reporte

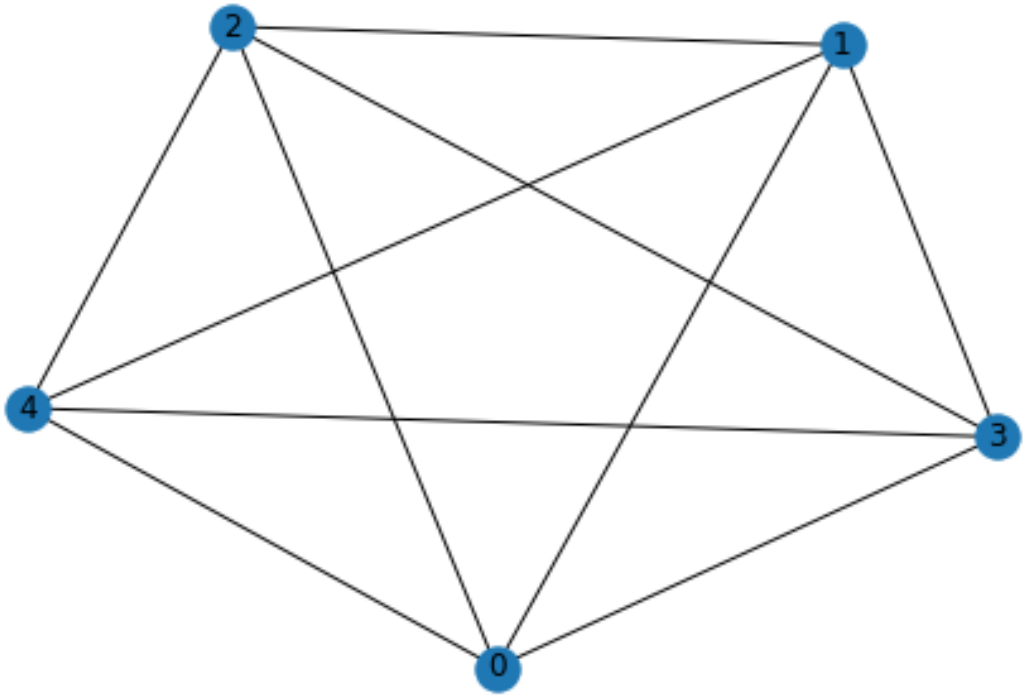
March 3, 2020

1 Actividad 6: Cliques y grafos planos

Alberto Benavides

En esta ocasión se determinará cuándo un grafo no es plano a través de cliques y el teorema de Kuratowski para el caso de K_5. Con el uso de la librería NetworkX y las $\binom{5}{2}$ combinaciones de cinco nodos generadas con la librería itertools se puede diseña un grafo K_5 al añadir aristas entre cada par de nodos de dicho grafo.

[75]:
```
import networkx as nx
import matplotlib.pyplot as plt
from random import randint, random
from itertools import combinations, chain

# https://www.youtube.com/watch?v=MLGm6MUW_YI
K = nx.Graph()
K.add_nodes_from(range(5))
K.add_edges_from(combinations(K.nodes, 2))

plt.figure()
nx.draw(K, with_labels=True)
```

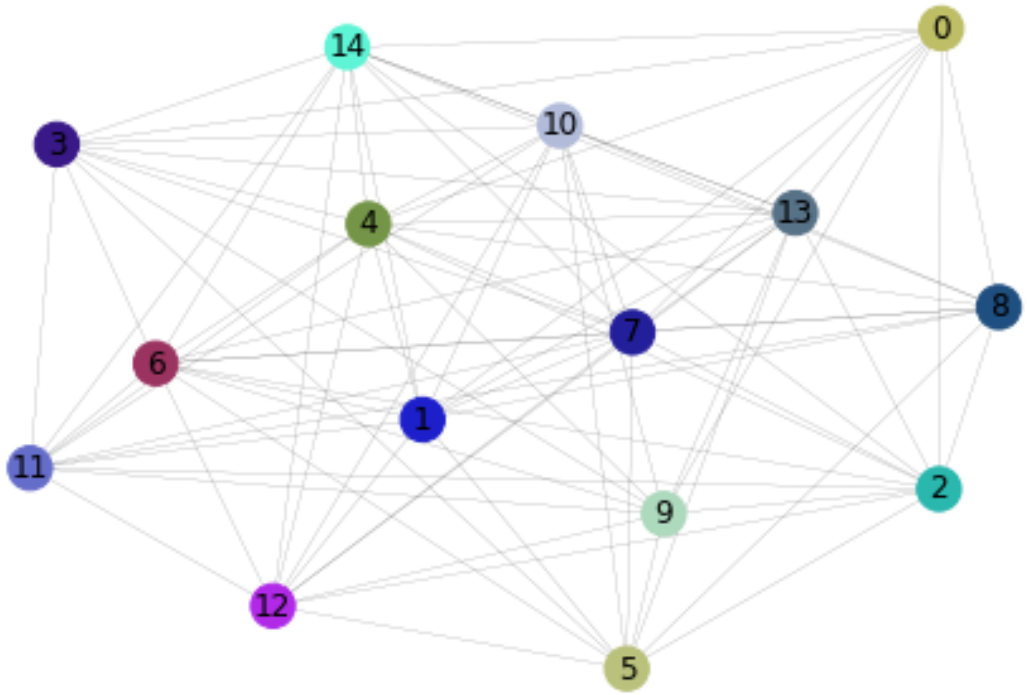


Una de las condiciones para que un grafo no sea plano es que no contenga un subgrafo isomorfo de este grafo K_5. Una manera de determinar esta condición es a través de cliques, pues el grafo K_5 contiene un clique máximo formado por todos sus nodos. De esta forma, si un un grafo se encuentra un clique de 5 nodos, dicho grafo no sería plano.

Ahora bien, se desarrollará un algoritmo para determinar los cliques y cliques máximos de un grafo. Inicialmente, se generará un grafo de n nodos que tienen probabilidad p de tener aristas con cualquier otro nodo.

[76]:
```
p = 0.8
n = 15

G = nx.Graph()
G.add_nodes_from(range(n))
colors = [(random(), random(), random()) for _i in range(n)]

for i in range(n):
    for j in range (i, n):
        if i != j and random() < p:
            G.add_edge(i, j)

options = {
    'node_size': 300,
    'width': 0.1
```

```
}

plt.figure()
# https://stackoverflow.com/a/8083655
nx.draw(G, with_labels=True, node_color=colors, **options)
```



Dado que un clique C es cualquier grafo cuyos nodos pertenezcan a un grafo G si y sólo si cada par de nodos de C tiene una arista entre sí que pertenece también a G, se puede determinar la cantidad de cliques en un grafo al explorar los subconjuntos de sus nodos, por lo que encontrar los cliques de un grafo se considera un problema de complejidad $\mathcal{O}(2^n)$. Para encontrar todos los subconjuntos de un grafo G dado, se usa la función de `powerset` definida en la documentación de itertools, la cual devuelve una concatenación de todas las combinaciones de un conjunto de valores (en esta caso, todos los nodos de G).

[77]:
```
def powerset(iterable):
    "powerset([1,2,3]) --> () (1,) (2,) (3,) (1,2) (1,3) (2,3) (1,2,3)"
    s = list(iterable)
    return chain.from_iterable(combinations(s, r) for r in range(len(s)+1))
combinaciones = list(powerset(G.nodes))
```

A partir de los subconjuntos de nodos en el grafo G, se pueden definir los cliques siguiendo la definición ya dada. Como hay que recorrer dos veces todos los nodos de todos los subconjuntos del grafo G, la complejidad computacional de encontrar los cliques de un nodo es $\mathcal{O}(2^n n^2)$.

Por otro lado, los cliques máximos son aquellos que no son subconjuntos de otros cliques del grafo G. Encontrar los cliques máximos implica haber encontrado todos los cliques y comparar cada clique con los demás para determinar cuáles no son subconjutos de los demás cliques, lo que sería un algoritmo de complejidad $\mathcal{O}(n^2)$, pero dado que el número de cliques máximos $m <= n$, entonces se puede compararse cada clique con el número de cliques máximos para tener una complejidad de $\mathcal{O}(nm)$. Con todo, esta complejidad puede reducirse más si se ordenan los subconjuntos de G de mayor a menor cardinalidad con la finalidad de que sólo se comparen conjuntos con subconjuntos cuya cardinalidad sea igual o mayor a la de ellos mismos, lo que implica una complejidad de $\mathcal{O}(\frac{nm}{2})$. Por este motivo se ordenan las combinaciones de mayor a menor cardinalidad.

[78]:
```
# Reverso de la lista
combinaciones = combinaciones[::-1]
```

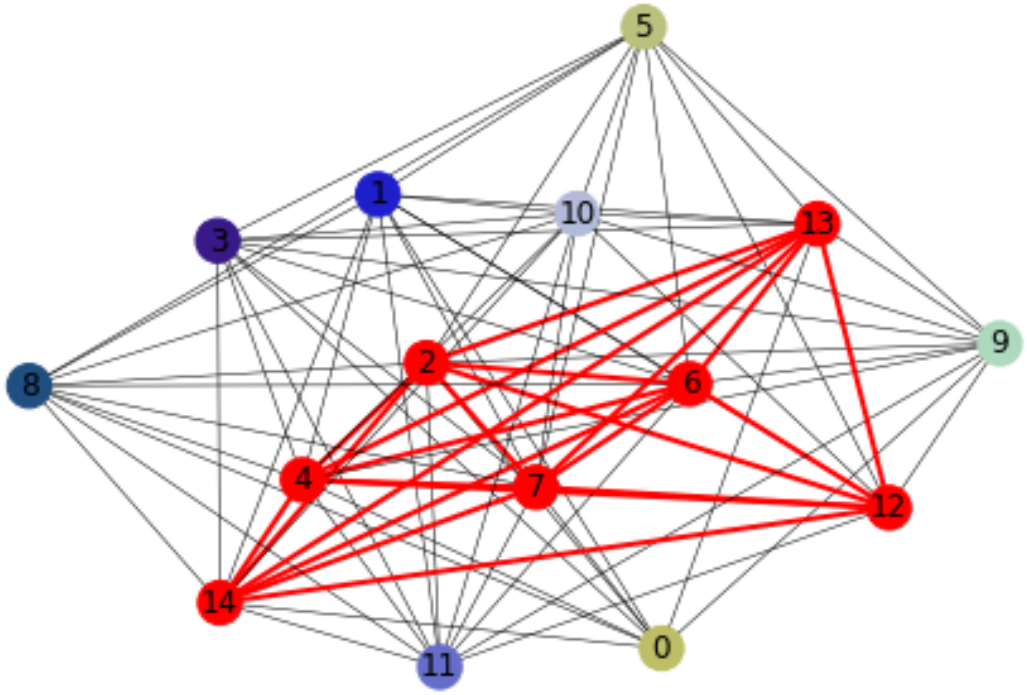
Posteriormente, se procede a determinar los cliques y cliques máximos y, en cada combinación, la determinación de que se trate de un clique máximo. De modo que el siguiente algoritmo tiene una complejidad $\mathcal{O}(2^{n-1} \cdot (n^2 + m))$.

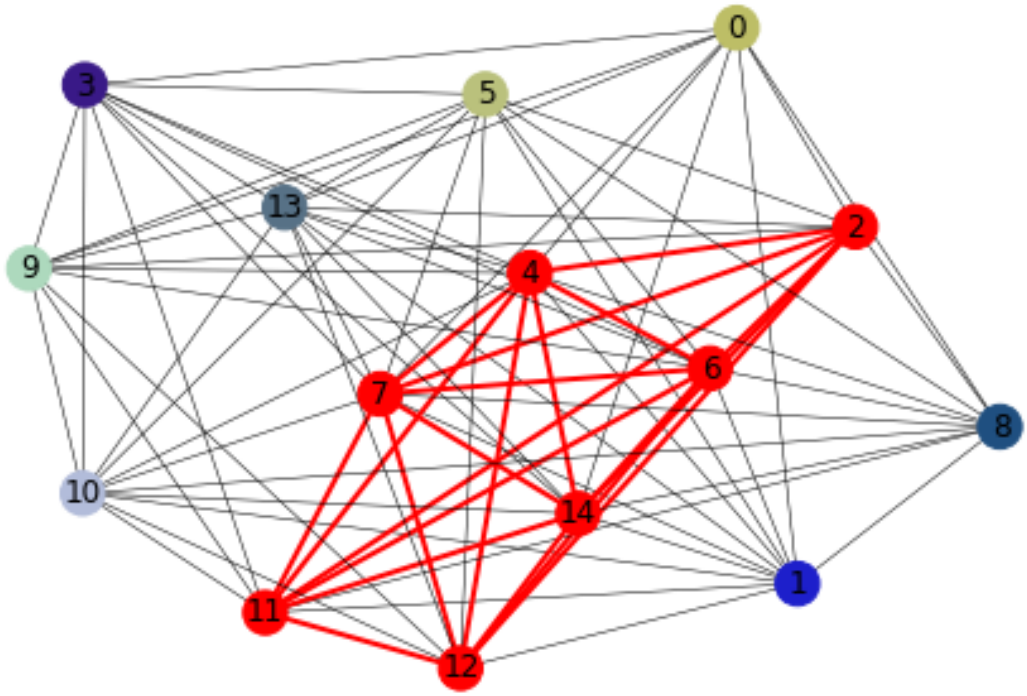
[79]:
```
cliques = []
max_cliques = []
# Por cada combinación
for i in combinaciones:
    es_clique = True
    C = nx.Graph()
    for j in i:
        for k in i:
            if j != k and not G.has_edge(j, k):
                es_clique = False
            else:
                C.add_nodes_from((j, k))
                C.add_edge(j, k)
    if es_clique and len(C.nodes) > 0:
        cliques.append(C)
        if len(cliques) == 1:
            max_cliques.append(C)
        else:
            es_max_clique = True
            for x in max_cliques:
                if C != x:
                    t = set(C.nodes)
                    t1 = set(x.nodes)
                    if t.issubset(t1):
                        es_max_clique = False
            if es_max_clique:
                max_cliques.append(C)
```

