Your Paper

Lucas de Oliveira Silva, 200022857

25 de fevereiro de 2022

Resumo

Trabalho de Teoria e Aplicação de Grafos utilizando o algoritmo de Bron Kerbosch para encontrar todos os cliques maximais.

1 Introduction

Dada todas as arestas não direcionadas de um grafo de uma rede social de relações em uma comunidade de 62 golfinhos que pode ser encontrado em [RA15] vamos montar a lista de adjacência desse grafo e em seguida fazer o que for pedido com o mesmo grafo.

2 Procedimentos

nesse trabalho iremos ter 3 etapas: Implementaremos duas 2 formas do algoritmo Bron-Kerbosch: uma com pivotamento, outra sem pivotamento; vamos encontrar e imprimir na tela (duas vezes, uma para cada implementação do Bron-Kerbosch) todos os cliques maximais (indicando o número de vértices e quais); E iremos encontrar o Coeficiente médio de Aglomeração do Grafo.

2.1 Bron-Kerbosh

Antes de implementar o algoritmo de Bron-Kerbosh, criei uma função para transformar as arestas dadas em uma lista de adjacência do grafo da seguinte maneira:

```
    #Cria a lista de adjacencia dos 62 nós baseada nas 159 arestas dadas
def ListaAdjacencia():
    grafo = {}
    #N = input('digite a quantidade de arestas que dara entrada:')
    #for i in range(N):

    #Apesar de ter feito para qualquer tamanho de grafo, ja
    #implementei o valor de arestas no for abaixo(159).
    for i in range(159):
        aresta = [int(x) for x in input().split()]
        grafo[aresta[0]] = grafo.get(aresta[0], []) + [aresta[1]]
        grafo[aresta[1]] = grafo.get(aresta[1], []) + [aresta[0]]
    return grafo
```

Utilizei a fonte com os códigos de Bron-Kerbosh em python, uma versão com pivotamento e outra sem pivotamento encontrados no seguinte link [Zot21] feita pelo usuario Mykola Zotko.

Na versão sem pivotamento teremos a seguinte lógica:

```
    #algorithm BronKerbosch1(R, P, #X) is
    #if P and X are both empty then:
    #    report R as a maximal clique
    #for each vertex v in P do
    #    BronKerbosch1(R  {v}, P  N(v), X  N(v))
    #    P := P \ {v}
    #    X := X  {v}
```

Com isso temos a seguinte função:

```
#Codigo fonte retirado de: https://stackoverflow.com/questions/13904636/implementing-bron-kerbosc
def BronKerbosch1(P, R=None, X=None):
    P = set(P)
    R = set() if R is None else R
    X = set() if X is None else X
    if not P and not X:
        yield sorted(R)
    while P:
        v = P.pop()
        yield from BronKerbosch1(P=P.intersection(grafo[v]), R=R.union([v]), X=X.intersection(gra
        X.add(v)
```

Também implementei o comando sorted() nos dois algoritmos para organizar os cliques em uma certa escala.

Na versão com pivotamento temos a seguinte lógica:

```
#algorithm BronKerbosch2(R, P, X) is
#    if P and X are both empty then
#        report R as a maximal clique
#    choose a pivot vertex u in P  X
#    for each vertex v in P \ N(u):
#        BronKerbosch2(R  {v}, P  N(v), X  N(v))
#        P := P \ {v}
#        X := X  {v}*
```

Com isso temos a seguinte função:

```
#Codigo fonte retirado de: https://stackoverflow.com/questions/13904636/implementing-bron-kerbosc
def BronKerbosch2(P, R=None, X=None):
P = set(P)
R = set() if R is None else R
X = set() if X is None else X
if not P and not X:
    yield sorted(R)
try:
    u = random.choice(list(P.union(X)))
    S = P.difference(grafo[u])
# if union of P and X is empty
except IndexError:
    S = P
for v in S:
    yield from BronKerbosch2(
        P=P.intersection(grafo[v]), R=R.union([v]), X=X.intersection(grafo[v]))
    P.remove(v)
    X.add(v)
```

