# Draw the full network
plt.figure(figsize=(12, 12))
pos = nx.spring_layout(network, k=0.3, seed=42)  # Force-directed layout
nx.draw(network, pos,
        node_size=20,
        edge_color='gray',
        alpha=0.6,
        with_labels=False)

plt.title("E. coli Transcription Factor Network", fontsize=14)
plt.tight_layout()
plt.savefig("ecoli_tf_network.png", dpi=300)
plt.show()
```

```
/var/folders/rb/b8cq5_zn7fl43dd098_9688c0000gn/T/ipykernel_6107/669901668.py:34:
UserWarning:

This figure includes Axes that are not compatible with tight_layout, so results
might be incorrect.
```

E. coli Transcription Factor Network



[30]:
```
'''
Module 2. Second method
'''

import networkx as nx
import plotly.graph_objects as go

# Create a small sample graph
G = nx.DiGraph()
G.add_edges_from([("A", "B"), ("B", "C"), ("C", "A"), ("C", "D")])

pos = nx.spring_layout(G)
```

```
edge_x = []
edge_y = []
for src, tgt in G.edges():
    x0, y0 = pos[src]
    x1, y1 = pos[tgt]
    edge_x += [x0, x1, None]
    edge_y += [y0, y1, None]

node_x = [pos[node][0] for node in G.nodes()]
node_y = [pos[node][1] for node in G.nodes()]

edge_trace = go.Scatter(x=edge_x, y=edge_y, line=dict(width=0.5, color='#888'),
                        hoverinfo='none', mode='lines')

node_trace = go.Scatter(x=node_x, y=node_y, mode='markers+text',
                        hoverinfo='text', text=list(G.nodes()),
                        marker=dict(size=10, color='lightblue'))

fig = go.Figure(data=[edge_trace, node_trace],
                layout=go.Layout(title='Interactive Network (Plotly)',
                                 showlegend=False, hovermode='closest'))
fig.show()
```

4.0.1 Interpretation:

First Method: NetworkX + Matplotlib In this method, I used networkx to build the E. coli transcription factor network directly from the data file, and then used matplotlib to draw it. The layout was generated using a force-directed algorithm (spring_layout), which tries to space out the nodes in a visually balanced way. This approach works well for basic visualization, especially if I want to save a static image (like a PNG). However, it's limited in terms of interaction — I can't zoom or move nodes. Also, with a large network like this, the visualization can look messy (funky?) and hard to interpret.

Second Method: NetworkX + Plotly (Interactive) In the second method, I still used networkx to define the network structure, but switched to Plotly for the visualization. Plotly creates a web-based interactive graph where I can play over nodes to see their labels. I manually passed coordinates from the layout into Plotly's Scatter objects to draw edges and nodes. Even though this method takes a little more setup (actually a lot of setup), it looks much better for exploring complex networks and is way more interactive and user-friendly. It's especially useful when I want to focus on subgraphs or share the network online.

5 Module 3 - Identifying Motifs

We want to find the number of C1-FFLs in our network: $x \to y \to z$ **and** $x \to z$

Using the module from model 1, write a piece of code that looks for I1-FFLs, and prints out the totla number in the network

I know, Professor, you do not like when we name function in one letter, thus I renamed the variables x, y, and z to TF1, TF2, and gene respectively.

[ ]:
```
'''
Module 3
'''

import networkx as nx

def count_c1_ffls(G):
    count = 0
    c1_ffls = []

    # For each transcription factor TF1
    for TF1 in G.nodes():
        # For each transcription factor TF2 regulated by TF1
        for TF2 in G.successors(TF1):
            # For each gene regulated by TF2
            for gene in G.successors(TF2):
                # Check if TF1 also directly regulates the same gene
                if gene != TF1 and G.has_edge(TF1, gene):
                    count += 1
                    c1_ffls.append((TF1, TF2, gene))

    print("Total number of C1-FFLs found:", count)
    return c1_ffls

# Load network and count motifs
G = open_network_nx('network_tf_tf_clean.txt')
motifs = count_c1_ffls(G)
```

```
Total number of C1-FFLs found: 0
```