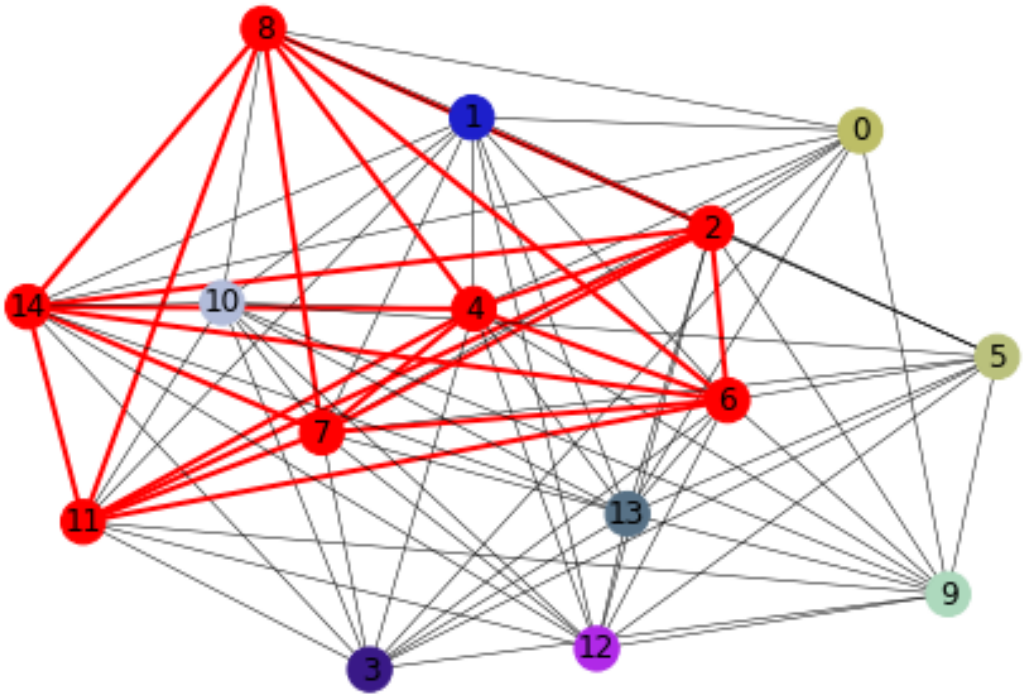
Finalmente, se pueden graficar los cliques máximos (o todos los cliques si se itera en `cliques` en lugar de `max_cliques`).

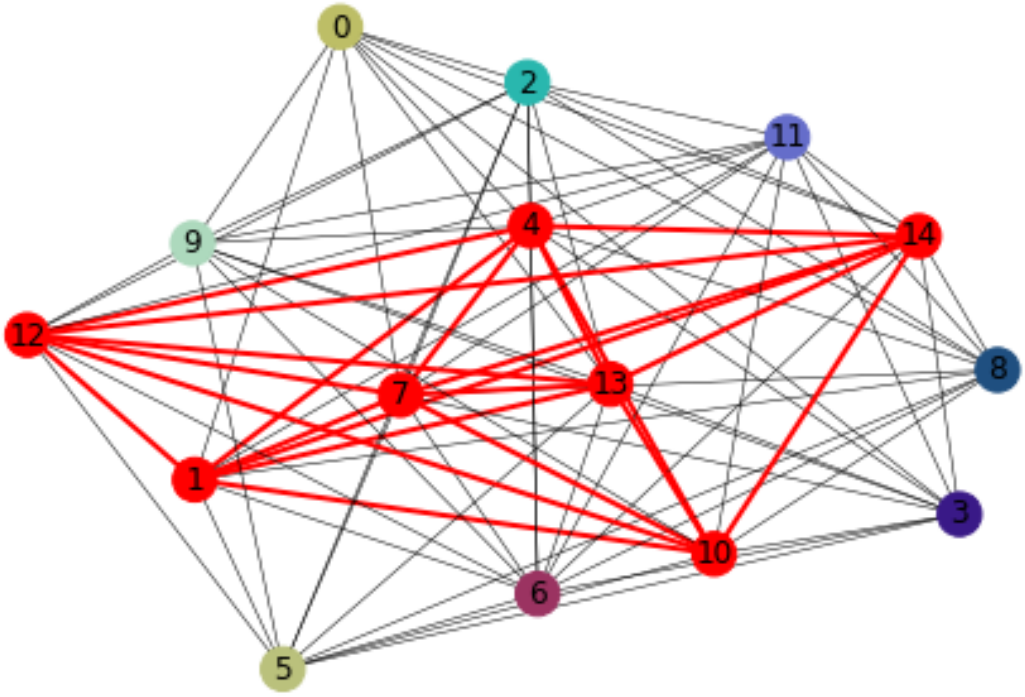
[80]:
```
for C in max_cliques:
    C_colors = []
    for i in G.nodes:
        if i in C.nodes:
            C_colors.append('red')
        else:
            C_colors.append(colors[i])
    edges = G.edges()
    edge_colors = []
    weights = []
    for i, j in G.edges:
        if C.has_edge(i, j):
            edge_colors.append('red')
            weights.append(2)
        else:
            edge_colors.append('black')
            weights.append(0.5)
    plt.figure()
    # https://stackoverflow.com/a/8083655
    nx.draw(G, with_labels=True, node_color=C_colors, edge_color=edge_colors,␣
 ↪node_size=300, width=weights)
```

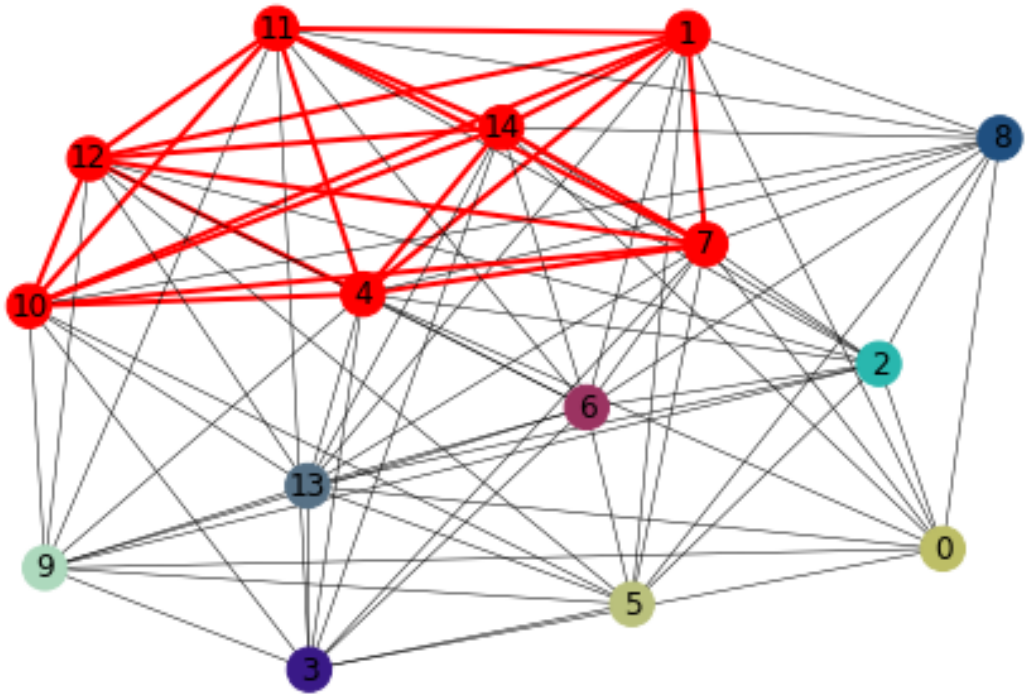
```
<ipython-input-80-0f40f686d80f>:18: RuntimeWarning: More than 20 figures have
been opened. Figures created through the pyplot interface
(`matplotlib.pyplot.figure`) are retained until explicitly closed and may
consume too much memory. (To control this warning, see the rcParam
`figure.max_open_warning`).
  plt.figure()
```