2.2 Cliques Maximais

Tendo os cliques maximais encontrados nos algoritmos da Seção 2.1 para imprimir na tela (duas vezes, uma para cada implementação do Bron-Kerbosch) todos os cliques maximais (indicando o número de vértices e quais) criei a seguinte função que recebe a lista de cliques e percorre ela por completo imprimindo o tamanho do clique pela quantidade de nós, seguido pelos nós que compõe cada clique:

```
    #2° parte: Função para mostrar os Cliques Maximais
def ImprimeCliques(cliques):
    print('Cliques Maximais:')
```

```
for i in cliques:
    print(f'({len(i)})nós:{i}')
```

Chamando a função teremos o seguinte retorno no algoritmo sem pivotamento:

```
Cliques Maximais:
(3)nós:[1, 16, 41]
(4)nós:[1, 11, 43, 48]
(3)nós:[1, 15, 41]
(2)nós:[2, 37]
(3)nós:[2, 42, 55]
(3)nós:[2, 18, 28]
(3)nós:[2, 20, 55]
(3)nós:[2, 27, 28]
(2)nós:[2, 29]
(3)nós:[3, 11, 43]
(2)nós:[3, 45]
(2)nós:[3, 62]
(3)nós:[4, 9, 60]
(2)nós:[4, 15]
(2)nós:[5, 52]
(2)nós:[6, 57]
(4)nós:[6, 10, 14, 58]
(5)nós:[7, 10, 14, 18, 58]
(4)nós:[7, 14, 55, 58]
(2)nós:[7, 57]
(3)nós:[8, 20, 55]
(3)nós:[8, 20, 31]
(2)nós:[8, 41]
(2)nós:[8, 28]
(3)nós:[9, 21, 29]
(3)nós:[9, 38, 46]
(3)nós:[9, 46, 60]
(3)nós:[10, 14, 33]
(4)nós:[10, 14, 42, 58]
(2)nós:[11, 30]
(2)nós:[12, 52]
(2)nós:[13, 34]
(4)nós:[14, 42, 55, 58]
(4)nós:[15, 17, 34, 51]
(4)nós:[15, 17, 34, 38]
(4)nós:[15, 17, 34, 39]
(4)nós:[15, 34, 35, 38]
(4)nós:[15, 34, 38, 41]
(4)nós:[15, 34, 38, 44]
(4)nós:[15, 34, 39, 44]
(3)nós:[15, 39, 53]
(3)nós:[15, 41, 53]
(2)nós:[15, 25]
(4)nós:[16, 19, 25, 46]
(2)nós:[16, 56]
(3)nós:[16, 46, 60]
(3)nós:[17, 21, 51]
(3)nós:[17, 21, 39]
(2)nós:[18, 32]
(3)nós:[18, 26, 28]
```

```
(2)nós:[18, 23]
(5)nós:[19, 25, 30, 46, 52]
(5)nós:[19, 22, 30, 46, 52]
(2)nós:[19, 21]
(3)nós:[21, 29, 48]
(2)nós:[21, 37]
(3)nós:[21, 39, 45]
(3)nós:[22, 34, 38]
(3)nós:[22, 38, 46]
(3)nós:[24, 46, 52]
(2)nós:[24, 37]
(3)nós:[26, 27, 28]
(3)nós:[29, 31, 48]
(2)nós:[30, 36]
(2)nós:[30, 53]
(2)nós:[30, 44]
(3)nós:[31, 43, 48]
(2)nós:[33, 61]
(2)nós:[35, 50]
(2)nós:[35, 45]
(2)nós:[37, 40]
(3)nós:[37, 38, 41]
(2)nós:[37, 60]
(2)nós:[38, 62]
(2)nós:[39, 59]
(2)nós:[40, 58]
(2)nós:[43, 51]
(2)nós:[44, 54]
(2)nós:[44, 47]
(3)nós:[46, 51, 52]
(2)nós:[47, 50]
(2)nós:[49, 58]
(2)nós:[52, 56]
(2)nós:[54, 62]
```

