Trabalho Computacional 3
Algoritmos e Estruturas de Dados II

Davi Juliano Ferreira Alves
Instituto de Ciência e Tecnologia
Bacharelado em Ciência e Tecnologia
Universidade Federal de São Paulo
Av. Cesare Monsueto Giulio Lattes, 1201, 12247-014
Email: davi.juliano@unifesp.br

***Abstract*—A utilização de grafos, hoje em dia, tem-se intensificado de acordo com a complexidade dos problemas que se há de resolver. Redes e mapas são definidas matematicamente por meio dessa área, ou seja, problemas também podem ser definidos matematicamente por meio dos grafos. Redes de relacionamentos podem ser definidas por meio de redes sociais que "mapeiam" os circulos sociais que serão estudados e os pesos são dados de acordo com a proximidade de um circulo social com outro. O estudo desse trabalho é sobre as relações de centralidade e de relação com os circulos sociais mais influentes, utilizando o Facebook© como rede social.**

I. Introdução

De acordo com [1], o prefeito de uma cidade próxima a Königsberg enviou uma carta ao matemático suíço em nome de Heinrich Kiihn, um professor de matemática local. As 8 mensagens trocadas inicialmente não foram recuperadas, mas uma carta datada de 09 de março de 1736 indica que eles haviam discutido o problema. O problema definido por Kiihn é dado pela seguinte carta:

"Um problema me foi apresentado sobre uma ilha na cidade de Konigsberg, cercada por um rio, atravessado por sete pontes, e foi me perguntado se alguém poderia atravessar as pontes separadas em uma caminhada contínua de tal forma que cada ponte fosse atravessada apenas uma vez. Fui informado que até então ninguém havia demonstrado a possibilidade de fazer isso, ou mostrado que é impossível. Esta questão é tão banal, mas pareceu-me digno de atenção em que nem a geometria, álgebra, ou mesmo a arte de contar foram suficientes para resolvê-lo".

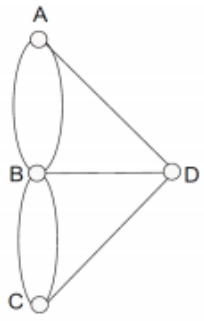

Fig. 1. Grafo que representa a cidade de Königsberg (Fonte:[1]).

Verificamos, então, que a partir do Problema das Pontes de Königsberg, Euler sistematizou um novo campo da Matemática - era o surgimento da Teoria dos Grafos. O matemático não precisou de mais de uma quinzena de dias para resolver o enigma. Para isso, ele criou um modelo matemático que simulasse a cidade russa, que é o que hoje chamados de grafo.

Utilizaremos as definições de grafos para um estudo dos circulos de amizades que existem entre o Facebook. De acordo com a [2], os dados do Facebook foram coletados dos participantes da pesquisa usando este aplicativo do Facebook. O conjunto de dados inclui recursos de nó (perfis), círculos e redes de egonet. Utilizaremos as arestas como as egonets combinadas. O dado pode ser verificado no link https://snap.stanford.edu/data/ego-Facebook.html ou no link do google drive que será disponibilizado por https://drive.google.com/drive/folders/1ewb7rgs8wuzmoKbYQY6jyXzKPUZrjxKm?usp=sharing.

A. Grafos

Os grafos de [3] são definidos por:

Definição 1. *Um grafo $G(V, A)$ é definido pelo par de conjuntos V e A, onde:*

- *V é conjunto não vazio cujos vértices ou nodos do grafo;*
- *A é conjunto de pares ordenados a=(v,w), v e w $\in$ V que são as arestas do grafo.*

Por exemplo, temos um o grafo G(V,A) dado por:

- $V = \{p|$ p é uma pessoa $\}$
- $A = \{(v, w)|\langle$ v é amigo de w $\rangle\}$

Esta definição representa toda uma família de grafos. Um exemplo de elemento desta família é dado por:

- V = {Maria, Pedro, Joana, Luiz}
- A = {(Maria, Pedro), (Pedro, Maria), (Joana, Maria), (Maria, Joana), (Pedro, Luiz), (Luiz, Pedro), (Joana, Pedro) , (Pedro, Joana)}

E o grafo é dado por:

Definição 2. *O grafo é dito ser um grafo orientado (ou digrafo), sendo que as conexões entre os vértices são chamadas de arcos.*

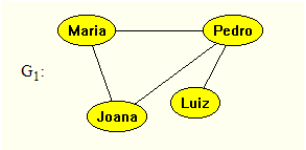

Fig. 2. Grafo que representa a relação exemplo (Fonte: [3]).