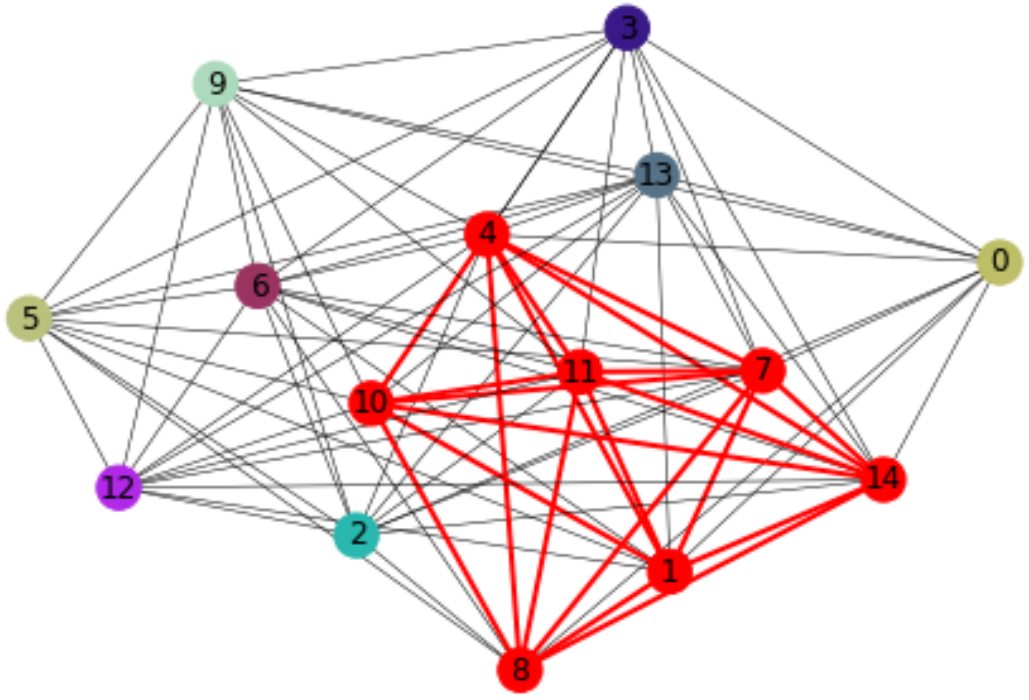

5
1
10
13
3
9
2
6
8
4
7
12
14
0
11

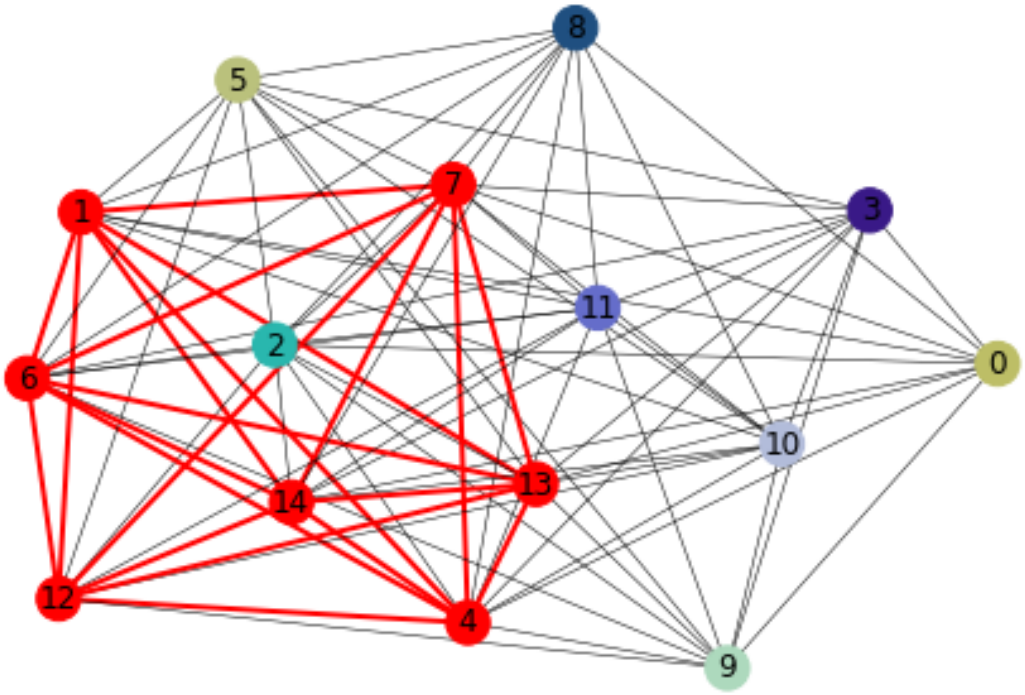

0
3
5
13
2
9
4
6
7
8
10
14
1
11
12

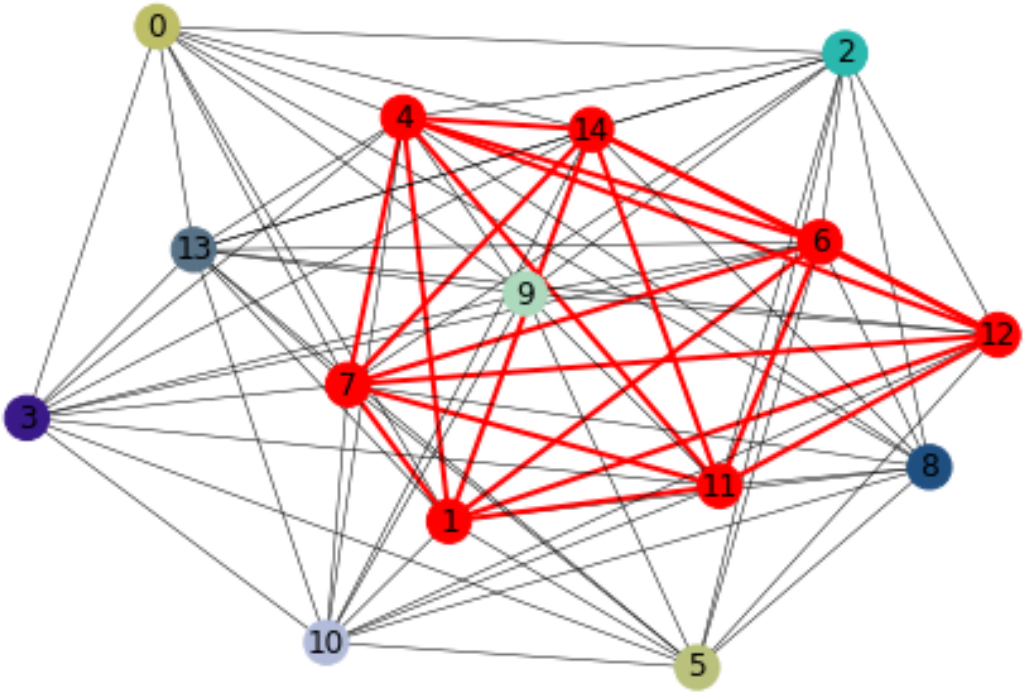

8
1
0
2
14
10
4
5
6
7
11
13
9
12
3

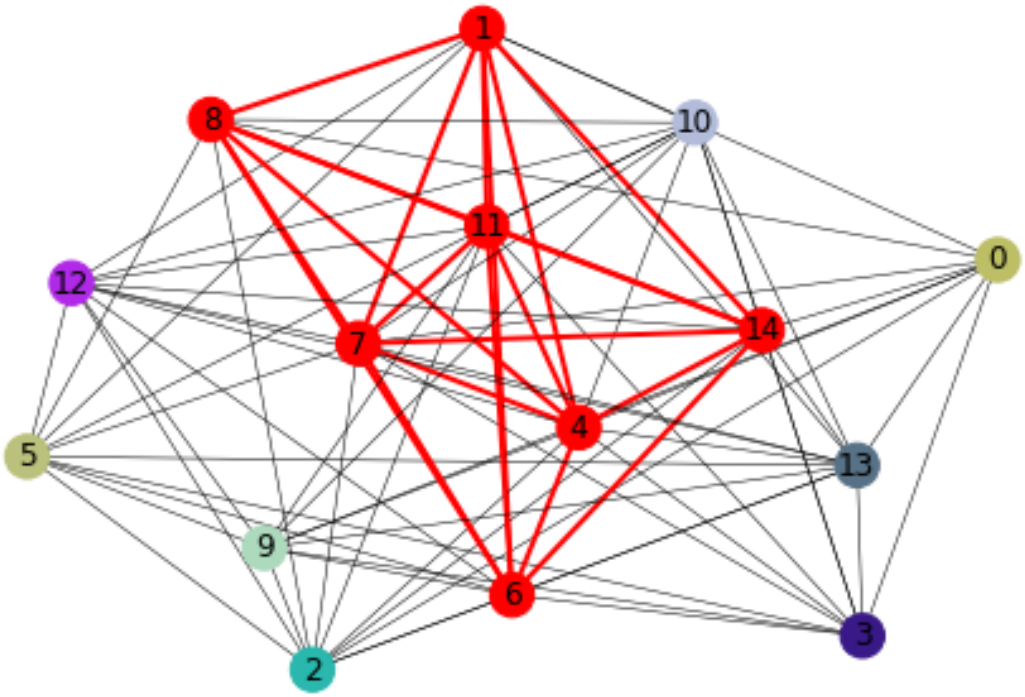

0
2
11
9
4
14
12
8
7
13
1
3
10
6
5

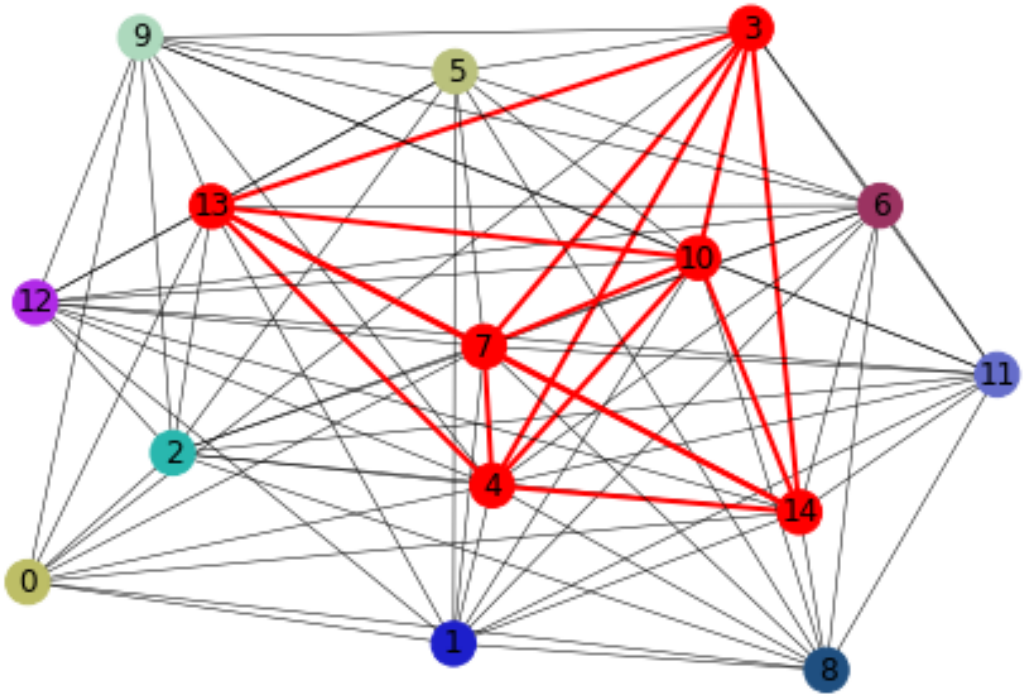

11
1
14
8
12
7
10
4
2
6
13
0
9
5
3

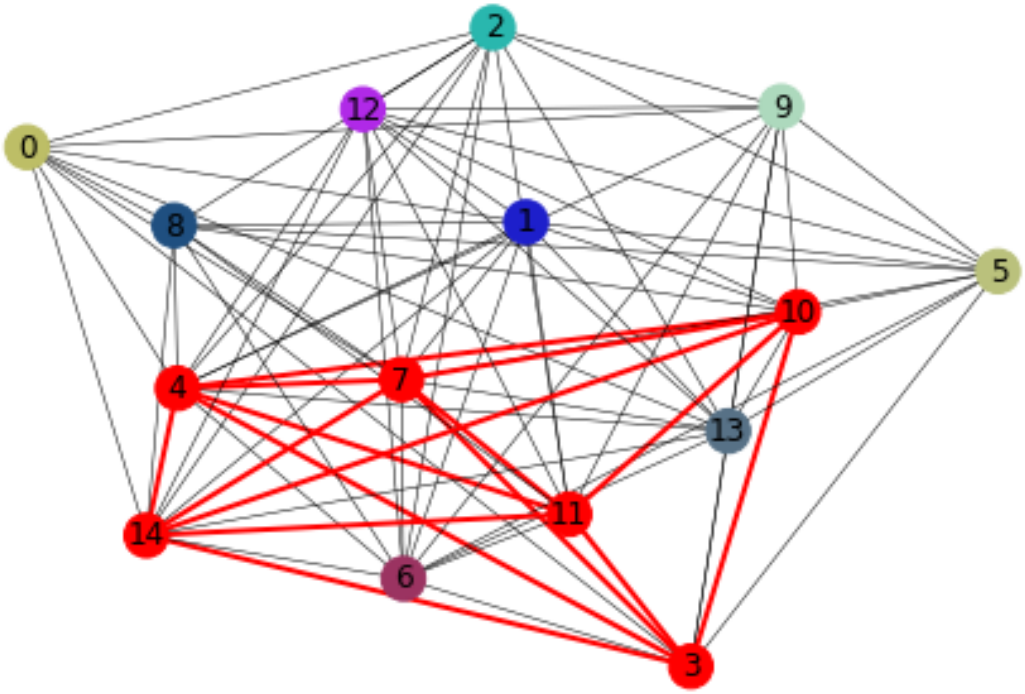

3
9
13
4
0
6
5
11
7
10
14
12
2
1
8

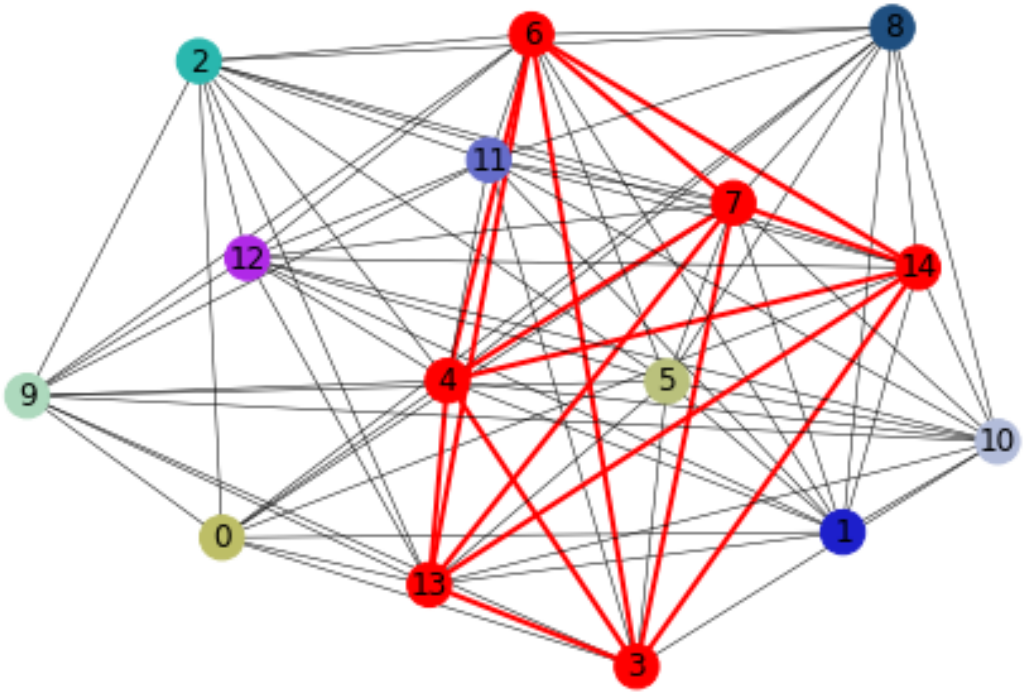

8
5
7
1
3
11
2
0
6
10
13