Chamando a mesma função teremos o seguinte retorno no algoritmo com pivotamento:

```
Cliques Maximais:
(3)nós:[1, 16, 41]
(4)nós:[1, 11, 43, 48]
(3)nós:[1, 15, 41]
(2)nós:[2, 37]
(3)nós:[2, 42, 55]
(3)nós:[2, 18, 28]
(3)nós:[2, 20, 55]
(3)nós:[2, 27, 28]
(2)nós:[2, 29]
(3)nós:[3, 11, 43]
(2)nós:[3, 45]
(2)nós:[3, 62]
(3)nós:[4, 9, 60]
(2)nós:[4, 15]
(2)nós:[5, 52]
(2)nós:[6, 57]
(4)nós:[6, 10, 14, 58]
(2)nós:[7, 57]
(4)nós:[7, 14, 55, 58]
```

(5)nós:[7, 10, 14, 18, 58]
(3)nós:[8, 20, 55]
(3)nós:[8, 20, 31]
(2)nós:[8, 41]
(2)nós:[8, 28]
(3)nós:[9, 21, 29]
(3)nós:[9, 38, 46]
(3)nós:[9, 46, 60]
(3)nós:[10, 14, 33]
(4)nós:[10, 14, 42, 58]
(2)nós:[11, 30]
(2)nós:[12, 52]
(2)nós:[13, 34]
(4)nós:[14, 42, 55, 58]
(4)nós:[15, 34, 35, 38]
(3)nós:[15, 41, 53]
(4)nós:[15, 34, 38, 41]
(4)nós:[15, 34, 38, 44]
(4)nós:[15, 34, 39, 44]
(4)nós:[15, 17, 34, 51]
(4)nós:[15, 17, 34, 38]
(4)nós:[15, 17, 34, 39]
(3)nós:[15, 39, 53]
(2)nós:[15, 25]
(4)nós:[16, 19, 25, 46]
(2)nós:[16, 56]
(3)nós:[16, 46, 60]
(3)nós:[17, 21, 51]
(3)nós:[17, 21, 39]
(2)nós:[18, 32]
(3)nós:[18, 26, 28]
(2)nós:[18, 23]
(5)nós:[19, 22, 30, 46, 52]
(5)nós:[19, 25, 30, 46, 52]
(2)nós:[19, 21]
(3)nós:[21, 29, 48]
(2)nós:[21, 37]
(3)nós:[21, 39, 45]
(3)nós:[22, 34, 38]
(3)nós:[22, 38, 46]
(3)nós:[24, 46, 52]
(2)nós:[24, 37]
(3)nós:[26, 27, 28]
(3)nós:[29, 31, 48]
(2)nós:[30, 36]
(2)nós:[30, 53]
(2)nós:[30, 44]
(3)nós:[31, 43, 48]
(2)nós:[33, 61]
(2)nós:[37, 40]
(3)nós:[37, 38, 41]
(2)nós:[37, 60]
(2)nós:[38, 62]
(2)nós:[39, 59]
(2)nós:[40, 58]
(2)nós:[43, 51]

```
(2)nós:[44, 54]
(2)nós:[44, 47]
(2)nós:[35, 45]
(3)nós:[46, 51, 52]
(2)nós:[49, 58]
(2)nós:[35, 50]
(2)nós:[47, 50]
(2)nós:[52, 56]
(2)nós:[54, 62]
```

2.3 O Coeficiente médio de Aglomeração do Grafo

Para encontrar o coeficiente médio de aglomeração do grafo utilizei a seguinte formula: coeficiente de aglomeração de cada nó = Quantidade de triangulos encontrados / Triangulos possiveis(n*(n-1)/2), Onde a variável n é a quantidade de vizinhos de cada nó.

Dividindo tudo pela quantidade de nós teremos O Coeficiente médio de Aglomeração do Grafo. [dlb22]

Criei a seguinte função para calcular:

```python
#3° parte descobrir todos os coeficientes de aglomeração dos nós
#e em seguida fazer a media do grafo
def Aglomeração(grafo, cliques):
    #todas os coeficientes de aglomeração dos nós
    Aglomeração_Geral = []

    #percorre o grafo
    for i in grafo:
        triangulos = []
        for l in grafo[i]:
            for k in grafo[i]:
                if l != k and [l, k] not in triangulos:
                    for clique in cliques:
                        if i in clique and l in clique and k in clique:
                            #compara o no(i) e 2 vizinhos distintos(l e k) para
                            #conferir se existe triangulo nos cliques
                            triangulos.append([l, k])
                            triangulos.append([k, l])
                            break
        #Equação para descobrir o coeficente de aglomeração do nó(i)
        if len(grafo[i]) > 1:
            #Quantidade de triangulos encontrados/triangulos possiveis(n*(n-1)/2)
            Aglomeração_Geral.append((len(triangulos)/2)/(len(grafo[i])*(len(grafo[i])-1)/2))
        else:
            #caso o nó so tenha uma aresta
            Aglomeração_Geral.append(0.0)
    #soma da aglomeração de cada nó/quantidades de nó do grafo
    return f'\nCoeficiente médio de Aglomeração do Grafo = {sum(Aglomeração_Geral)/len(grafo)}'
```

