Practical 1 - Network Motifs

Jan Izquierdo Ramos, Jaume Jurado Sanchez

October 2024

SYSTEMS AND NETWORK BIOLOGY- PRACTICAL 1

Network motifs

To submit your report, answer the questions below and save the *notebook* clicking on `File > Download as > iPython Notebook` in the menu at the top of the page. **Rename the notebook file** to ''`practicalN_name1_name2.ipynb`'', where `N` is the number of the practical, and `name1` and `name2` are the first surnames of the two team members (only one name if the report is sent individually). Finally, **submit the resulting file through the *Aula Global***.

Remember to label the axes in all the plots.

IMPORTANT REMINDER: Before the final submission, remember to ***reset the kernel*** *and re-run the whole notebook again to check that it works.*

The objectives of this practical are: - To become familiar with the Python package networkx for graph manipulation. - To identify network motifs in two publicly available transcriptional networks for *E. coli.*

0. Introduction to `NetworkX`

The networkX package has a large number of functions to work with network objects. During this practical session you will need to use some of them that are listed here. **Notice that expressions like `<graph>` or `<node>` are not valid Python, and you are expected to substitute them by a variable that holds a networkx graph object or a string with the name of a node, respectively.**

- `<graph> = nx.DiGraph()` creates a directed graph.
- `<graph>.number_of_nodes()` returns the number of nodes of the `<graph>` object.
- `<graph>.number_of_edges()` returns the number of edges of the `<graph>` object.
- `<graph>.nodes()` returns a list with all the nodes in `<graph>`
- `<graph>.neighbors(<node>)` returns a list with all the neighbors of node `<node>`
- `<graph>.successors(<node>)` returns a list with all nodes receiving connections from `<node>`

- `<graph>.add_edge(<node1>,<node2>)` adds an edge in the `<graph>` graph from node `<node1>` to node `<node2>` .
- `<graph>.remove_node(<node>)` removes the node with name `<node>` from the `<graph>` graph.
- `<graph_copy> = <graph>.copy()` returns a **deep copy** of the `<graph>` graph object (**deep** which means that one can modify `<graph_copy>` without affecting `<graph>`).

1. Importing the network

There is a large number of databases of cellular regulatory networks (involving metabolic reactions, protein-protein interactions, gene regulation, . . .). In this practical session we will work with a transcription factor network, which can usually be treated as a directed simple graph. We will use the database compiled in the lab of Uri Alon from existing literature (Shen-Orr et al, Nature Genetics, 2002), which describes gene regulatory interactions in *Escherichia coli*.

First we will import all necessary packages including the `networkX` package:

[3]:
```
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Next we will download the database, loading it into a graph object. To that end, we have created the following function:

[4]:
```
def load_network(fname):
    G=nx.DiGraph()
    fh=open(fname,'r')
    for line in fh.readlines():
        n1,n2,s=line.split()
        G.add_edge(n1,n2)
    return G
```

The database can be found at Uri Alon's lab website. In that page you will find a **Downloable Materials** drop down menu. Within that menu, select **Donwloadable data**, then **E. coli transcription network** (third by the end) and then download **Version 1.1**, which is a compressed file. Extract the files and explore their content, in particular the `README.txt` file, that explains the contents of the folder.

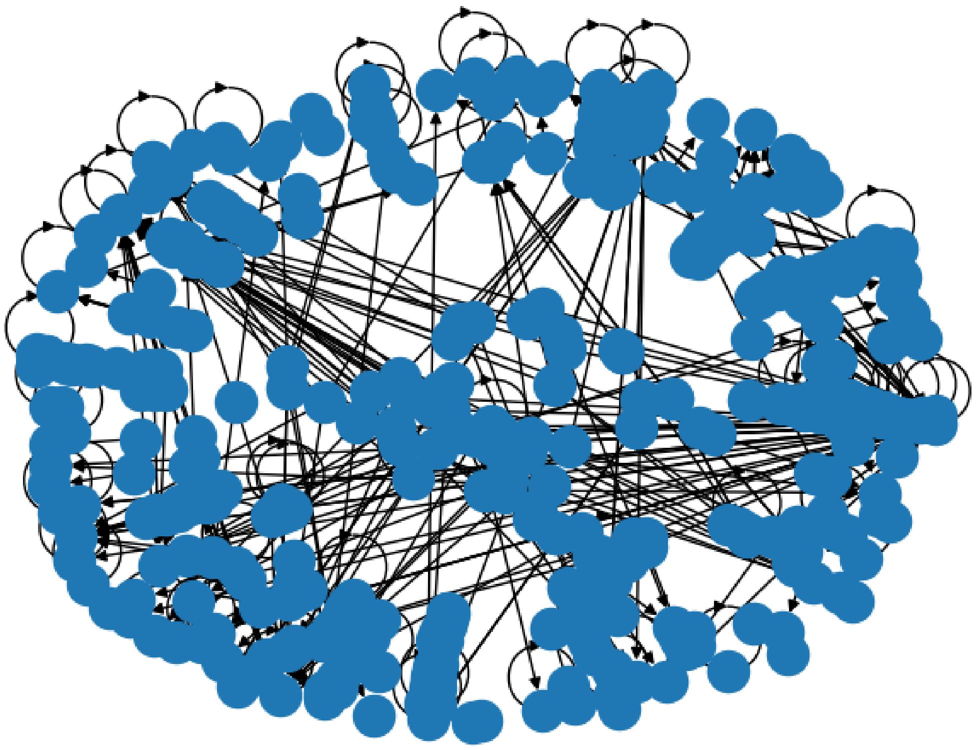
Use the function `load_network` given above to load the file `coliInterFullVec.txt`.

[5]:
```
nw=load_network("coliInterFullVec.txt")
```

2. Visualizing and analyzing the network

Plot the network using networkx draw function.

[6]:
```
nx.draw(nw)
```



Calculate (1) how many operons (nodes) are included in the network, (2) how many regulations (edges) are described, (3) the average connectivity $\lambda = E/N$ of the network, and (4) the connection probability $p = E/N^2$.

[7]:
```
nodes=nw.number_of_nodes()
edges=nw.number_of_edges()
conn=edges/nodes
p=edges/(nodes**2)

print(f'Operons: {nodes} \nRegulations: {edges} \nAverage connectivity: {conn}␣
↪\nConnection probability: {p}')
```

```
