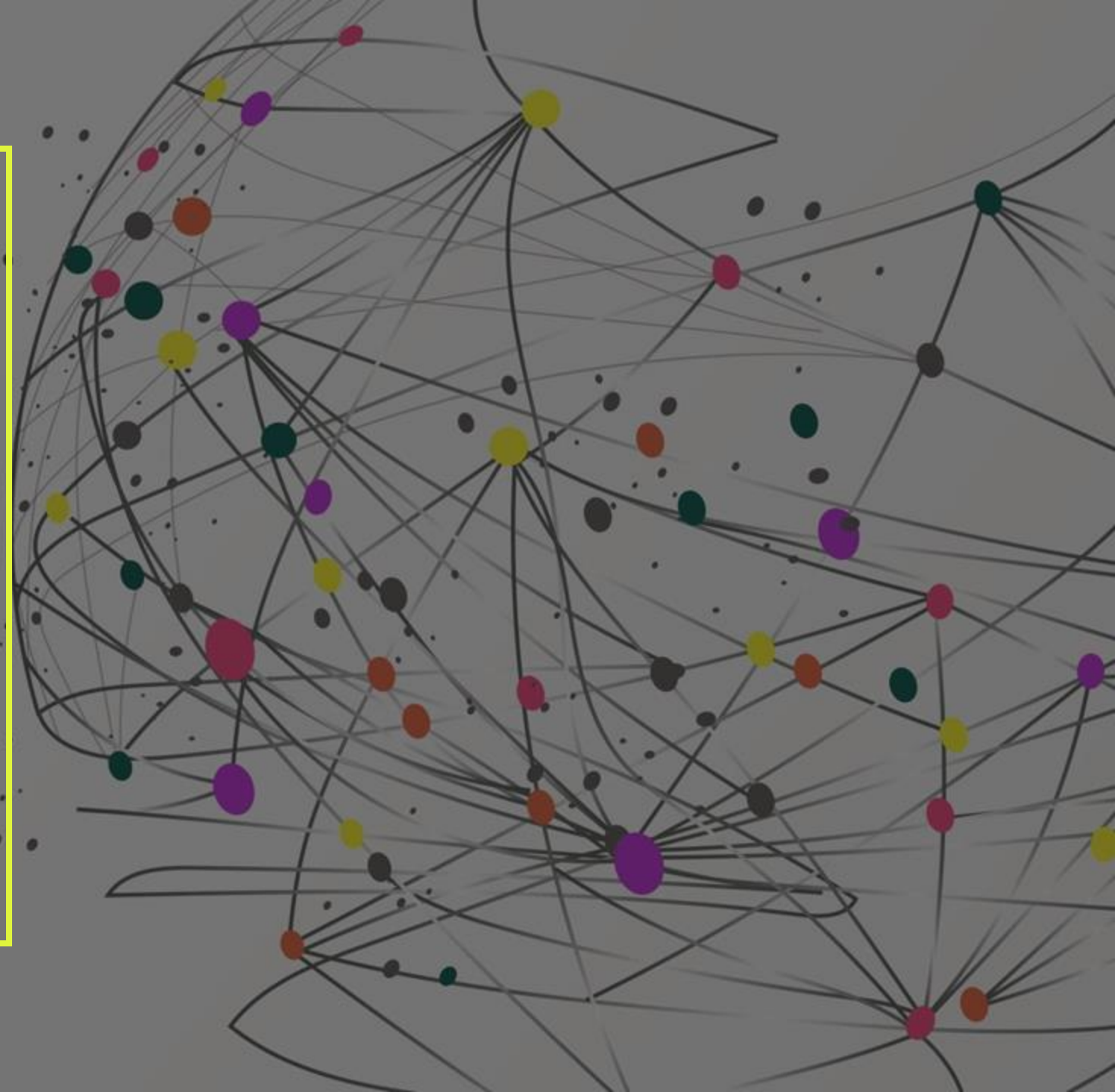
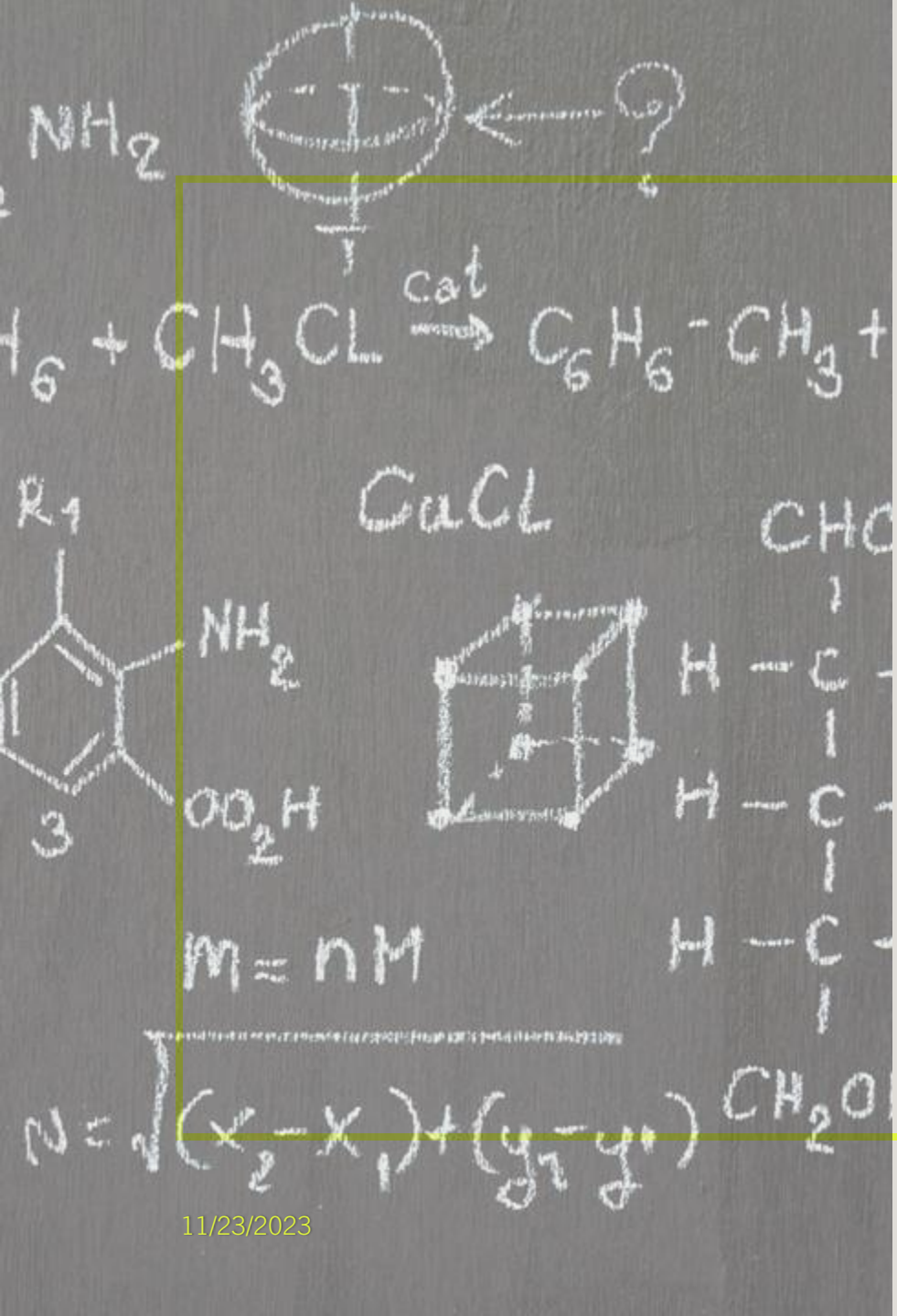
VISUALIZAÇÃO GRÁFICA DE GRAFOS COM COMUNIDADES ESPECIFICADAS USANDO O ALGORITMO RELAXMAP