II. Definição dos Algoritmos Utilizados

Os algoritmos são definidos por [4], que nada mais é do que a documentação do Network X, um pacote do Python que lida com redes complexas. A própria documentação do pacote argumenta e dita que para citar o Network X, deve-se usar o artigo citado acima. A definição da classe de um grafo é dado pelo seguinte link da documentação: https://networkx.org/documentation/stable//_modules/networkx/classes/digraph.html#DiGraph

A. Algoritmo do Betweenness

A função Betweenness, definida por [4], é dada por:

Definição 3. *O Betweenness de um nó v é a soma da fração de todos os pares de caminhos mais curtos que passam por v, ou seja:*

$$c_B(x) = \sum_{s,t \in V} \frac{\sigma(s,t|v)}{\sigma(s,t)}$$

onde V é o conjunto de nós, $\sigma(s,t)$ *é o número dos caminhos mais curtos (s,t) e* $\sigma(s,t|v)$ *é o número daqueles caminhos que passam por algum nó v diferente de s , t. Se s = t,* $\sigma(s,t)$ *= 1, e se* $v \in s,t$*,* $\sigma(s,t|v)$ *= 0.*

O código pode ser dado por:

```
__author__ = """Aric Hagberg (hagberg@lanl.gov)"""

def betweenness_centrality(G, k=None, normalized=
    True, weight=None, endpoints=False, seed=None):
    betweenness = dict.fromkeys(G, 0.0)  # b[v]=0
    para v no grafo G
    if k is None:
        nodes = G
    else:
        random.seed(seed)
        nodes = random.sample(G.nodes(), k)

    for s in nodes:
        # Caminho mais curto de fonte única
        if weight is None:  # Usar o algorítmo de
    Bellman-Ford
            S, P, sigma =
    _single_source_shortest_path_basic(G, s)
        else:  # Usar o algoritmo de Dijkstra
            S, P, sigma =
    _single_source_dijkstra_path_basic(G, s, weight)

        # Acumulação
        if endpoints:
            betweenness = _accumulate_endpoints(
    betweenness, S, P, sigma, s)
        else:
            betweenness = _accumulate_basic(
    betweenness, S, P, sigma, s)

    # Reescalando
    betweenness = _rescale(betweenness, len(G),
    normalized=normalized,
                           directed=G.is_directed(),
     k=k)
    return betweenness
```

Onde a função rescale, accumulate endpoints, accumulate basic, single source dijkstra path basic (Algoritmo de caminho mínimo de Dijsktra) e single source shortest path basic(Algoritmo de Bellman-Ford) são funções complementares e estão definidas no drive já dado pelo link. Se há um peso nos grafos, será utilizado o algoritmo de Dijkstra, entretanto se não há, há uma busca em largura. Após isso, há uma acumulação que se dá pelo método do sigma, e também uma reescalada do grafo dentro dos números.

B. Algoritmo do Closeness

A função Closeness, definida por [4], é dada por:

Definição 4. *A Closeness de um nó u é o recíproco da distância média do caminho mais curto para u em todos os n-1 nós alcançáveis. Ou seja, é dado pela forma:*

$$C(u) = \frac{n-1}{\sum_{v=1}^{n-1} d(v,u)}$$

onde $d(v,u)$ *é a distância do menor caminho entre v e u, e n é o número de nós que podem alcançar u. Observe que a função de distância de proximidade calcula a distância de chegada a u para gráficos direcionados, portanto quanto maior o valor de closeness indica maior centralidade.*

O código pode ser dado por:

```
__author__ = "\n".join(['Aric Hagberg <aric.
    hagberg@gmail.com>',
                        'Pieter Swart (swart@lanl.
    gov)',
                        'Sasha Gutfraind (
    ag362@cornell.edu)'])

def closeness_centrality(G, u=None, distance=None,
    normalized=True):
    if distance is not None:

        #Usar o Algoritmo de Dijkstra para atributos
     específicos com peso
        path_length = functools.partial(nx.
    single_source_dijkstra_path_length,
                                        weight=
    distance)
    else:
        #Usar o Algoritmo de Bellman-Ford para
    atributos sem peso
        path_length = nx.
    single_source_shortest_path_length

    if u is None:
        nodes = G.nodes()
    else:
        nodes = [u]
    closeness_centrality = {}
    for n in nodes:
        sp = path_length(G,n)
        totsp = sum(sp.values())
        if totsp > 0.0 and len(G) > 1:
            closeness_centrality[n] = (len(sp)-1.0)
    / totsp

```

```
            # normalizar o numero de nós-1 na parte conectada
            if normalized:
                s = (len(sp)-1.0) / ( len(G) - 1 )
                closeness_centrality[n] *= s
        else:
            closeness_centrality[n] = 0.0
    if u is not None:
        return closeness_centrality[u]
    else:
        return closeness_centrality
```

No código, temos que se a distância for nula, os pesos serão dados pelo algoritmo de Bellman-Ford para busca em largura, mas se ela tiver pesos, ela é dada pelo algoritmo de Dijsktra. Após o calculo do closeness, há uma normalização dos dados, e um print do mesmo semelhantemente ao Betweenness.