Operons: 423
Regulations: 578
Average connectivity: 1.3664302600472813
Connection probability: 0.0032303315840361262
```

3. Direct feedback loops

Next, calculate the number of direct feedback loops (self-edges, to be called simply "feedback loops" in what follows) in this network.

[8]:
```
fdbk=len(list(nx.selfloop_edges(nw)))
fdbk
```

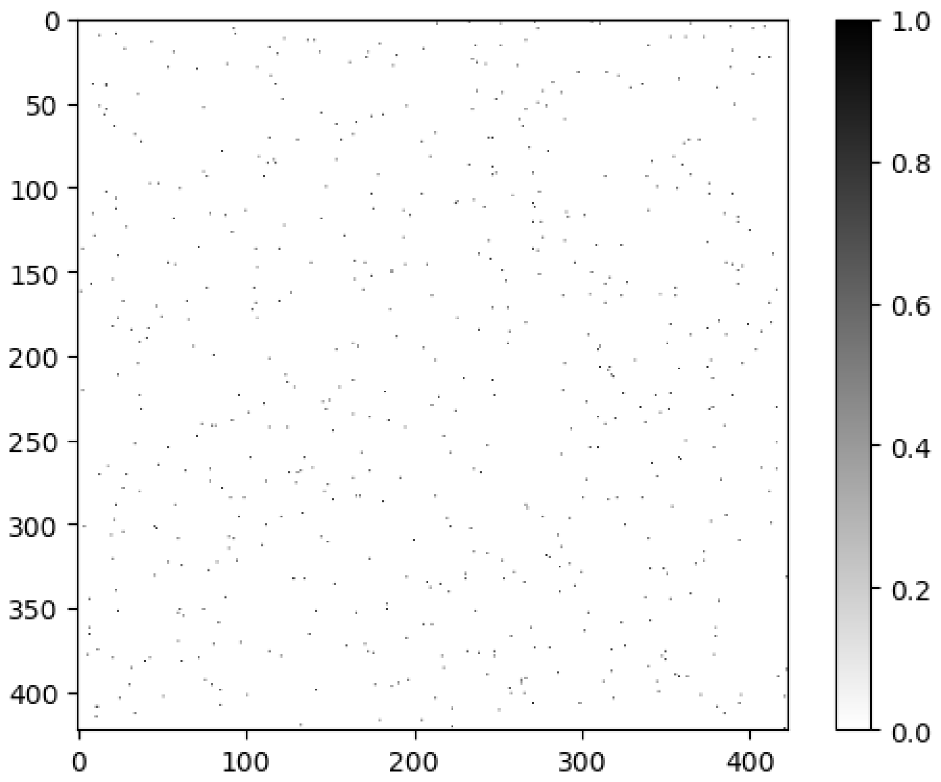
[8]:
```
59
```

In order to determine whether the feedback loop is a network motif, we need to compare this number with the number of feedback loops we would expect by chance. To that end, we will now create a random network with the same number N of nodes and connection probability p as the real gene regulatory network. The simplest way of doing this is by creating an $N \times N$ matrix whose elements are uniform random numbers distributed between 0 and 1 (using the function `rand` from the package `numpy.random`), and comparing each element of this matrix with the connection probability p. The result of this comparison for all matrix elements will correspond to the adjacency matrix of the random network. Next, use the command `<graph> = nx.from_numpy_matrix(A , create_using=nx.DiGraph())` to generate the corresponding graph. Calculate the number of edges in the graph to check if the process has been done correctly. Draw the matrix.

[9]:
```
M=[]
for i in range(nodes):
    m=[]
    for j in range(nodes):
        r=np.random.rand()
        if p>r:
            m.append(1)
        else:
            m.append(0)
    M.append(m)