Integrantes: Dhruv Babani, Bernado Balzan, Eduardo Cardoso





INTRODUÇÃO

- O estudo apresenta a visualização dos grafos com as suas comunidades detectadas, utilizando tanto o algoritmo de RelaxMap quanto a biblioteca Toyplot do Python

- Sabendo disso, é realizado a comparação dos dois resultados e é concluido a melhor forma de representar um grafo de forma visual.

SELEÇÃO DE FERRAMENTAS

- NodeXl -- Aplicativo
- Infomap Online -- WEB

EXPLORANDO O PYTHON

- Após a escolha do Infomap, como a ferramenta primária, foi realizado mais pesquisas, com o intuito de aprimorar a visualização da estrutura de dados. Logo, foi encontrado uma biblioteca da linguagem Python para manipulação de Grafos e Redes, chamada Toyplot.
- Além disso ela possui uma vasta implementação a séries de algoritmos de detecção de comunidades. Portanto, decidimos que vai ser feito o uso dessa biblioteca para cumprir o propósito da pesquisa.

EXPERIMENTOS OFICIAIS

- Agora que temos todos os recursos para realizar os experimentos. Sabendo disso, o experimento em si, se divide em três passos:

  -Seleção dos Grafos

  -Implementação do Script em Python, usando o Toyplot

  -Representação visual dos Grafos no Infomap

SELEÇÃO DOS GRAFOS

Graph	Nodes	Edges
bio-celegans-dir	453	4.6k
ENZYMES-g228	34	152
ia-southernwomen	50	230

```
import numpy as np
import networkit as nk
import os

def str_list_to_int(str_list):
    return [int(item) for item in str_list]

def read_edges_from_file(filename):
    with open(filename, "r") as f:
        lines = f.readlines()
        edges = [str_list_to_int(line.split()) for line in lines]
    return edges

def get_nk_communities():
    nk_graph = nk.graph.Graph()
    edges = []

    for edge in read_edges_from_file(os.path.dirname(__file__) + '/road-euroroad/road-euror
        nk_graph.addEdge(edge[0], edge[1], 0, True)
        edges.append([edge[0], edge[1]])
```