2.4 Código final

Juntando todas as funções e suas respectivas chamadas teremos os códigos completos sem e com pivotamento, segue os dois códigos.

Bron-Kerbosch algoritmo sem pivotamento:

```python
#Cria a lista de adjacencia dos 62 nós baseada nas 159 arestas dadas
def ListaAdjacencia():
    grafo = {}
    #N = input('digite a quantidade de arestas que dara entrada:')
```

```
    #for i in range(N):

    #Apesar de ter feito para qualquer tamanho de grafo, ja
    #implementei o valor de arestas no for abaixo(159).
    for i in range(159):
        aresta = [int(x) for x in input().split()]
        grafo[aresta[0]] = grafo.get(aresta[0], []) + [aresta[1]]
        grafo[aresta[1]] = grafo.get(aresta[1], []) + [aresta[0]]
    return grafo

#algorithm BronKerbosch1(R, P, X) is
#    if P and X are both empty then:
#        report R as a maximal clique
#    for each vertex v in P do
#        BronKerbosch1(R  {v}, P  N(v), X  N(v))
#        P := P \ {v}
#        X := X  {v}
#Codigo fonte da função a seguir retirado
#de: https://stackoverflow.com/questions/13904636/implementing-bron-kerbosch-algorithm-in-python
def BronKerbosch1(P, R=None, X=None):
    P = set(P)
    R = set() if R is None else R
    X = set() if X is None else X
    if not P and not X:
        yield sorted(R)
    while P:
        v = P.pop()
        yield from BronKerbosch1(
            P=P.intersection(grafo[v]), R=R.union([v]), X=X.intersection(grafo[v]))
        X.add(v)

#2° parte: Função para mostrar os Cliques Maximais
def ImprimeCliques(cliques):
    print('Cliques Maximais:')

    for i in cliques:
        print(f'({len(i)})nós:{i}')

#3° parte descobrir todos os coeficientes de aglomeração dos nós
#e em seguida fazer a media do grafo
def Aglomeração(grafo, cliques):
    #todas os coeficientes de aglomeração dos nós
    Aglomeração_Geral = []

    #percorre o grafo
    for i in grafo:
        triangulos = []
        for l in grafo[i]:
            for k in grafo[i]:
                if l != k and [l, k] not in triangulos:
                    for clique in cliques:
                        if i in clique and l in clique and k in clique:
                            #compara o no(i) e 2 vizinhos distintos(l e k) para
                            #conferir se existe triangulo nos cliques
                            triangulos.append([l, k])
                            triangulos.append([k, l])
```

```
                            break
        #Equação para descobrir o coeficente de aglomeração do nó (i)
        if len(grafo[i]) > 1:
            #Quantidade de triangulos encontrados/triangulos possiveis(n*(n-1)/2)
            Aglomeração_Geral.append((len(triangulos)/2)/(len(grafo[i])*(len(grafo[i])-1)/2))
        else:
            #caso o nó so tenha uma aresta
            Aglomeração_Geral.append(0.0)
    #soma da aglomeração de cada nó/quantidades de nó do grafo
    return f'\nCoeficiente médio de Aglomeração do Grafo = {sum(Aglomeração_Geral)/len(grafo)}'

#Construir a lista de adjacencia
grafo = ListaAdjacencia()

#Aplicando a funçao BronKerbosch1 para conseguir a lista de cliques maximais
cliques = list(BronKerbosch1(grafo.keys()))

#Imprime todos os cliques encontrados
ImprimeCliques(cliques)

#soma da aglomeração de cada nó/quantidades de nó do grafo
print(Aglomeração(grafo, cliques))
#
#
#
#
#
#
```