M=np.array(M)
randNW=nx.from_numpy_array(M, create_using=nx.DiGraph())
```

[16]:
```
print("Number of edges:",randNW.number_of_edges())
#nx.draw(randNW)

plt.imshow(M,cmap='binary')
plt.colorbar()
plt.show()
```

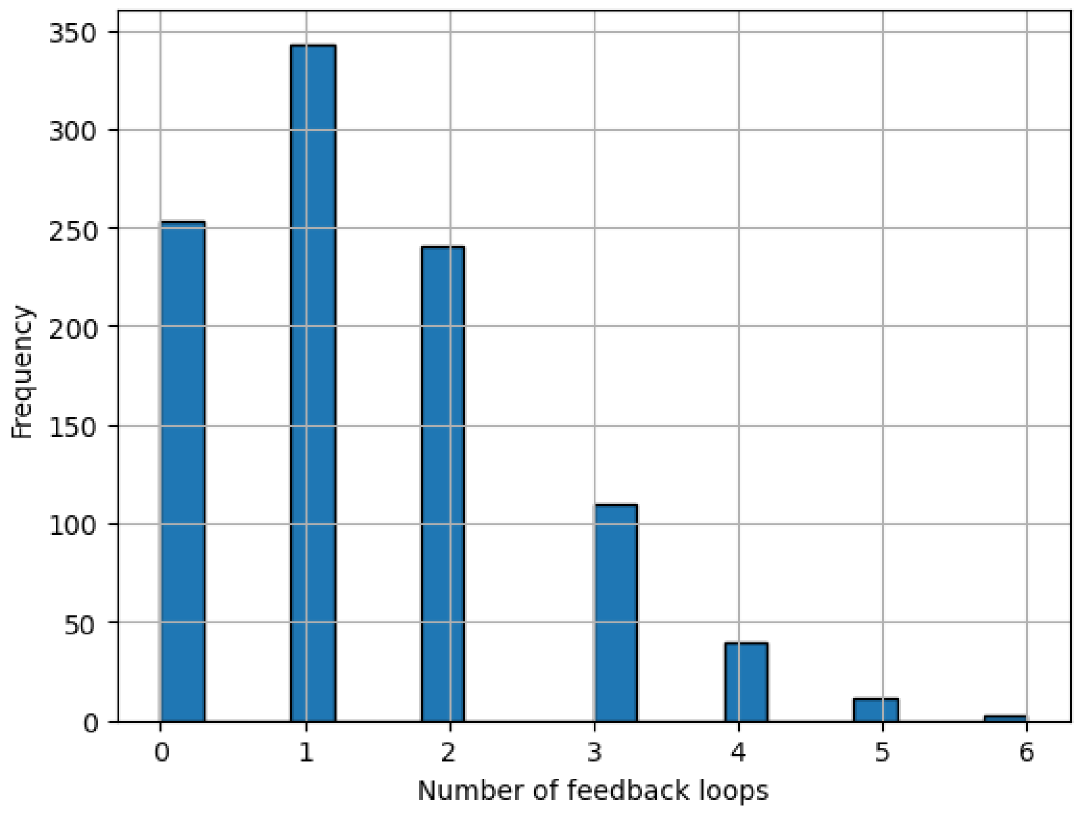
```
Number of edges: 593
```



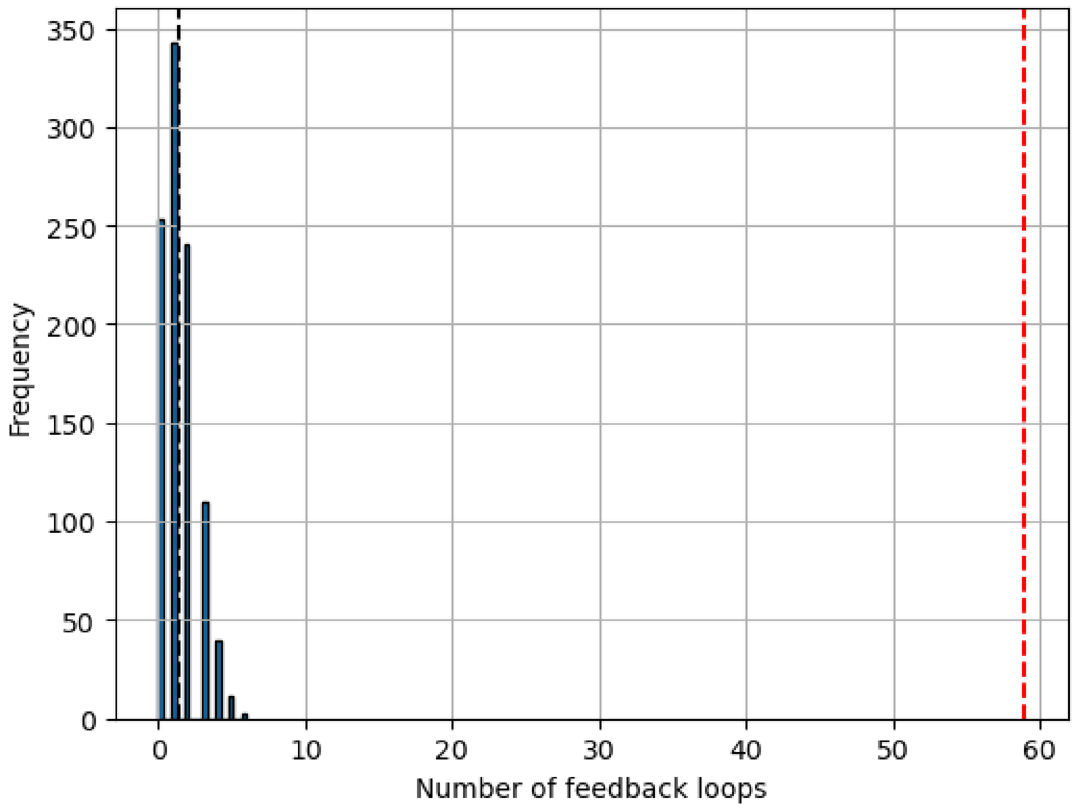
Next, generate a collection of 1000 random matrices and calculate for each instance the number of feedback loops. Plot the distribution (histogram) of this quantity and determine its average value. Compare this result with the number of copies of feedback loops found in the real gene regulatory network, to establish if that module is a network motif.

[61]:
```
def matrix_gen(n):
    M=[]
    for i in range(nodes):
        m=[]
        for j in range(nodes):
            r=np.random.rand()
            if p>r:
                m.append(1)
            else:
                m.append(0)
        M.append(m)