VISUALIZAZAÇÃO-TOYPLOT

```
communities = nk.community.detectCommunities(nk_graph, nk.community.PLP(nk_graph))

communities_keys = []

for community in communities.getVector():
    if community not in communities_keys and community != 0:
        communities_keys.append(community)

truth = [[]] * (nk_graph.numberOfNodes() - 1)

for index, community in enumerate(communities_keys):
    for member in communities.getMembers(community):
        truth[member - 1] = [member, index + 1]

return (edges, truth)

dges, truth = get_nk_communities()

olormap = tp.color.brewer.map("Set1")
color = "lightgray"
lstyle = {"fill":"white"}
color = np.array(truth)[:,1]

raph = tp.graph(np.array(edges), ecolor=ecolor, vsize=20, vlstyle=vlstyle, vcolor=(vcolor, colormap))
```

VISUALIZAZAÇÃO-TOYPLOT

Load network
Edit network or load file

Run Infomap
Toggle parameters or add arguments

Explore map!
Save result or open in Network Navigator

Load network

--clu --ftree

Run Infomap

Open in Navigator

```
#source target [weight]
1 2
1 3
1 4
2 1
2 3
3 2
3 1
4 1
4 5
4 6
5 4
5 6
6 5
6 4
```

Load network by dragging & dropping.
Supported formats

```
Trial 1/1 starting at 2023-11-14 09:39:21
Two-level compression: 9.2% 0%
Partitioned to codelength 0.142857143 + 2.17787321 = 2.320730357 in 2 modules.
Super-level compression: to codelength 2.320730357 in 2 top modules.

Recursive sub-structure compression: 0% . Found 2 levels with codelength
2.320730357

=> Trial 1/1 finished in 0.0121s with codelength 2.32073036
Write flow tree to ./two_triangles.ftree... done!
Write node modules to ./two_triangles.clu... done!


Summary after 1 trial
Best end modular solution in 2 levels:
Per level number of modules:          [          2,          0] (sum: 2)
Per level number of leaf nodes:       [          0,          6] (sum: 6)
Per level average child degree:       [          2,          3] (average: 2.75)
Per level codelength for modules:     [0.142857143, 0.000000000] (sum:
0.142857143)
Per level codelength for leaf nodes: [0.000000000, 2.177873214] (sum:
2.177873214)
Per level codelength total:           [0.142857143, 2.177873214] (sum:
2.320730357)

  Infomap ends at 2023-11-14 09:39:21
  (Elapsed time: 0.0351s)
```

```
# v2.6.1
# ./Infomap two_triangles . --clu --ftre
# started at 2023-11-14 09:39:21
# completed in 0.0347 s
# partitioned into 2 levels with 2 top
# codelength 2.32073 bits
# relative codelength savings 9.2279
# flow model undirected
# module level 1
# node_id module flow
1 1 0.214286
2 1 0.142857
3 1 0.142857
4 2 0.214286
5 2 0.142857
6 2 0.142857
```