III. Resultados

Primeiramente, antes da criação do grafo, criaremos um grafo aleatório para teste do plot dos grafos. Portanto, temos:

```
G = nx.DiGraph()
G.add_edge('A', 'B', weight=4)
G.add_edge('B', 'D', weight=2)
G.add_edge('A', 'C', weight=3)
G.add_edge('C', 'D', weight=4)

nx.draw(G, with_labels=True, font_weight='bold')
plt.savefig("grafo.png")
```

E a figura é dada por:

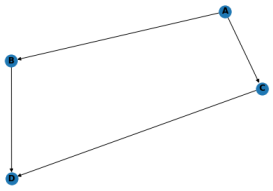

Fig. 3. Grafo que representa o grafo exemplo acima (Fonte: Autor).

Agora inserindo os grafos dentro da estrutura de dados, temos:

```
import random

vertice1=[]
vertice2=[]
weight=[]

with open('facebook_combined.txt', 'r') as f:
  for line in f.readlines():
      vet = line.split(' ')
      vertice1.append(int(float(vet[0])))
      vertice2.append(int(float(vet[1])))

#print(vertice1)
#print(vertice2)

for i in range(len(vertice1)):
  weight.append(random.randint(0,10))

#print(weight)

#A estrutura de grafo pelo NetworkX é dado pelo link
# Link: https://networkx.org/documentation/networkx-1.10/_modules/networkx/classes/digraph.html#DiGraph
```

```

Grafo = nx.DiGraph()

for i in range(len(vertice1)):
  Grafo.add_edge(vertice1[i], vertice2[i], weight=weight[i])

vertices=[i for i in range(4038)]
```

Note que os pesos são dados de maneira aleatória pela função random, pois os dados não revelam os pesos, portanto a geração de pesos aleatórios se dá de maneira interessante. O print do grafo da base de dados se deu de maneira esquisita, pois como a base de dados era enorme, ele ficou basicamente uma nuvem negra de arestas e vértices:

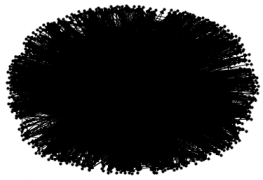

Fig. 4. Grafo que representa o grafo da instância (Fonte: Autor).

O grafo acima é dado por 4038 vértices e 88234 arestas. Aplicando o método do Closeness nos vértices, teremos:

```
import operator

close = closeness_centrality(Grafo)

close_list = sorted(close.items(), key=operator.itemgetter(1))

#print(close_list)
```

Que retorna uma lista dada em ordem crescente pelo closeness onde a primeira entrada é o vértice e a segunda entrada é o closeness.

Aplicando o método do Betweenness no grafo, teremos:

```
bet = betweenness_centrality(Grafo)

bet_list = sorted(bet.items(), key=operator.itemgetter(1))

#print(bet_list)
```

Que também retorna uma lista dada em ordem crescente pelo betweenness onde a primeira entrada é o vértice e a segunda entrada é o betweenness.

IV. Conclusão

O estudo foi devidamente aplicado sobre as listas houveram resultados promissores com relação às teorias de centralidade sobre os círculos sociais principais de acordo com o vetor randômico de relações sociais. Entretanto, houve uma limitação com relação ao plot do grafo, pois o grafo possuia 4038 vértices e 88234 arestas, ou seja, um grande volume de vértices, impossibilitanto o plot. Entretanto, para redes como o grafo com apenas 4 vértices e 4 arestas, houve uma clara visualização do sistema. Além de todas as considerações, o estudo sobre como as redes criadas por circulos sociais podem

mostrar como as redes sociais são complexas e possuem uma relação parecida com um mapa, ou seja, pode-se garantir que as relações sociais por gostos geram um certa bolha que se mostra por meio da obtenção do mapa social, reduzindo as relações sociais à uma conexão de nível extremamente complexo.

REFERENCES

[1] Lauro Chagas e Sá Sandra Aparecida Fraga da Silva. Ensinando grafos a partir de abordagem histórico-investigativa. *Biblioteca Nilo Peçanha do Instituto Federal do Espírito Santo*, page 54, 2006.

[2] Jure Leskovec. Social circles: Facebook. Disponível em: https://snap.stanford.edu/data/ego-Facebook.html. Accessed: 2020-02-22.

[3] Antonio C. Mariani. Teoria dos grafos - livro eletrônico: Definições básicas. Disponível em: https://www.inf.ufsc.br/grafos/definicoes/definicao.html. Accessed: 2020-02-26.

[4] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.