    M=np.array(M)
    return M

def loop_plotter(list_to_plot, real_nw_num, loop_type):
    list_avg=np.mean(list_to_plot)
    plt.hist(list_to_plot, bins=20, edgecolor='black')
    plt.grid(True)
    plt.xlabel(f'Number of {loop_type} loops')
    plt.ylabel('Frequency')
    plt.show()

    print(f"The average number of {loop_type} loops of the 1000 matrices is ",␣
↪list_avg)
    print(f"The number of loops {loop_type} found in the real gene network are␣
↪", real_nw_num)


    plt.hist(list_to_plot, bins=20, edgecolor='black')
    plt.grid(True)
    plt.xlabel(f'Number of {loop_type} loops')
    plt.ylabel('Frequency')
    plt.axvline(list_avg, color='black', linestyle='dashed', linewidth=1.5)
    plt.axvline(real_nw_num, color='red', linestyle='dashed', linewidth=1.5)
    plt.show()
```

[62]:
```
feedback_list=[]
for _ in range(1000):
    M=matrix_gen(nodes)
    mynw=nx.from_numpy_array(M, create_using=nx.DiGraph())
    feedback_list.append(len(list(nx.selfloop_edges(mynw))))

loop_plotter(feedback_list, fdbk, "feedback")
```



```
The average number of feedback loops of the 1000 matrices is  1.386
The number of loops feedback found in the real gene network are  59
```



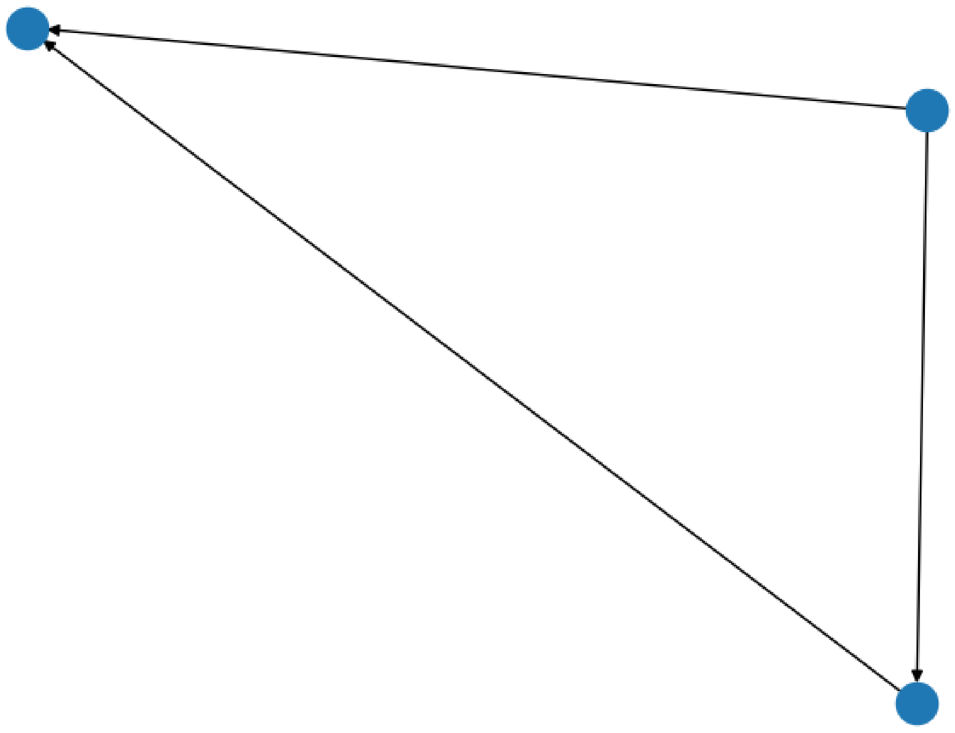
4. Feedforward loops

Finally, we will compute the number of feedforward loops in both the real gene regulatory network and in its randomized versions. To identify the feedforward loops in a graph we will use the following function:

[63]:
```
import networkx.algorithms.isomorphism as iso

def find_pattern(graph, pattern):
    """Find all the subgraphs isomorphic to the given pattern.