14
12
4
9

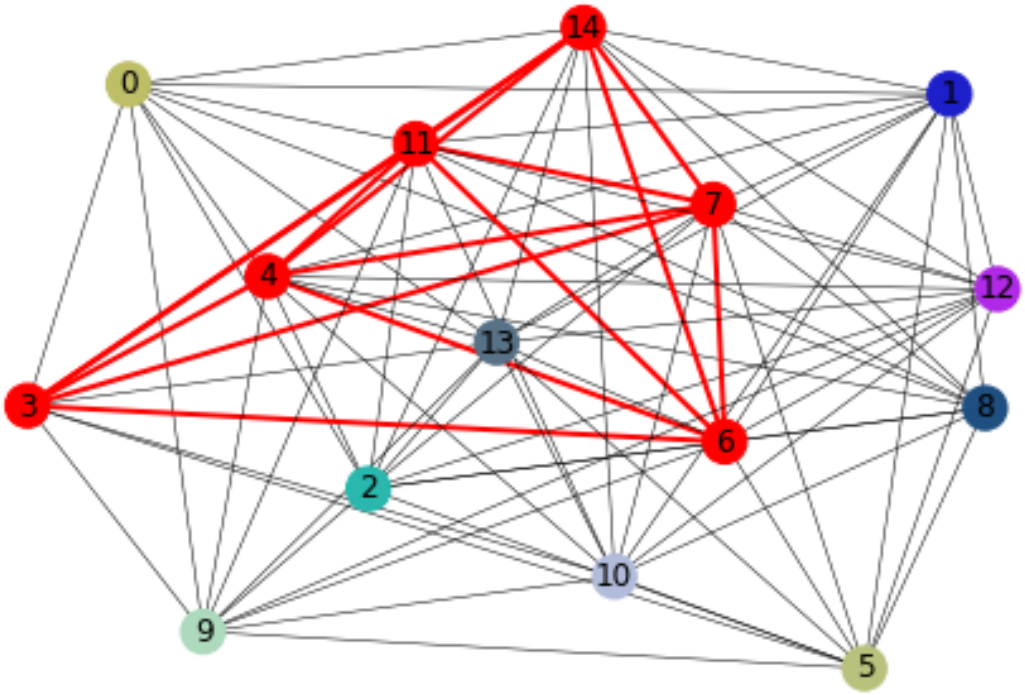

0
2
4
14
13
6
9
12
7
3
8
11
1
10
5

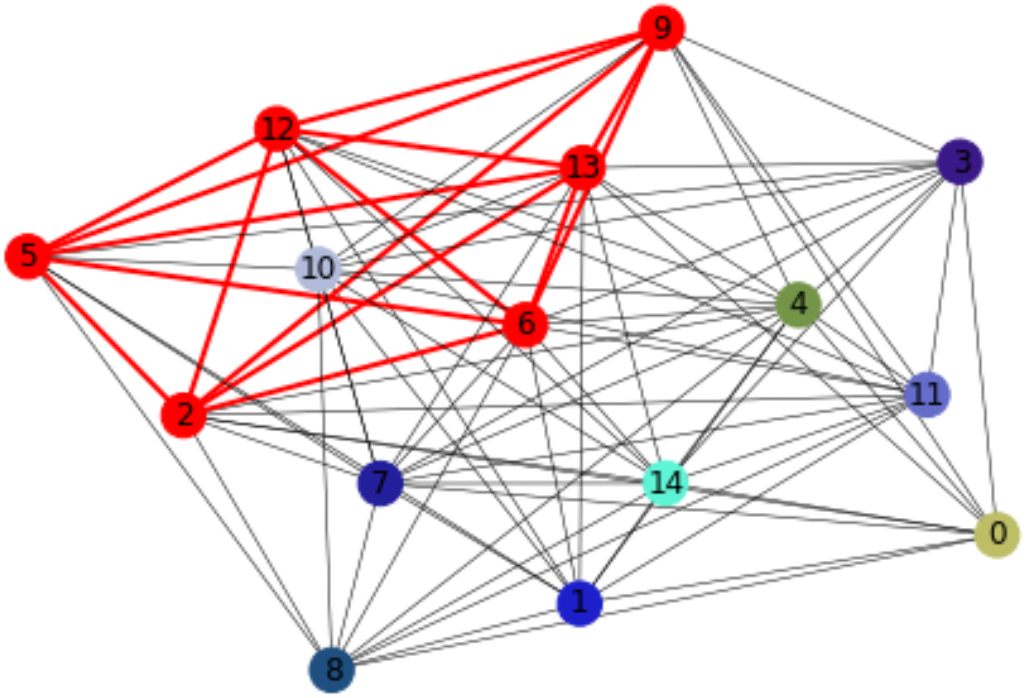

1
8
10
11
0
12
7
14
4
5
13
9
6
3
2

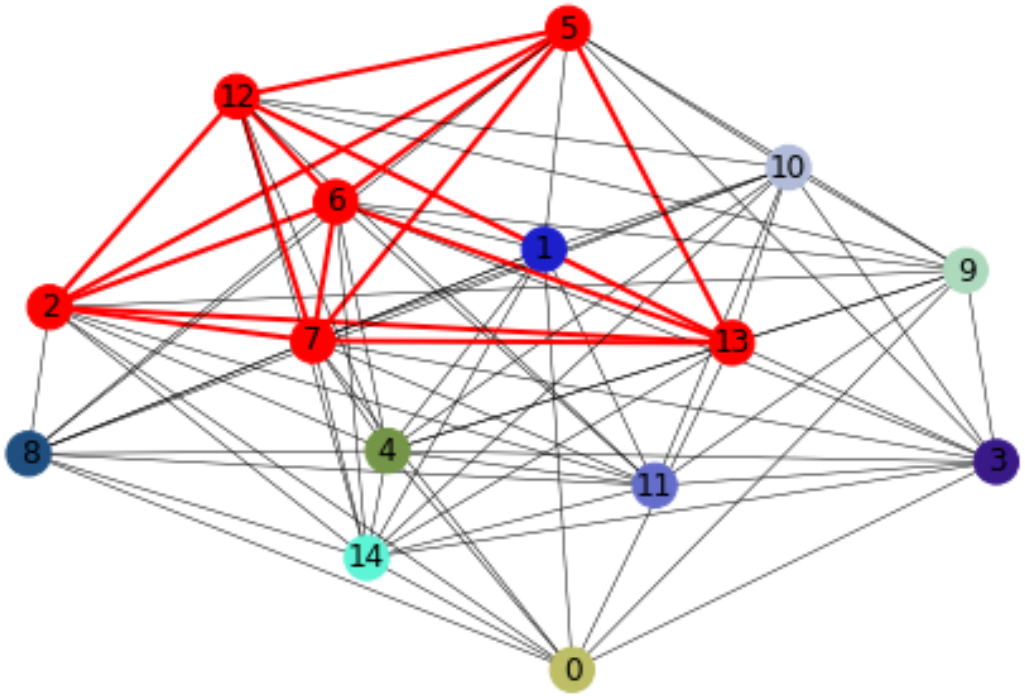

9
3
5
13
6
10
12
7
11
2
4
14
0
1
8

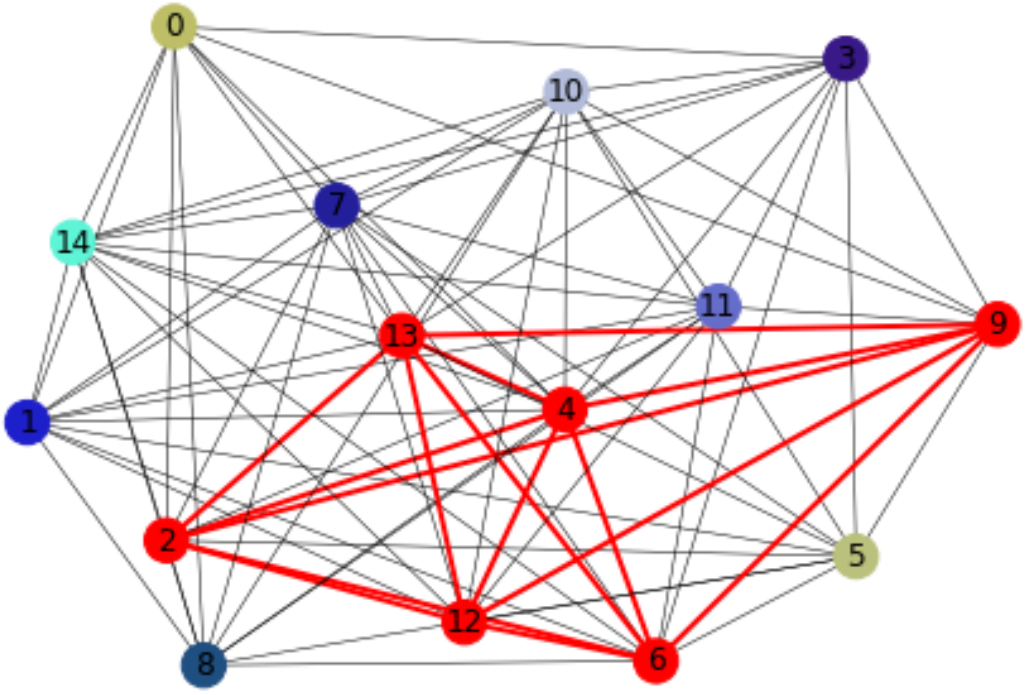

2
12
9
0
8
1
5
10
4
7
13
11
14
6
3

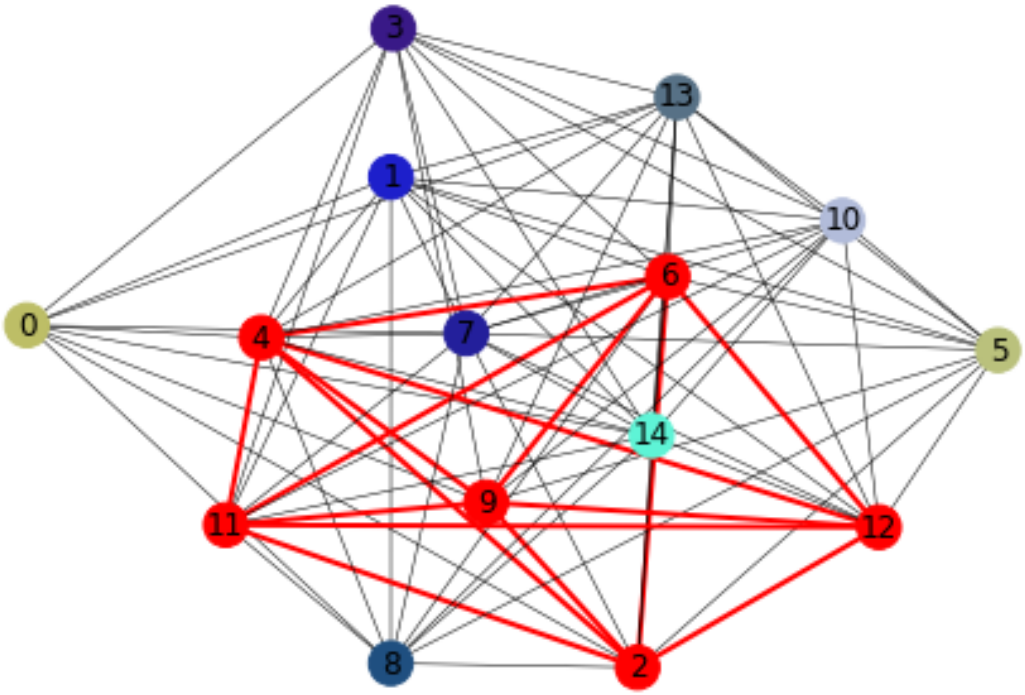

6
8
2
11
7
12
14
9
4
5
10
0
1
13
3

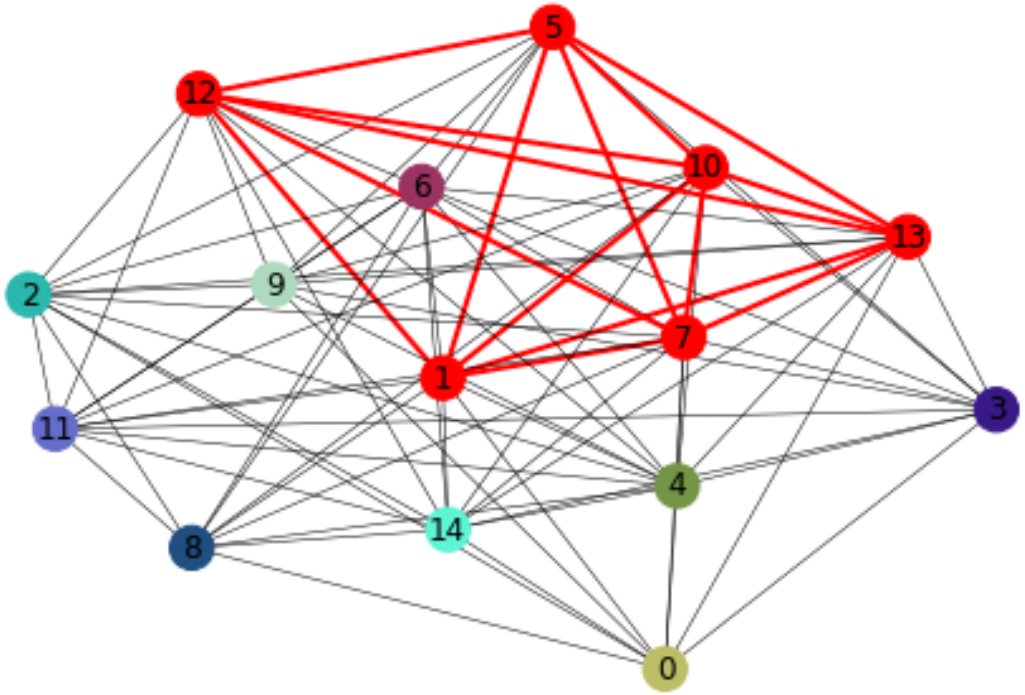

14
0
1
11
7
4
12
13
3
8
6
2
10
9
5

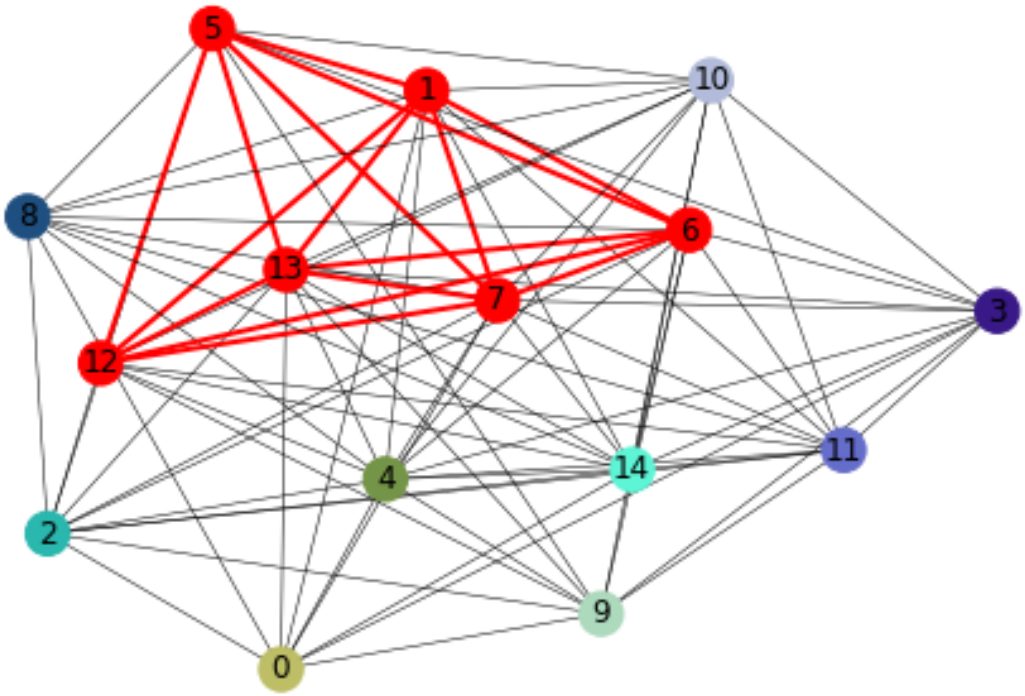

9
12
3
13
5
10