Bron-Kerbosch algoritmo com pivotamento:

```
import random

#Cria a lista de adjacencia dos 62 nós baseada nas 159 arestas dadas
def ListaAdjacencia():
    grafo = {}
    #N = input('digite a quantidade de arestas que dara entrada:')
    #for i in range(N):

    #Apesar de ter feito para qualquer tamanho de grafo, ja
    #implementei o valor de arestas no for abaixo(159).
    for i in range(159):
        aresta = [int(x) for x in input().split()]
        grafo[aresta[0]] = grafo.get(aresta[0], []) + [aresta[1]]
        grafo[aresta[1]] = grafo.get(aresta[1], []) + [aresta[0]]
    return grafo

#algorithm BronKerbosch2(R, P, X) is
#    if P and X are both empty then
#        report R as a maximal clique
#    choose a pivot vertex u in P  X
#    for each vertex v in P \ N(u):
#        BronKerbosch2(R  {v}, P  N(v), X  N(v))
#        P := P \ {v}
#        X := X  {v}*
#Codigo fonte da função a seguir retirado
#de: https://stackoverflow.com/questions/13904636/implementing-bron-kerbosch-algorithm-in-python
def BronKerbosch2(P, R=None, X=None):
```

```
    P = set(P)
    R = set() if R is None else R
    X = set() if X is None else X
    if not P and not X:
        yield sorted(R)
    try:
        u = random.choice(list(P.union(X)))
        S = P.difference(grafo[u])
    # if union of P and X is empty
    except IndexError:
        S = P
    for v in S:
        yield from BronKerbosch2(
            P=P.intersection(grafo[v]), R=R.union([v]), X=X.intersection(grafo[v]))
        P.remove(v)
        X.add(v)

#2° parte: Função para mostrar os Cliques Maximais
def ImprimeCliques(cliques):
    print('Cliques Maximais:')

    for i in cliques:
        print(f'({len(i)})nós:{i}')

#3° parte descobrir todos os coeficientes de aglomeração dos nós
#e em seguida fazer a media do grafo
def Aglomeração(grafo, cliques):
    #todas os coeficientes de aglomeração dos nós
    Aglomeração_Geral = []

    #percorre o grafo
    for i in grafo:
        triangulos = []
        for l in grafo[i]:
            for k in grafo[i]:
                if l != k and [l, k] not in triangulos:
                    for clique in cliques:
                        if i in clique and l in clique and k in clique:
                            #compara o no(i) e 2 vizinhos distintos(l e k) para
                            #conferir se existe triangulo nos cliques
                            triangulos.append([l, k])
                            triangulos.append([k, l])
                            break
        #Equação para descobrir o coeficente de aglomeração do nó (Grafo[i])
        if len(grafo[i]) > 1:
            #Quantidade de triangulos encontrados/triangulos possiveis(n*(n-1)/2)
            Aglomeração_Geral.append((len(triangulos)/2)/(len(grafo[i])*(len(grafo[i])-1)/2))
        else:
            #caso o nó so tenha uma aresta
            Aglomeração_Geral.append(0.0)
    #soma da aglomeração de cada nó/quantidades de nó do grafo
    return f'\nCoeficiente médio de Aglomeração do Grafo = {sum(Aglomeração_Geral)/len(grafo)}'

#Construir a lista de adjacencia
grafo = ListaAdjacencia()
```

```
#Aplicando a funçao BronKerbosch2 para conseguir a lista de cliques maximais
cliques = list(BronKerbosch2(grafo.keys()))

#Imprime todos os cliques encontrados
ImprimeCliques(cliques)

#soma da aglomeração de cada nó/quantidades de nó do grafo
print(Aglomeração(grafo, cliques))
```

Referências

[dlb22] CIC dlb, UnB. Teoria e aplicação de grafos, aula.5. https://aprender3.unb.br/mod/resource/view.php?id=665269, 2022.

[RA15] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[Zot21] Mykola Zotko. Implementing bron–kerbosch algorithm in python. https://stackoverflow.com/questions/13904636/implementing-bron-kerbosch-algorithm-in-python, 2021.