    Return an iterator over all the subgraphs in `graph` that are isomorphic to
    the specified `pattern`. It works for both directed and undirected graphs,
    but the type of `graph` and `pattern` arguments must be coherent.


    Based on the code from
    https://zulko.wordpress.com/2012/10/13/
  ↪finding-a-subnetwork-with-a-given-topology-in-e-coli/
    """
    edge_match = None
    if graph.is_directed() and pattern.is_directed():
        matcher = iso.DiGraphMatcher(graph, pattern, edge_match=edge_match)
    elif not graph.is_directed() and not pattern.is_directed():
        matcher = iso.GraphMatcher(graph, pattern, edge_match=edge_match)
    else:
        raise TypeError("type of `graph` and `pattern` arguments is not "
                        "coherent!")

    return matcher.subgraph_isomorphisms_iter()
```

First, we need to design a pattern in the form of a feedforward loop. Create a directed graph and add three edges to it using the corresponding functions given in Section 0 above. Draw the circuit to check that it is a feedforward loop.

[64]:
```
g_4=nx.DiGraph()
g_4.add_edge("A", "B")
g_4.add_edge("B", "C")
g_4.add_edge("A", "C")

nx.draw(g_4)
```



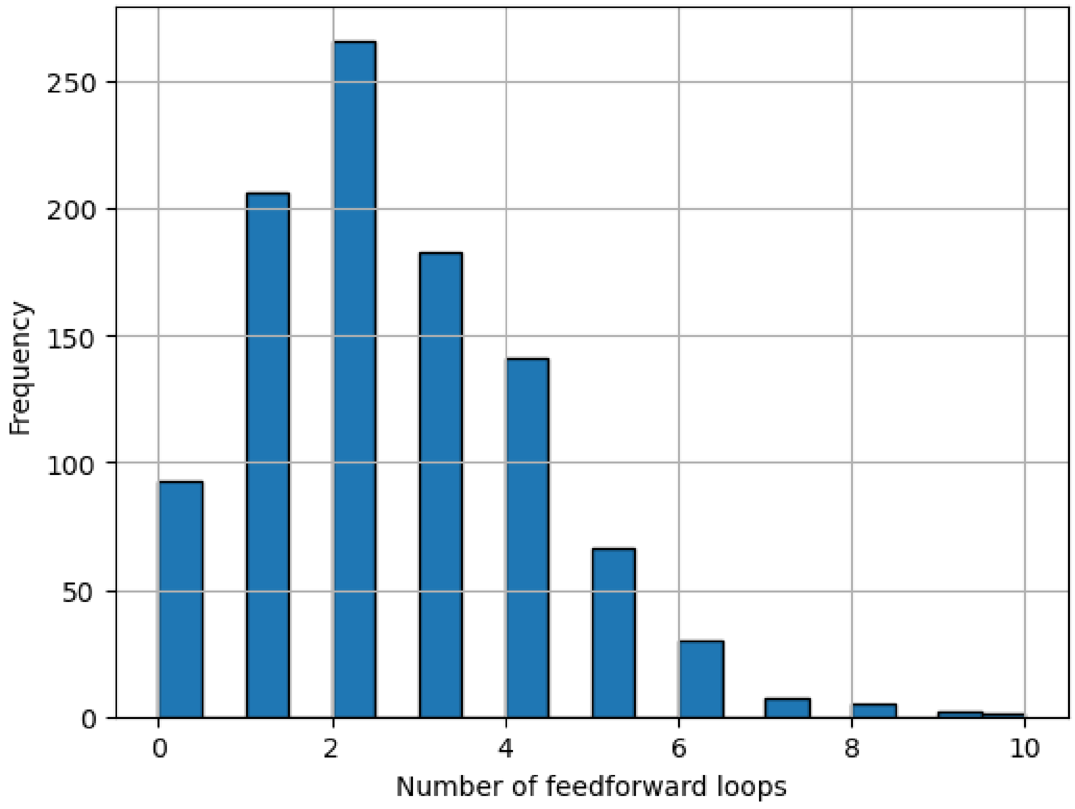