4
6
11
2
7
14
0
1
8

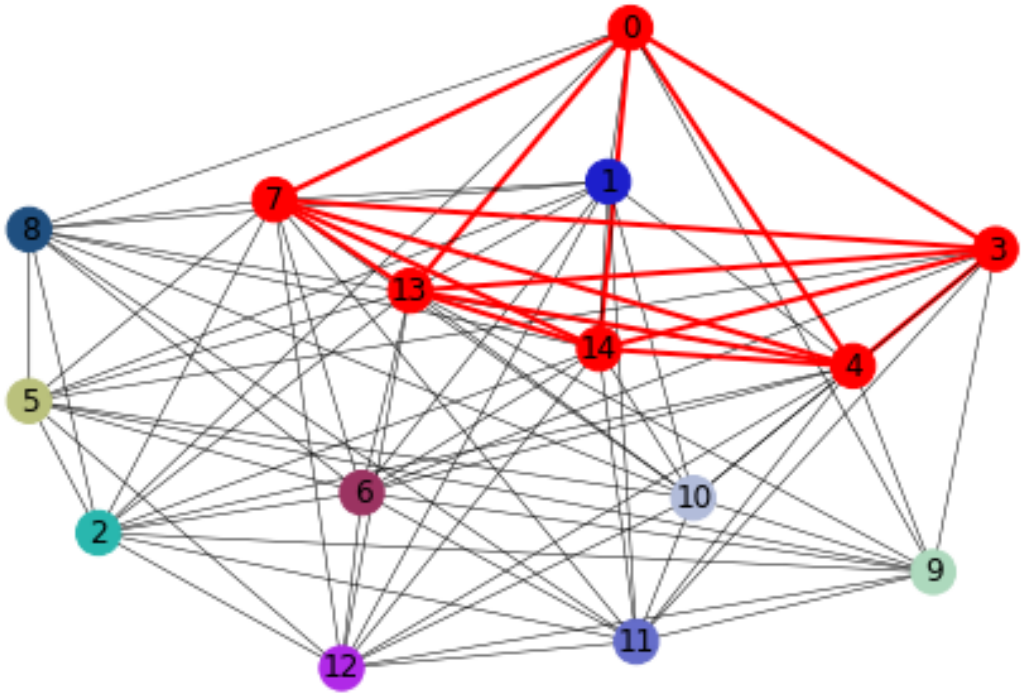

5
12
10
6
1
9
2
7
13
8
4
3
11
14
0

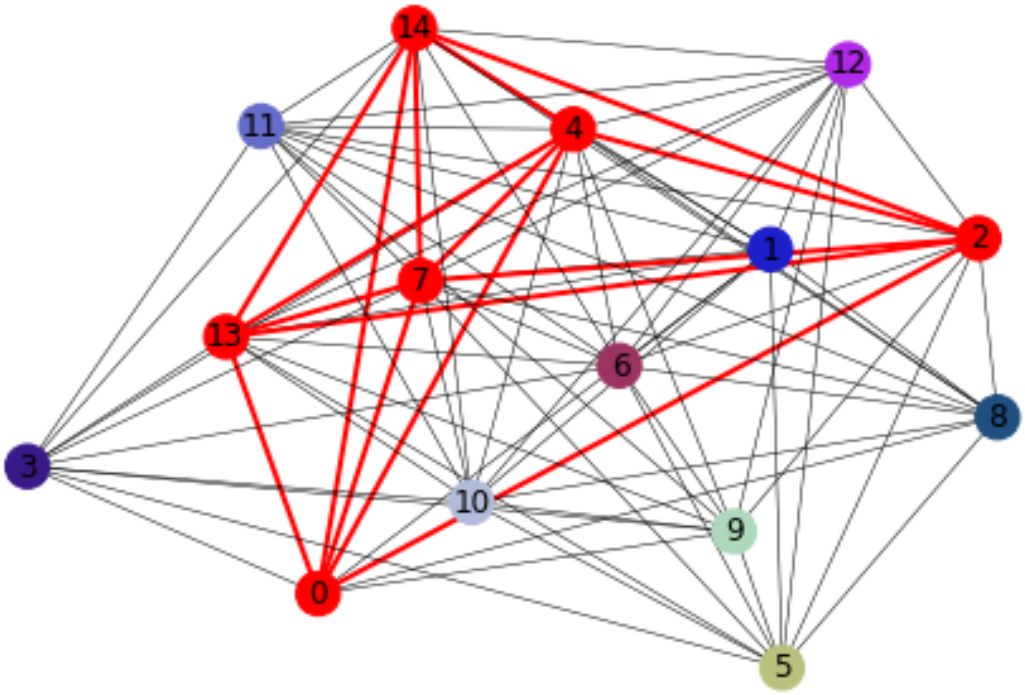

0
3
10
7
14
11
9
13
1
4
2
5
12
8
6

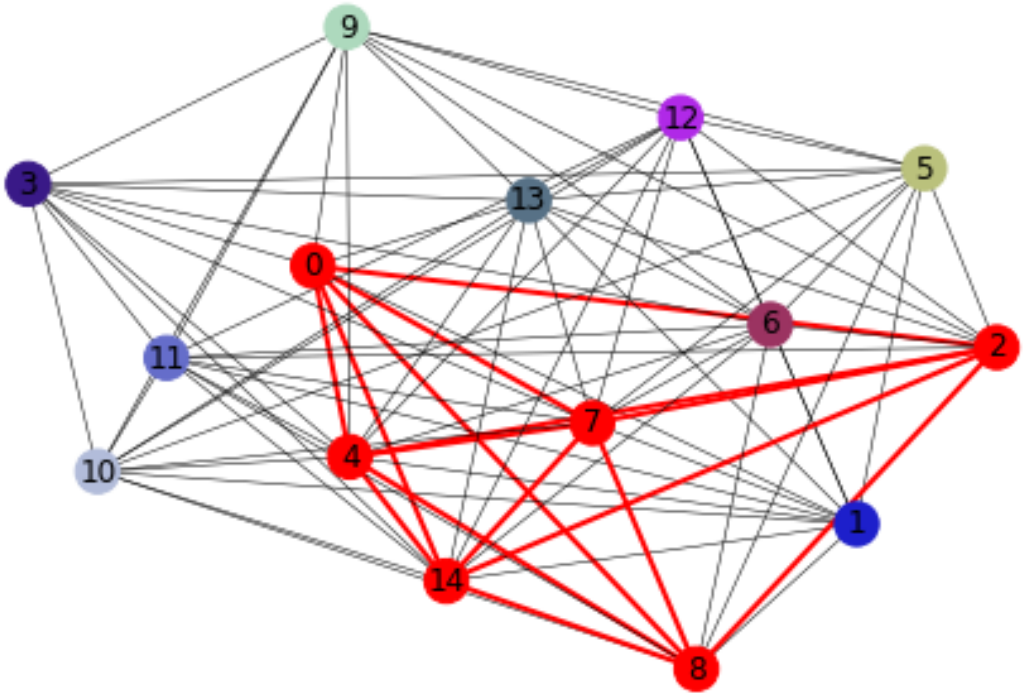

3
13
1
10
6
0
4
7
5
14
9
11
12
8
2

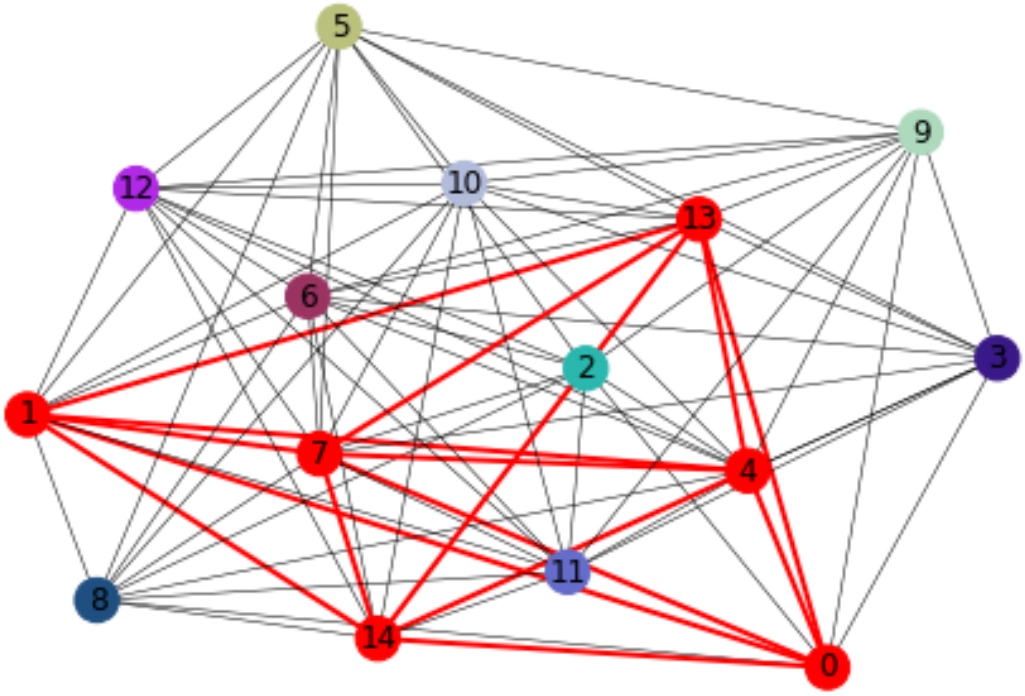

5
12
10
6
13
2
9
7
1
3
11
4
14
8
0

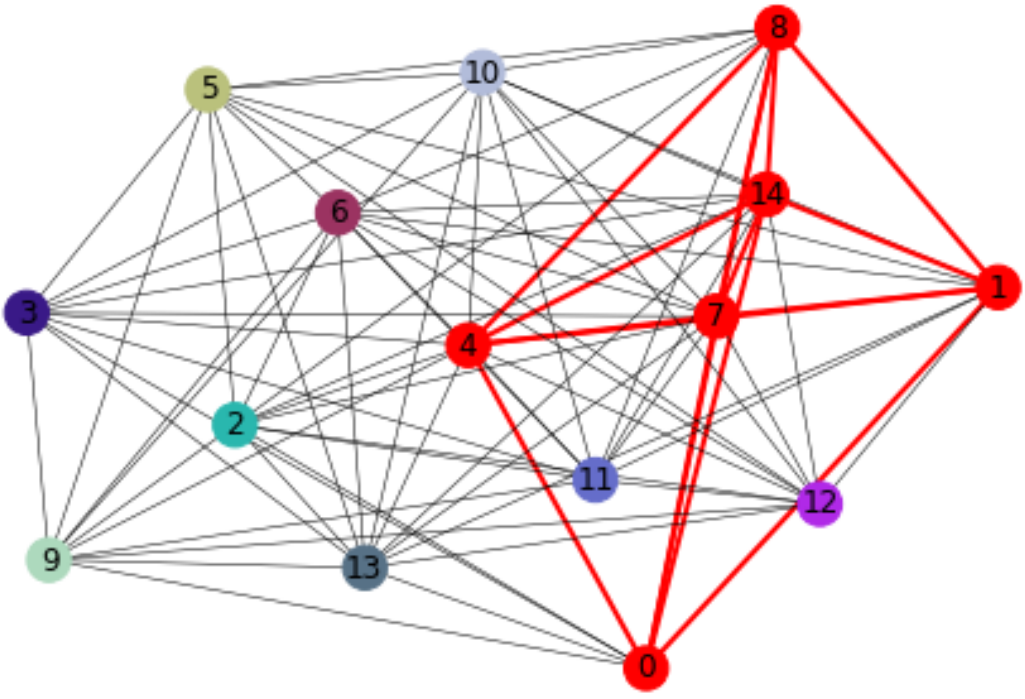

5
1
10
8
6
13
7
3
12
11
14
4
2
9
0

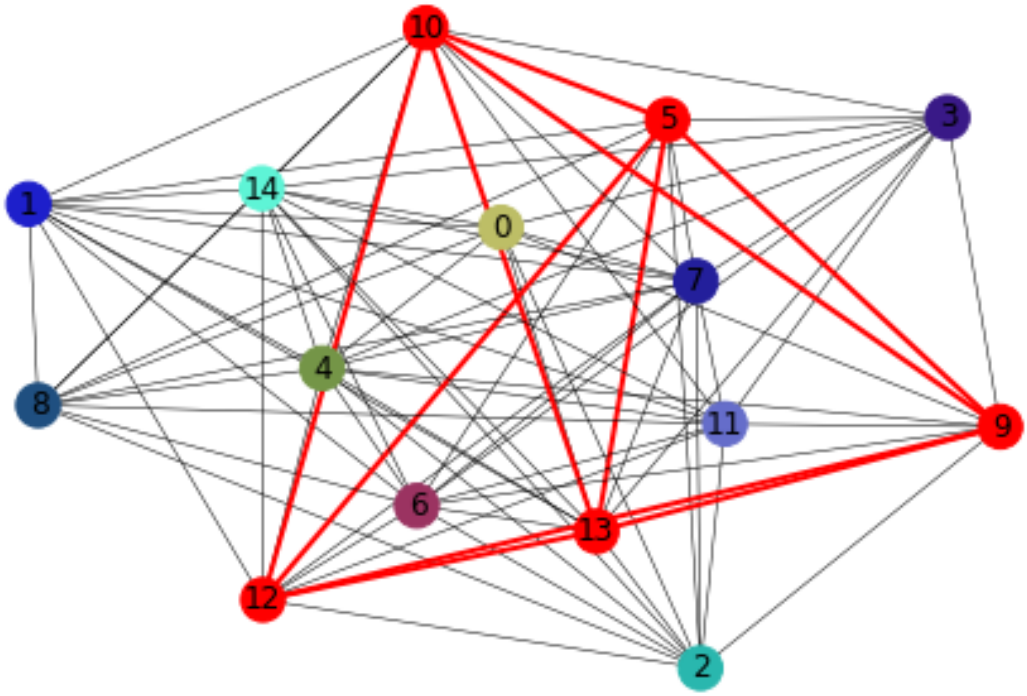

0
7
1
8
3
13
14
4
5
6
10
2
9
11
12

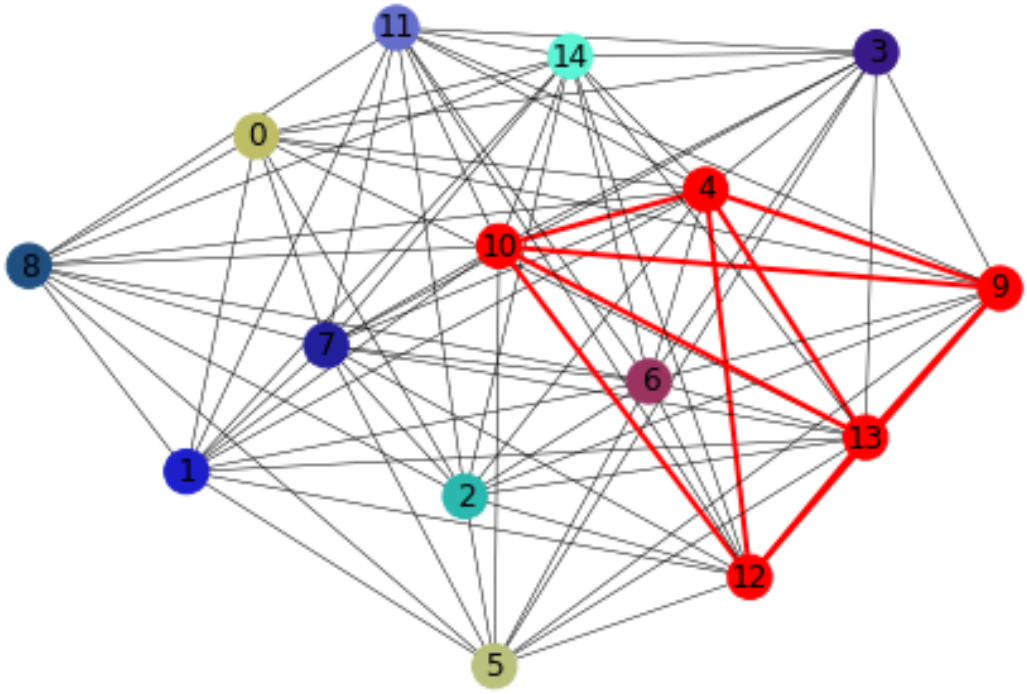

14
12