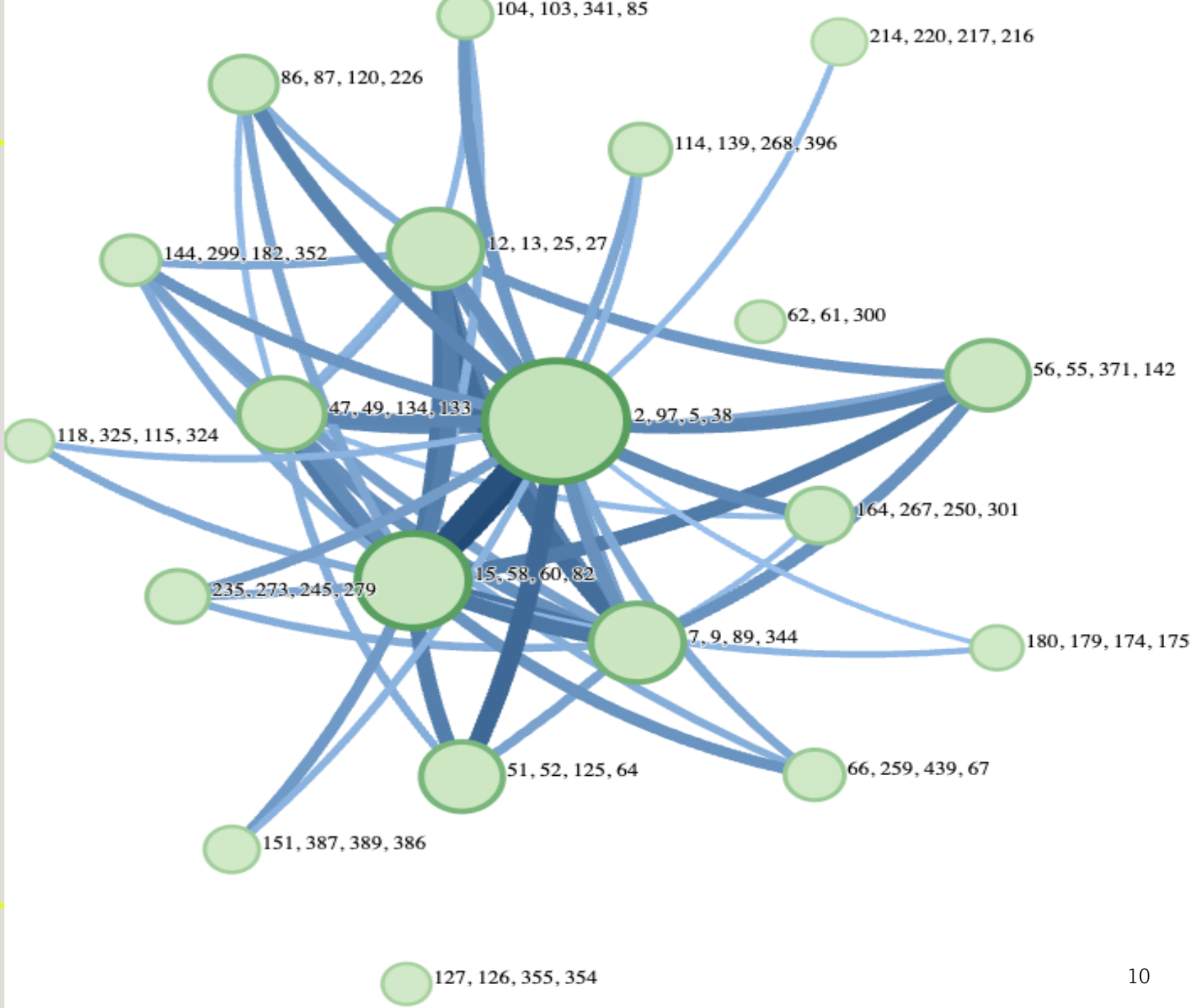
Clu

Ftree

VISUALIZAÇÃO - INFOMAP ONLINE

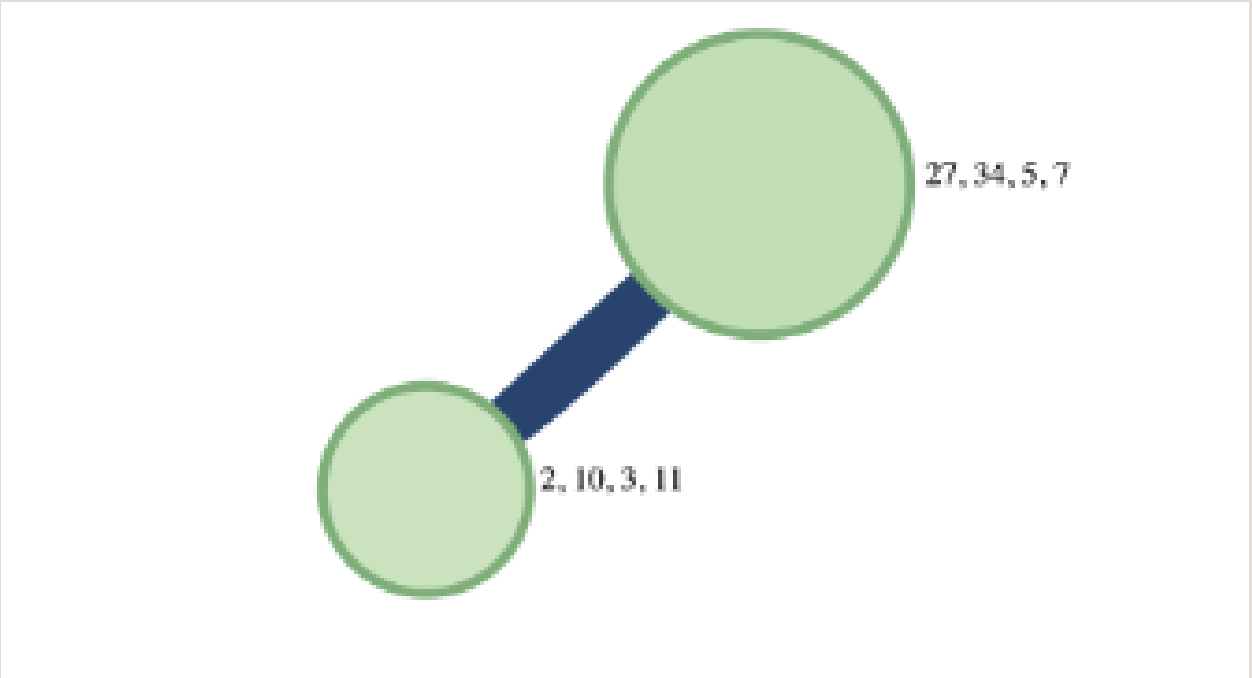
RESULTADOS INFOMAP ONLINE

- bio-celegans-dir:



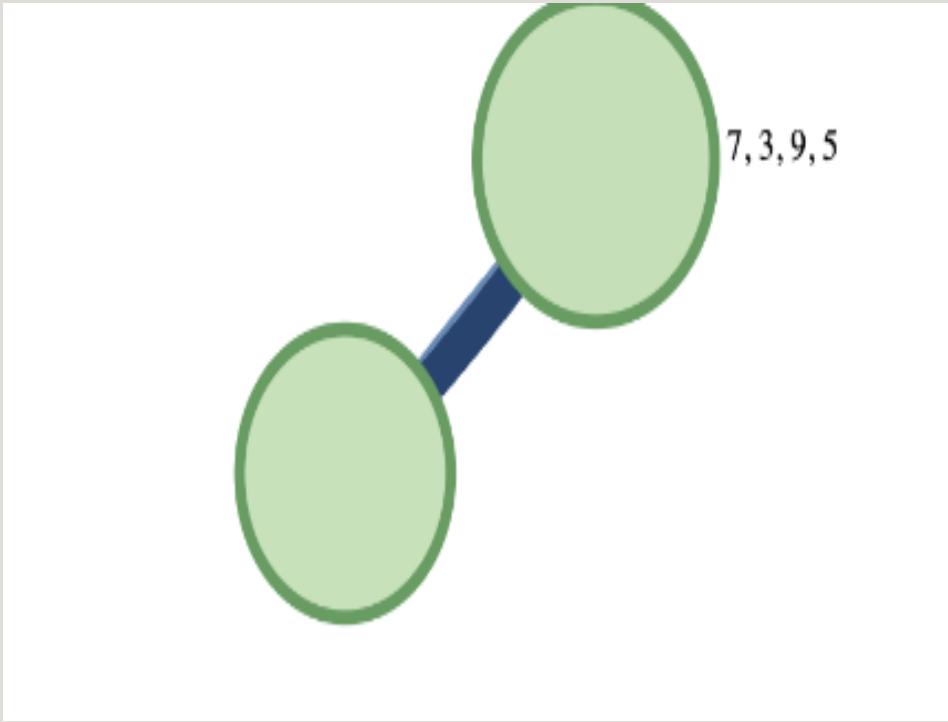
RESULTADOS INFOMAP ONLINE

- ENZYMESg228:



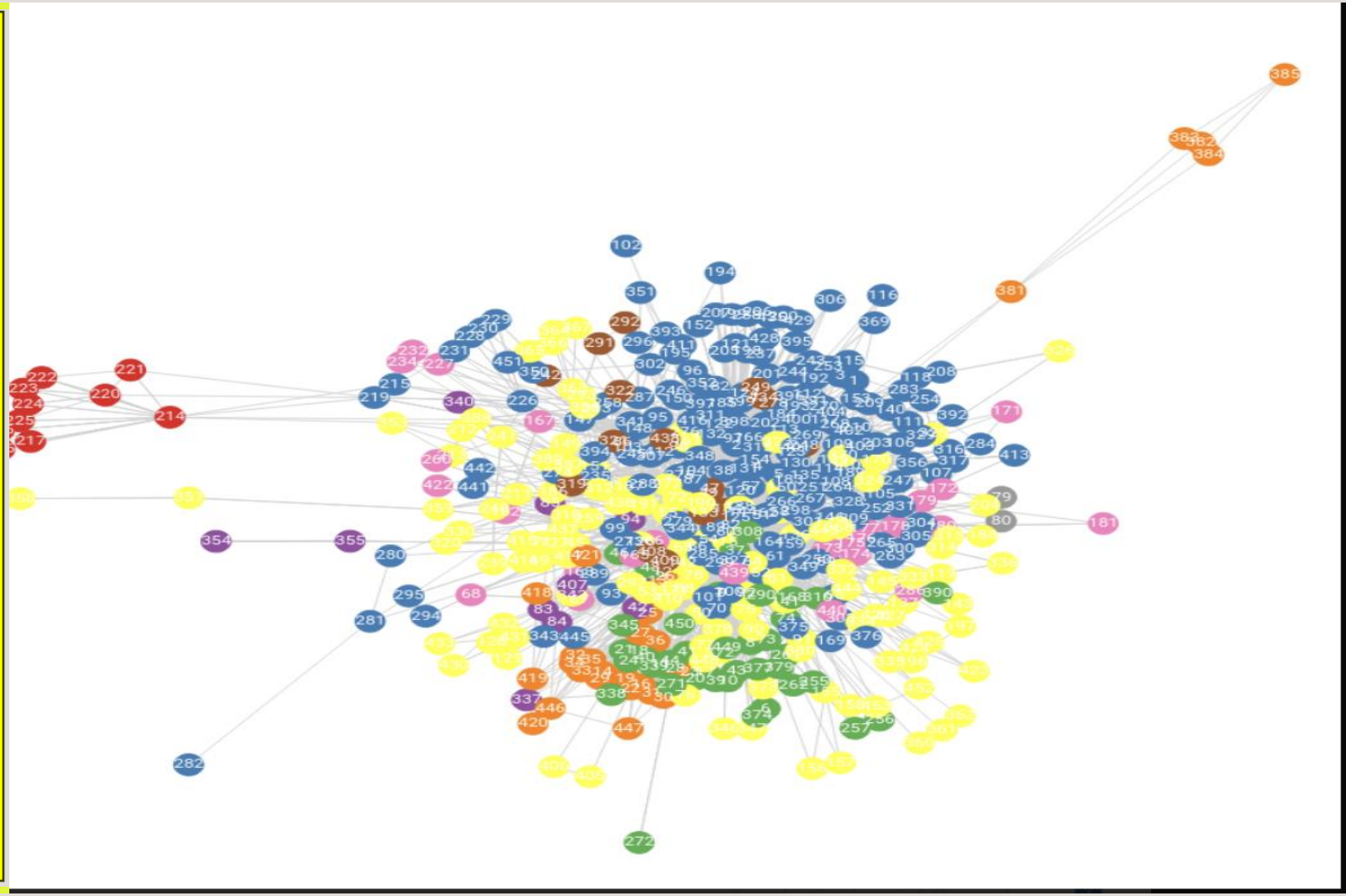
RESULTADOS INFOMAP ONLINE

- IA-southernwomen:



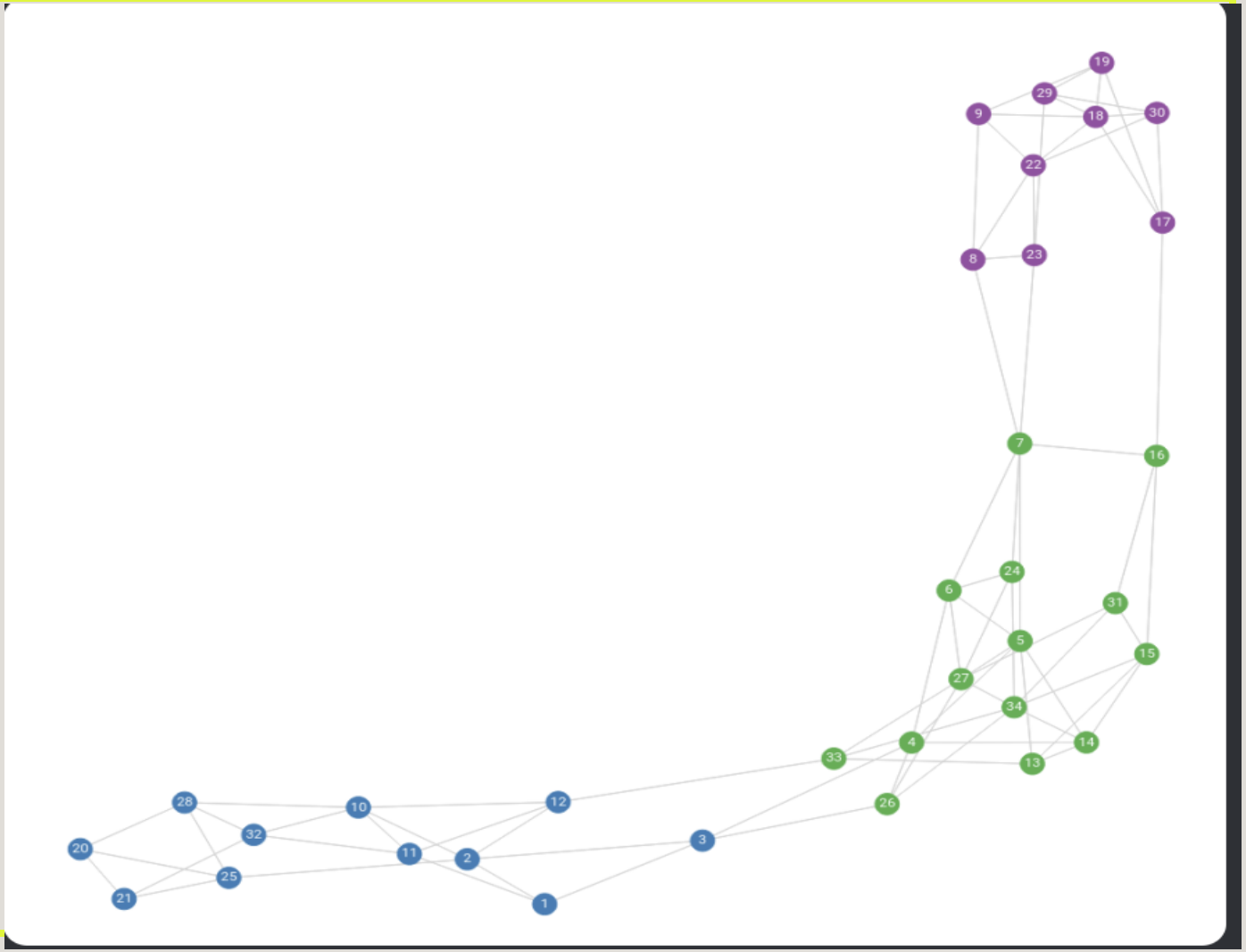
RESULTADOS TOYPLOT

bio-celegans-dir:



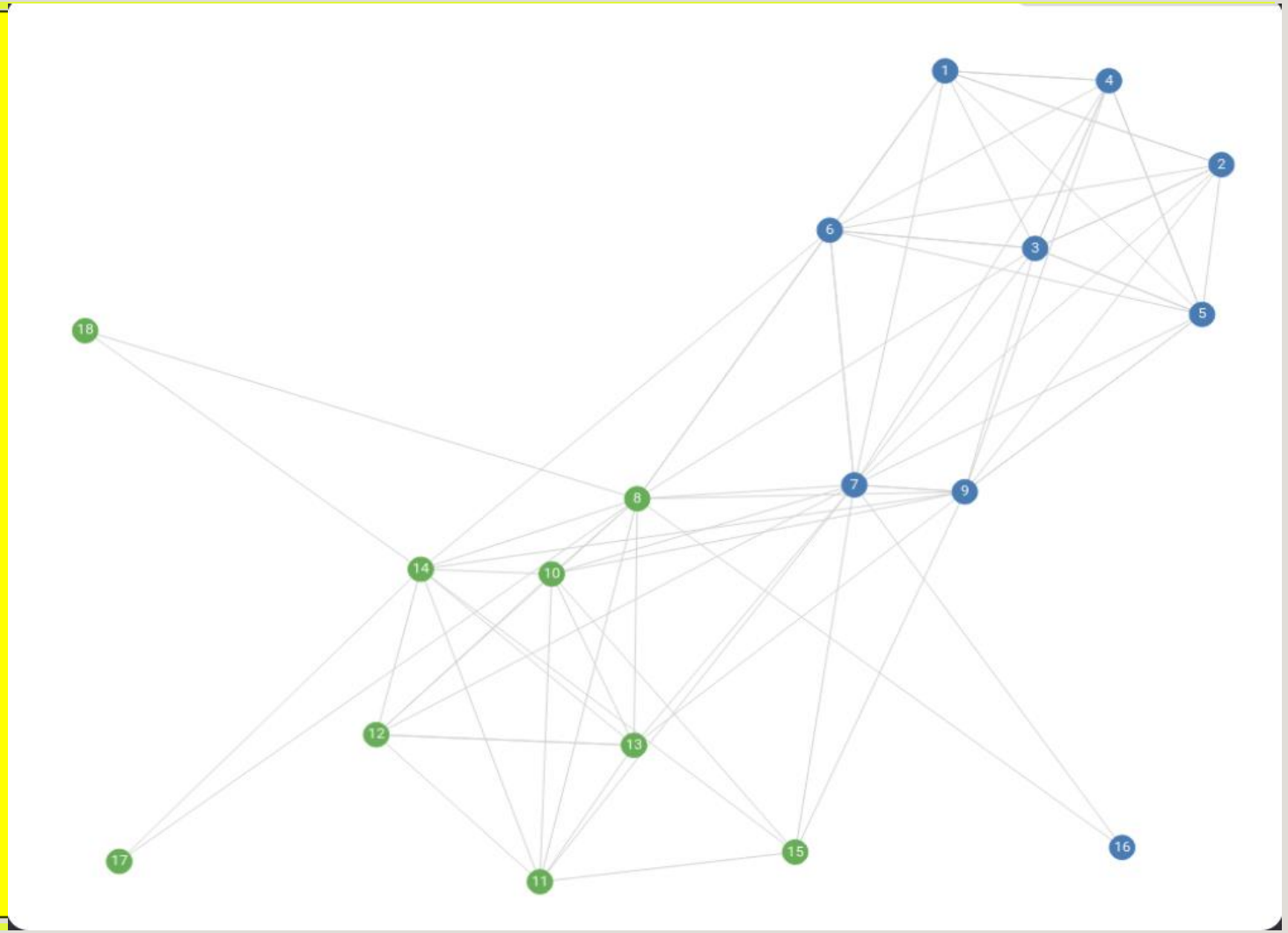
RESULTADOS TOYPLOT

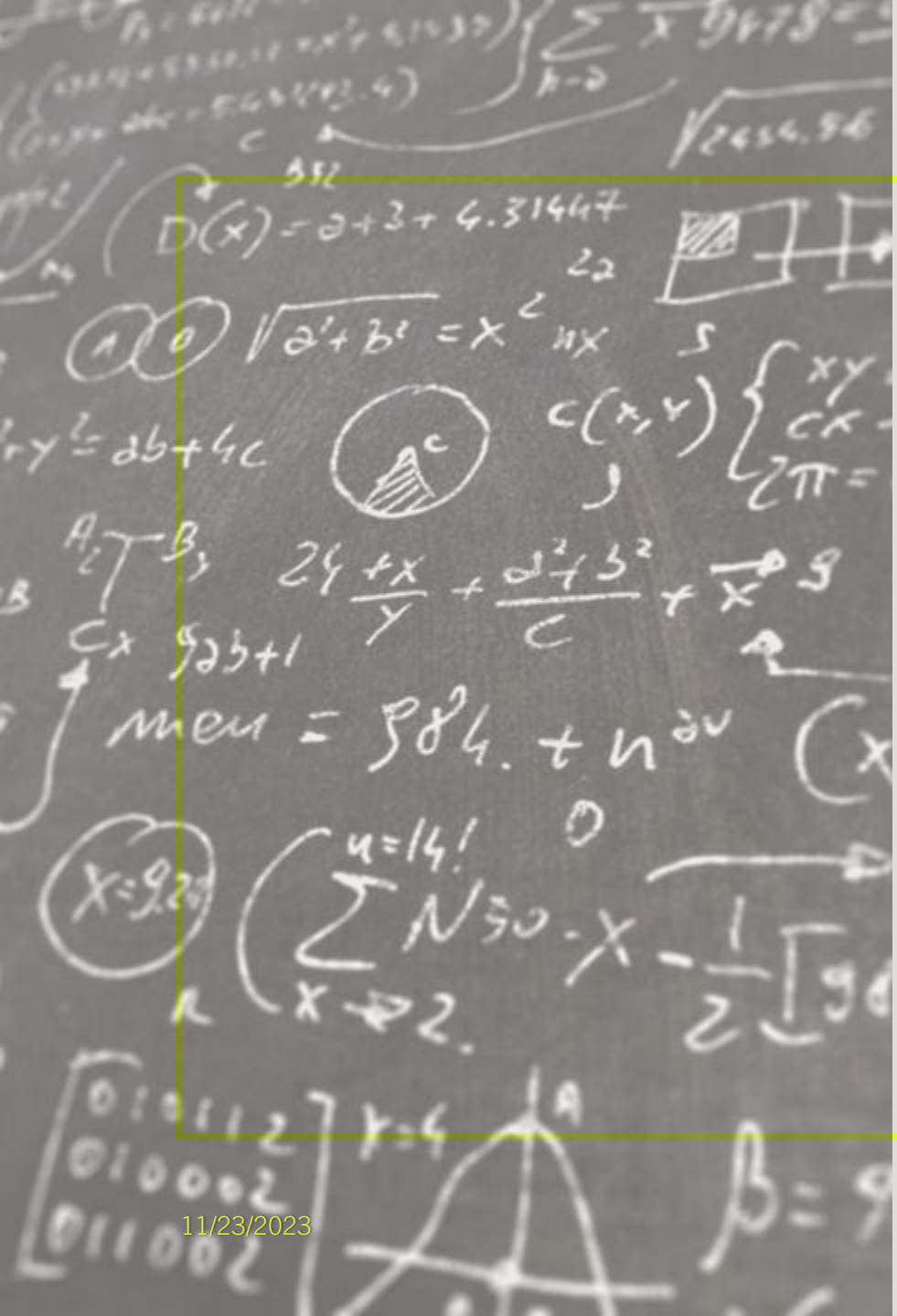
ENZYMESg228:



RESULTADOS TOYPLOT

IA-southernwomen:





CONCLUSÕES

- Á nossa perspectiva, o estudo realizado neste trabalho foi muito gratificante, porque nos levou a concluir que a visualização dos grafos é realmente uma ótima alternativa, quando queremos entender um cenário dentro do universo da computação, principalmente no estudo de Escalabilidade Paralela.
- Além disso, podemos concluir que entre os dois tipos de visualização gerados, a alternativa do Toyplot se apresenta bem mais satisfatório em relação ao que foi gerado pelo Infomap Online, devido ao numero de detalhes e a divergência de cor na composição do Grafo. Logo, este estudo conclui que quando o problema é visualizar os grafos, a melhor alternativa é o uso da biblioteca Toyplot do Python.

REFERENCES

- Giordani, G. (2021). Towards a scalable parallel infomap algorithm for community de- tection. page 4. [Giordani 2021]

VISUALIZAÇÃO GRÁFICA DE GRAFOS COM COMUNIDADES ESPECIFICADAS USANDO O ALGORITMO RELAXMAP

Integrantes: Dhruv Babani, Bernado Balzan, Eduardo Cardoso