Next, use the function `find_pattern` to find all instances of feedforward circuits in the real gene regulatory network. To do that we will now load the version of the network with no self-loops, `coliInterNoAutoRegVec.txt`. How many feedforward loops are found in the network?

[65]:
```python
nw_clean=load_network("coliInterNoAutoRegVec.txt")
fdfw=len(list(find_pattern(nw_clean, g_4)))
print("The number of feedforward loops in the real network is:", fdfw)
```

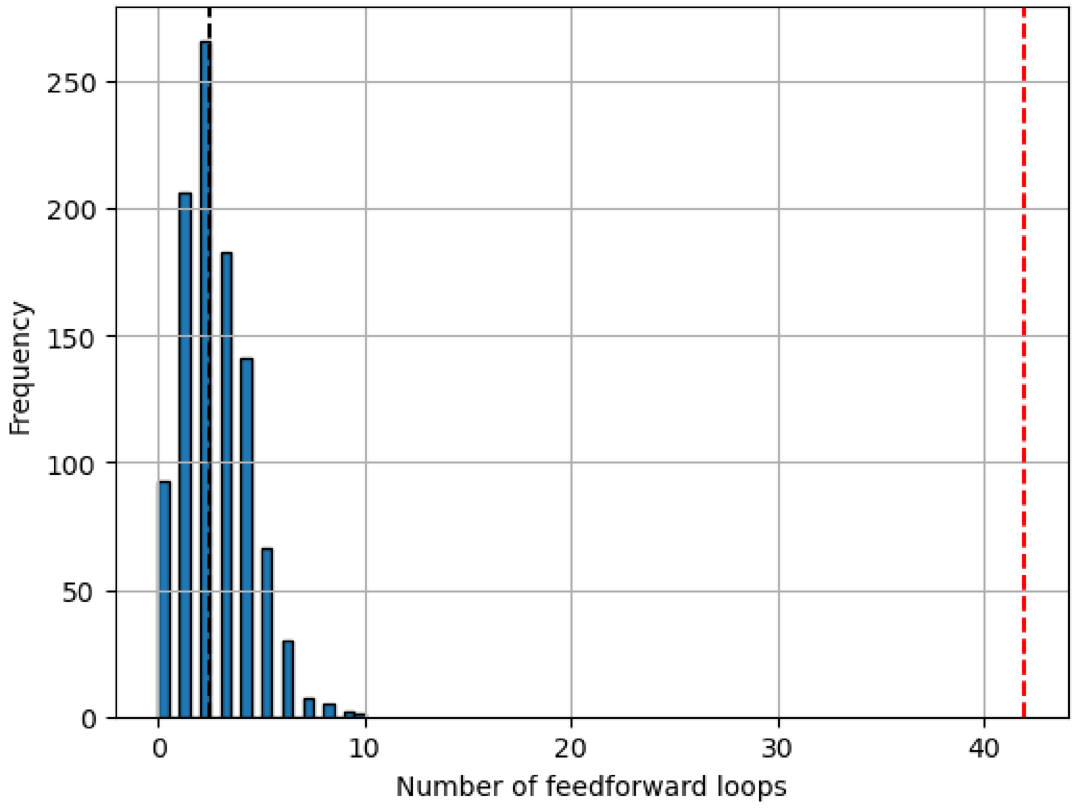
```
The number of feedforward loops in the real network is: 42
```

Finally, generate a collection of 1000 random matrices and calculate for each instance the number of feedforward loops. Plot the distribution (histogram) of this quantity and determine its average value. Compare this result with the number of copies of feedforward loops found in the real gene regulatory network, to establish if that module is a network motif.

[66]:
```python
feedforward_list=[]
for _ in range(1000):
    M=matrix_gen(nodes)
    mynw=nx.from_numpy_array(M, create_using=nx.DiGraph())
    feedforward_list.append(len(list(find_pattern(mynw, g_4))))

loop_plotter(feedforward_list, fdfw, "feedforward")
```



```
The average number of feedforward loops of the 1000 matrices is  2.478
The number of loops feedforward found in the real gene network are  42
```


Frequency
Number of feedforward loops