11
4
2
1
7
13
6
8
3
10
9
0
5

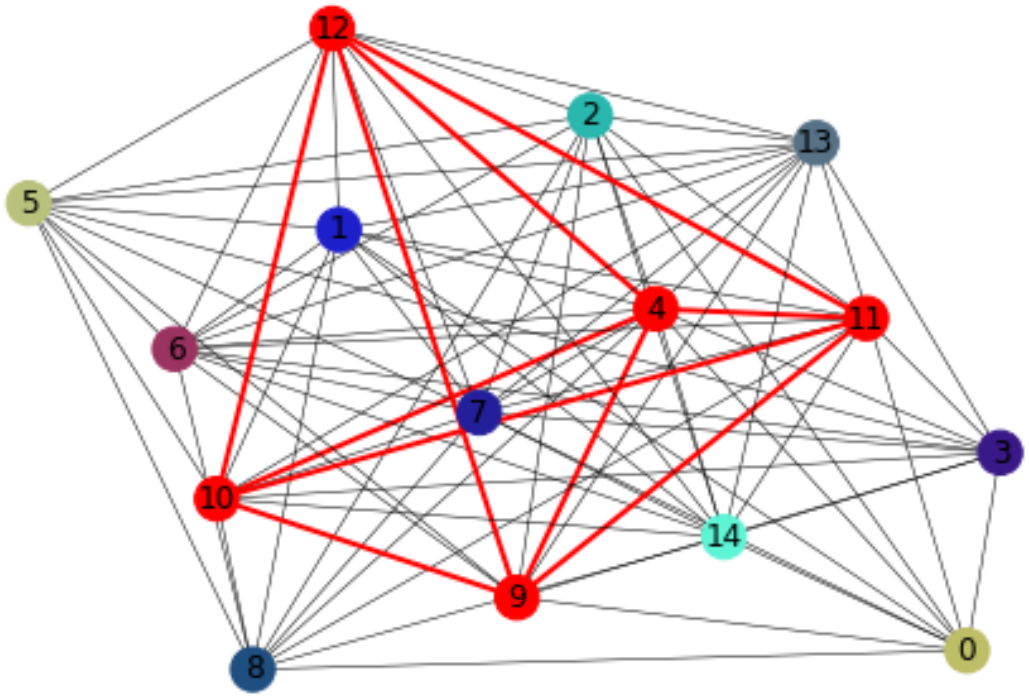

9
12
3
5
13
0
6
2
11
7
4
10
1
14
8

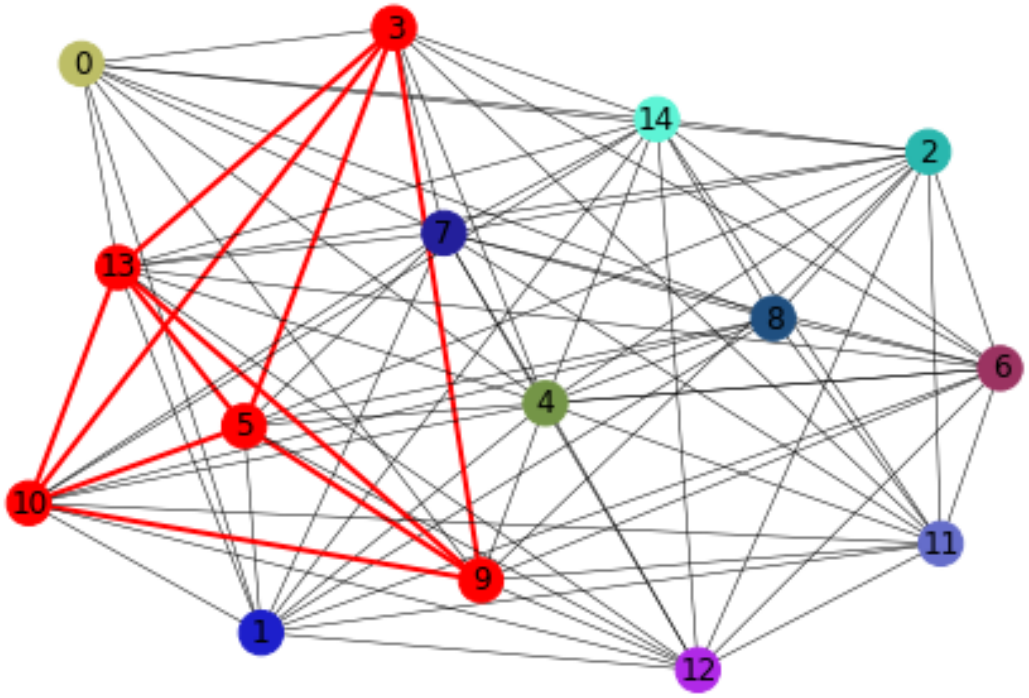

5
9
12
10
13
6
3
2
1
7
4
11
8
14
0

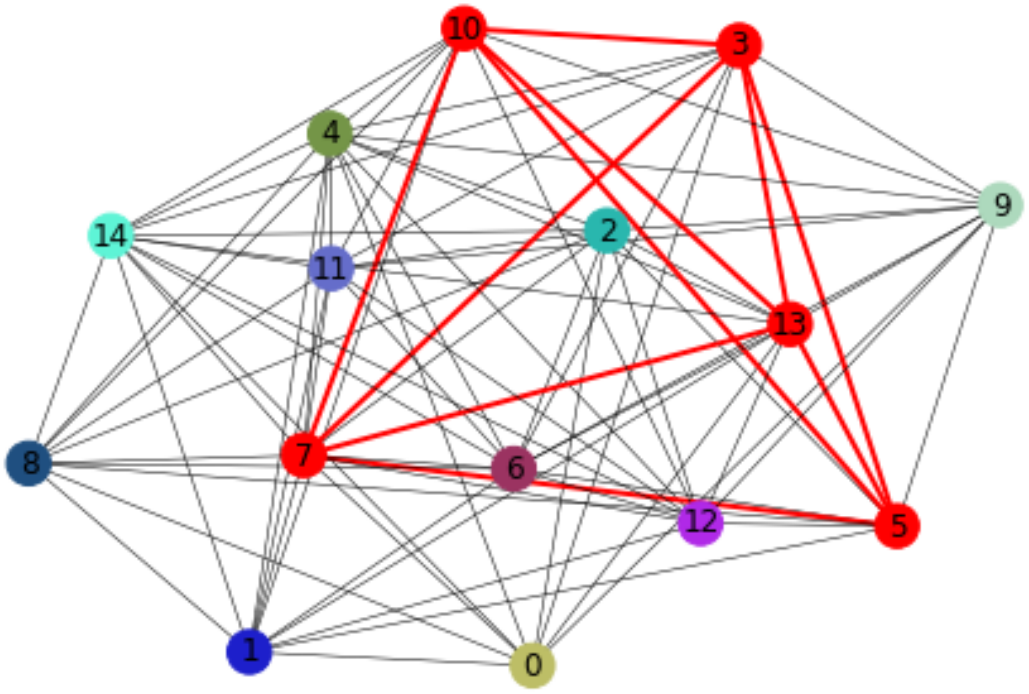

8
10
5
14
6
1
3
7
4
2
11
12
9
13
0

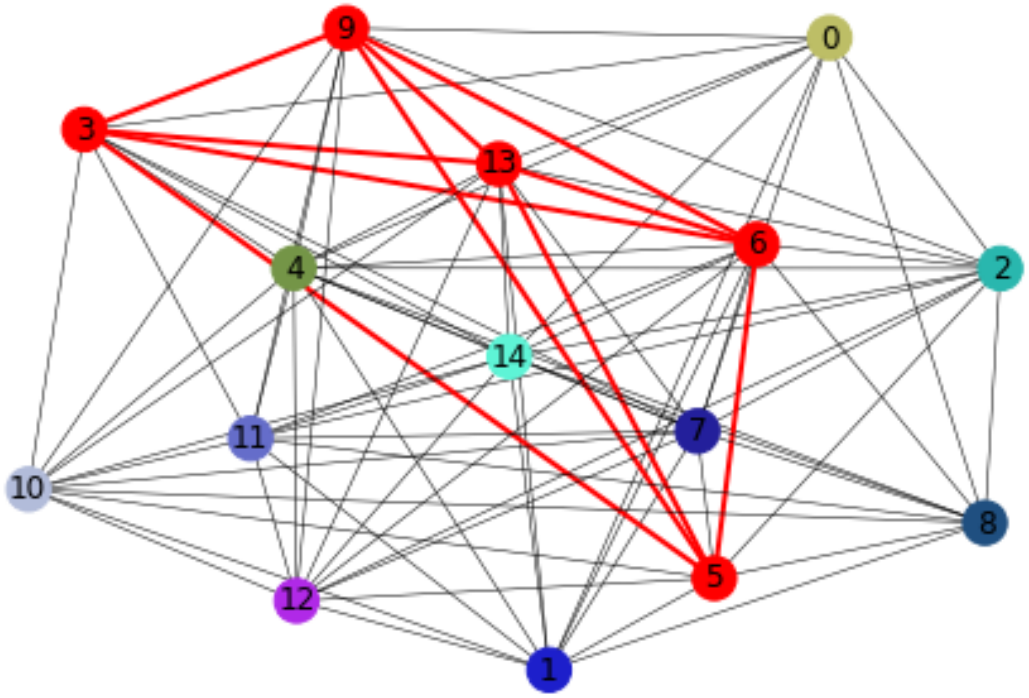

10
5
3
14
1
0
7
4
8
11
9
6
13
12
2

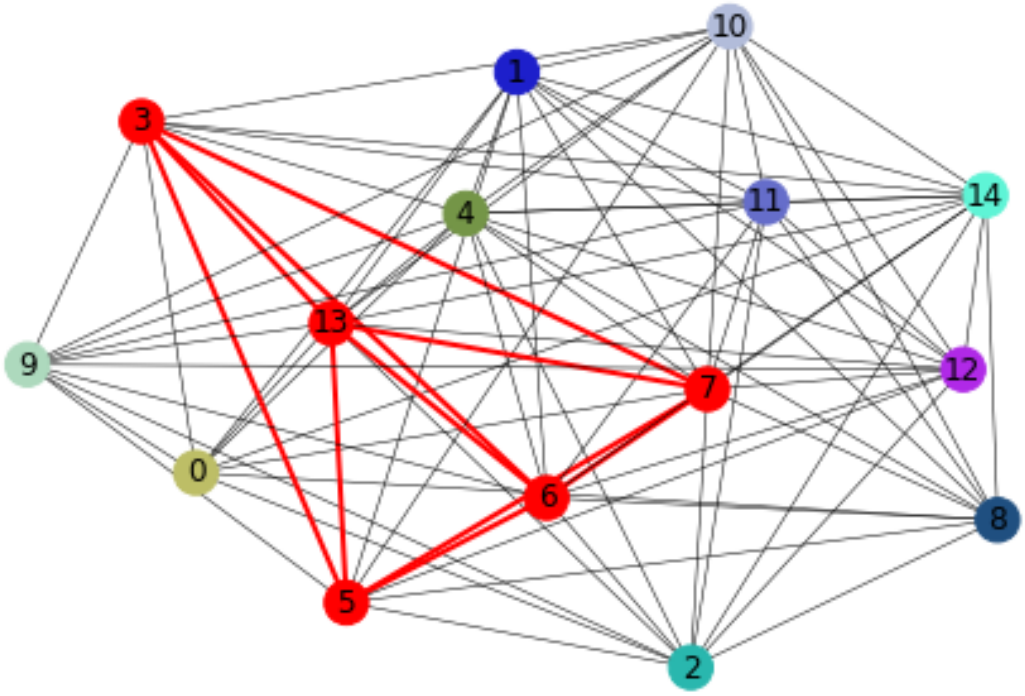

11
14
3
0
4
10
8
9
7
6
13
1
2
12
5

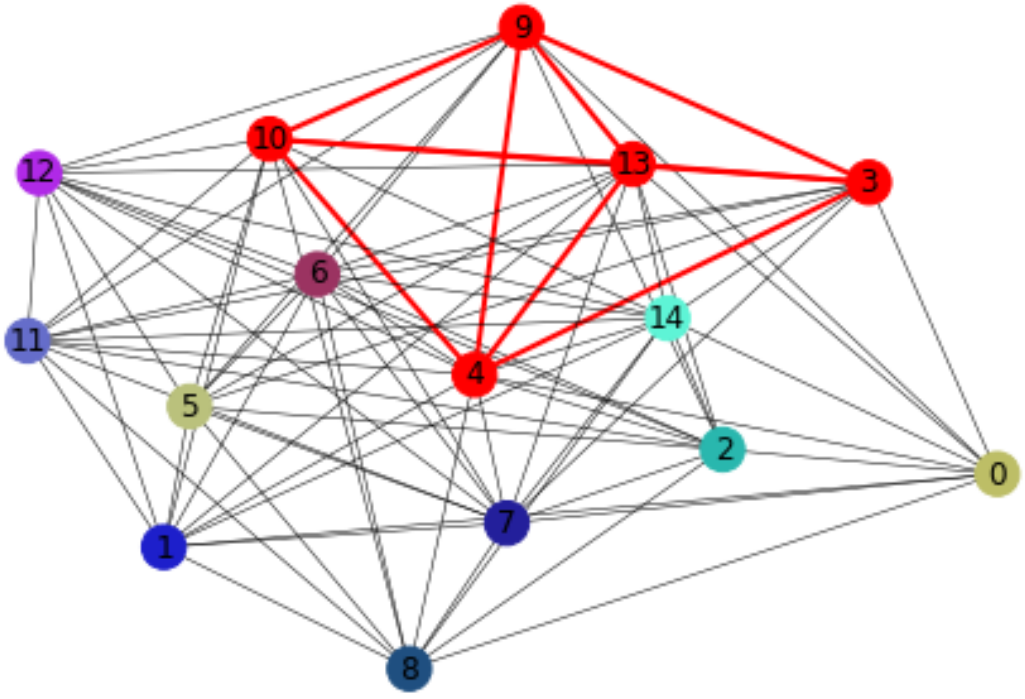

12
2
13
5
1
4
11
6
7
3
10
14
9
0
8

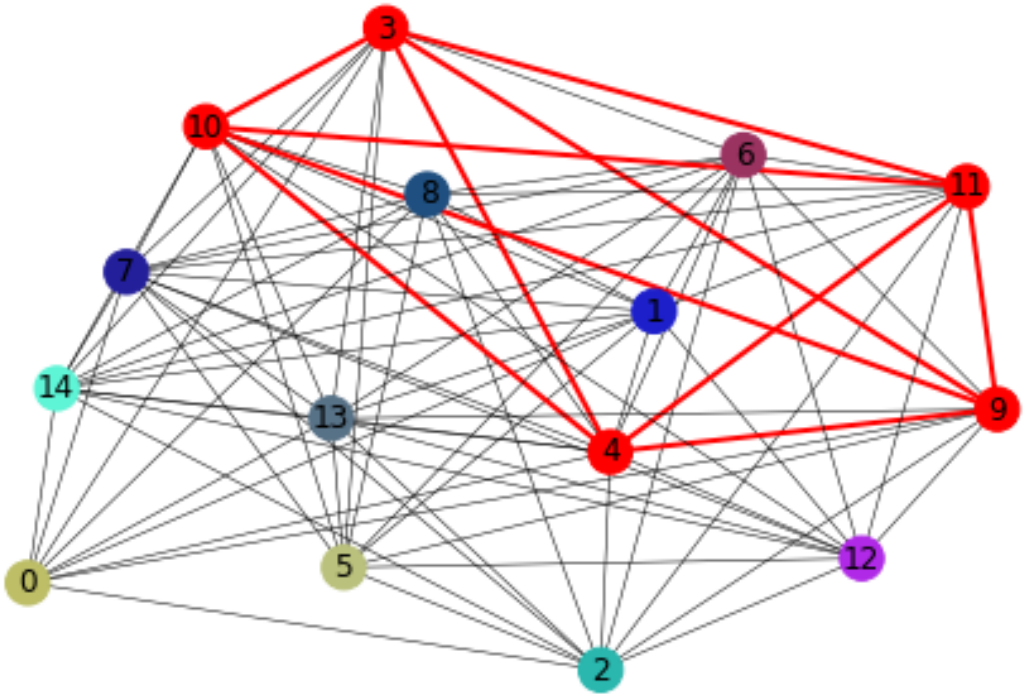

3
0
14
2
7
13
8
6
4
5
10
11
9
1
12

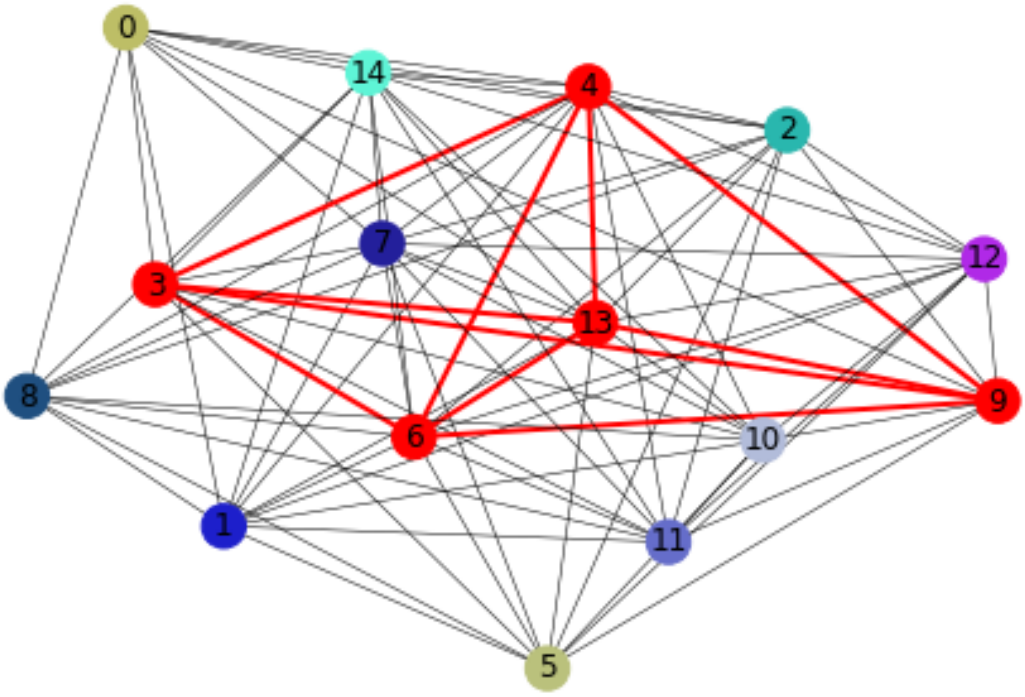

10
3
4
9
14
2
11
13
8
7
6
12
5
1
0

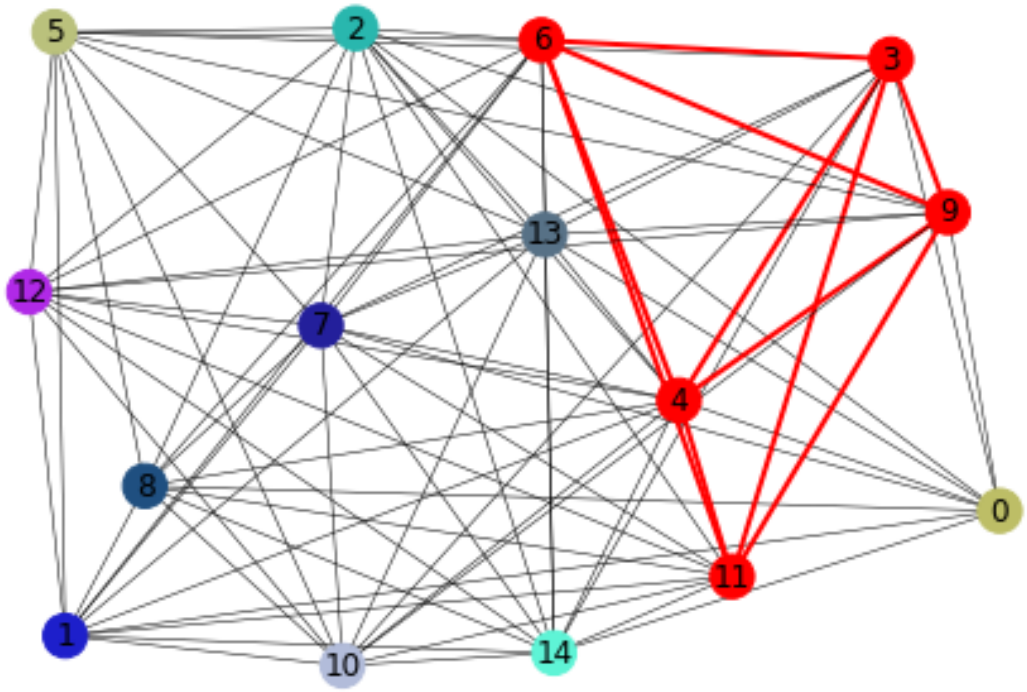

9
0
3
13
6
4
2
14
11
7
10
8
5
12
1

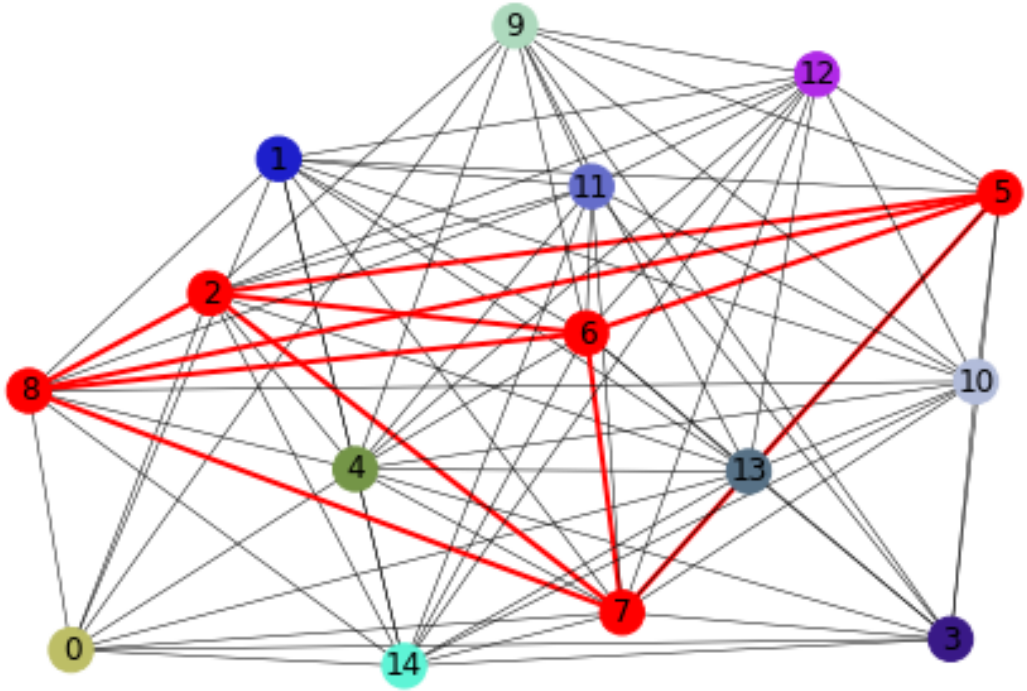

10
1
3
4
11
14
13
9
12
7
0
6
8
5
2

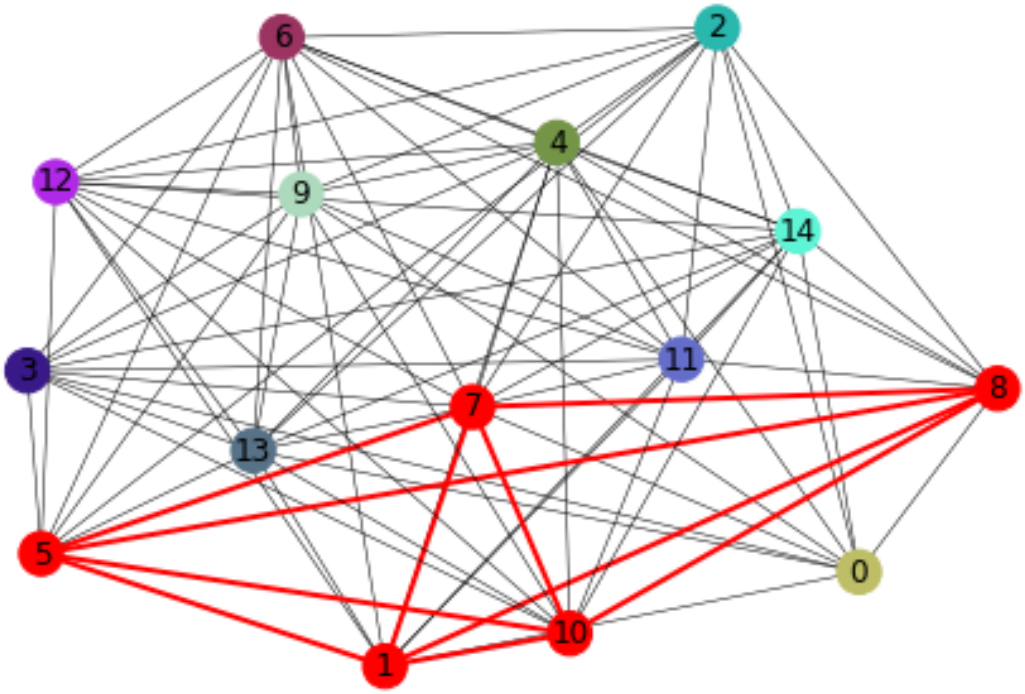

9
10
13
12
3
6
14
11
4
5
2
0
7
1
8

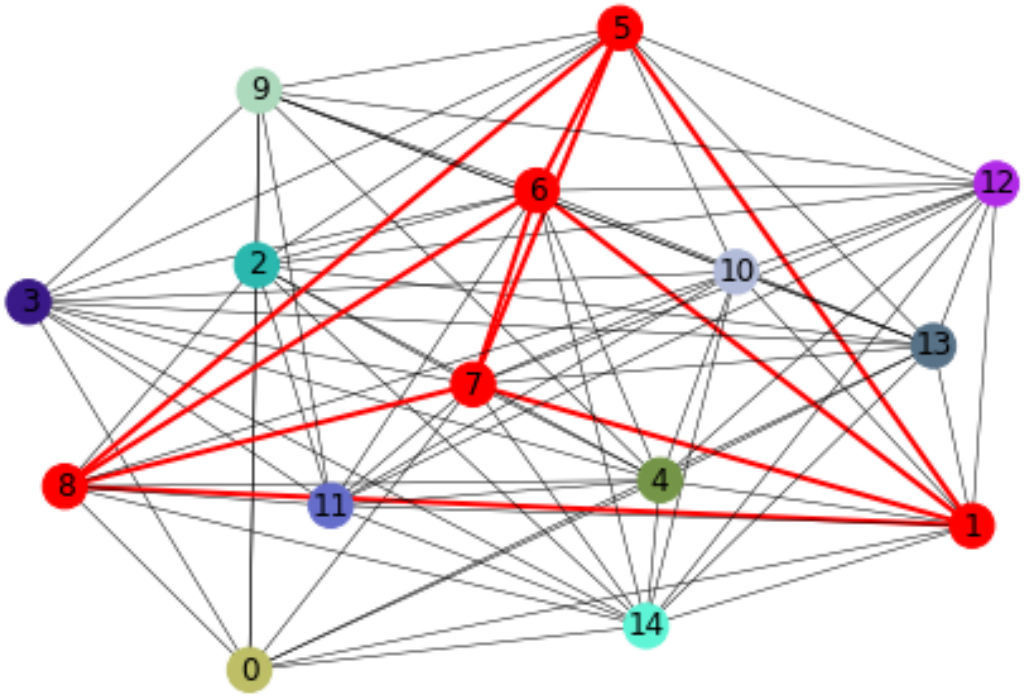

3
10
6
11
8
7
1
14
13
9
4
0
5
12
2

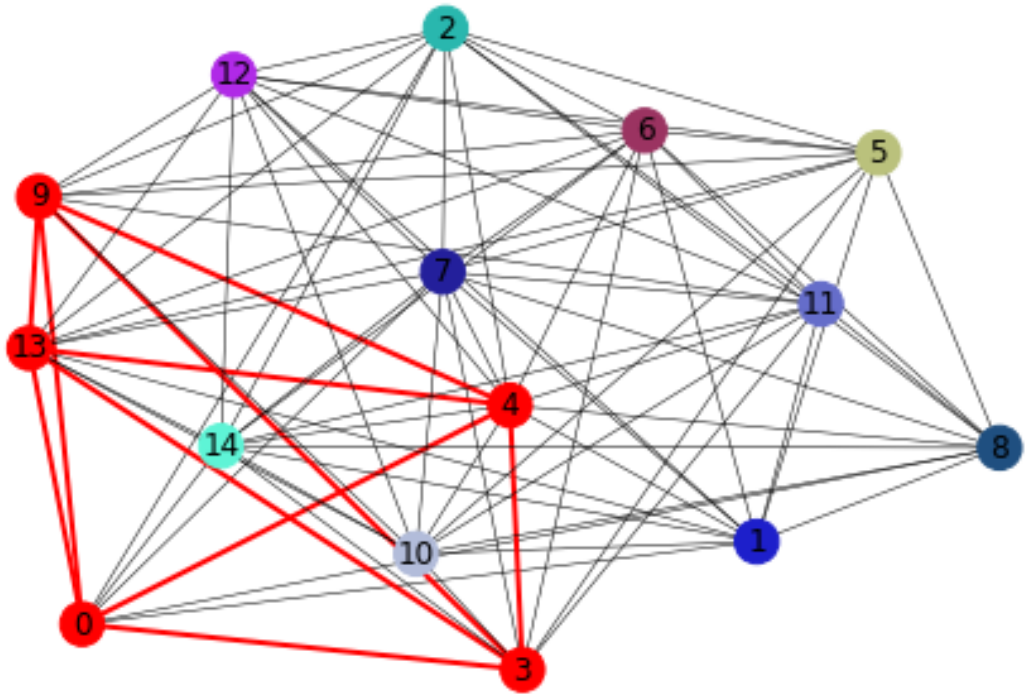

0
14
4
2
7
12
3
13
8
9
6
10
1
11
5

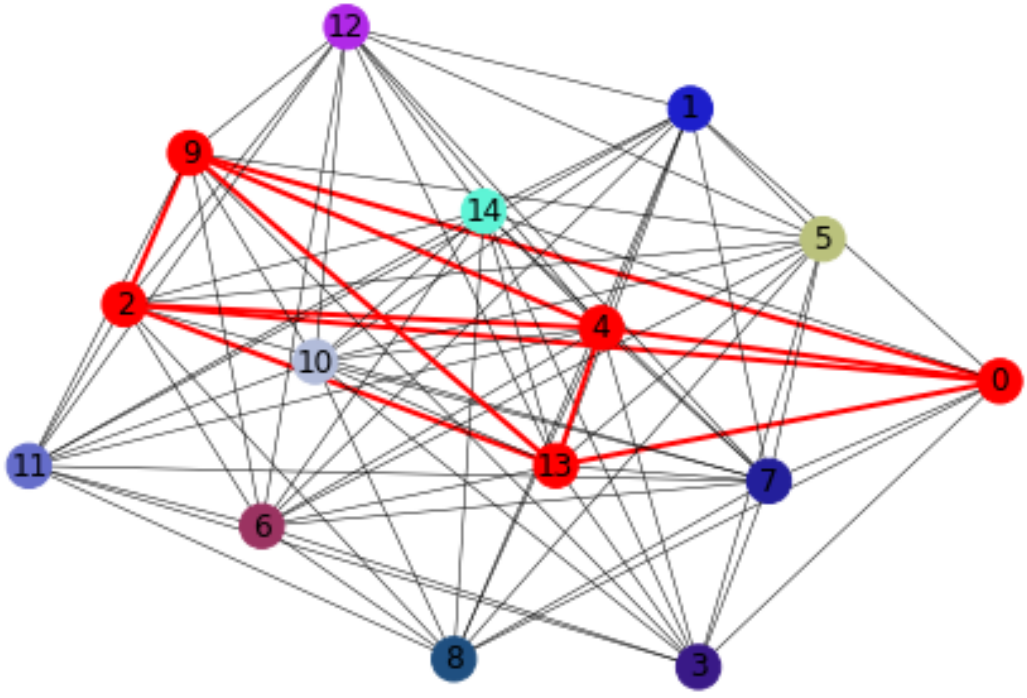

5
2
6
3
9
13
12
7
4
8
0
11
1
10
14


9
12
1
11
5
2
6
8
10
4
13
7
0
14
3


6
2
4
12
9
14
3
11
8
7
13
5
0
10
1


5
9
6
12
2
10
3
13
7
8
4
11
1
14
0


2
12
6
5
9
7
11
13
4
14
8
10
1
0
3


12
1
9
14
5
2
4
10
0
11
13
7
6
8
3

La determinación de que un grafo no sea plano a partir del teorema de Kuratowski para el caso de K_5 se debe revisar si existen cliques de cardinalidad 5.

[81]:
```
es_plano = True
for i in cliques:
    if len(i.nodes) == 5:
        es_plano = False
        break

print(es_plano)
```

```
False
```