Tutorial Projeto 5
Alunos:

Rafael Bastos Saito - 726580
Renata Sarmet Smiderle Mendes - 726586
Rodrigo Pesse de Abreu - 726588

O Problema do Caixeiro Viajante

O Problema do Caixeiro Viajante (PCV) é um problema que tenta determinar a menor rota para percorrer uma série de cidades (visitando uma única vez cada uma delas), retornando à cidade de origem. Ele é um problema de otimização NP-difícil inspirado na necessidade dos vendedores em realizar entregas em diversos locais (as cidades) percorrendo o menor caminho possível, reduzindo o tempo necessário para a viagem e os possíveis custos com transporte e combustível.

Caminho Hamiltoniano

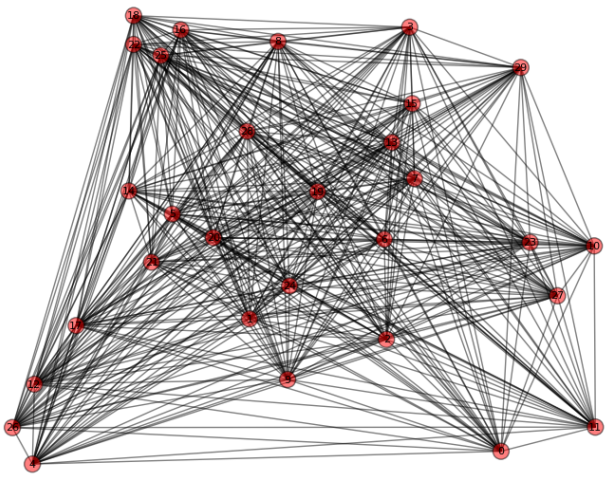
Um caminho hamiltoniano é um caminho que permite passar por todos os vértices de um grafo G, não repetindo nenhum, ou, seja, passar por todos uma e uma só vez por cada. Caso esse caminho seja possível descrever um ciclo, este é denominado ciclo hamiltoniano (ou circuito hamiltoniano) em G. E, um grafo que possua tal circuito é chamado de grafo hamiltoniano.

Problema

Desenvolver um programa que deve ler um grafo Hamiltoniano ponderado a partir de um arquivo qualquer e através de um algoritmo visto em sala (2-otimal ou Twice-Around) obter 10 soluções diferentes para o problema do caixeiro-viajante.

Metodologia

Para obter soluções distintas para o problema há algumas heurísticas comumente adotadas na prática: utilizar diferentes inicializações, ou seja, soluções iniciais. Elas podem ser geradas simplesmente aleatoriamente (selecionando vértices quaisquer) ou utilizando alguma heurística, como por exemplo a escolha do vizinho mais próximo por exemplo. Dessa forma, escolhe-se aleatoriamente apenas o primeiro vértice do ciclo (v0) e depois sempre é escolhido como próximo elemento da sequência o vizinho mais próximo do vértice atual, até que o ciclo Hamiltoniano seja formado (não sobre mais vértices).



Resolução

Para a implementação foi utilizada a linguagem Python, a biblioteca NetworkX e Matplotlib.

T5grafos.py

```python
import numpy as np
import networkx as nx
import matplotlib.pyplot as plt
from heapq import heappop, heappush
pop = heappop
push = heappush

def twice_around(G, origin = 0):
    H = nx.minimum_spanning_tree(G)                 # Encontra a MST de G
    H = nx.MultiGraph(H)

    Haux = H.copy()                                 # Fazendo uma cópia de H para utilizar no loop

    for u,v in Haux.edges():                        # Duplica as arestas da MST
        H.add_edge(u,v)

    euleraux = list(nx.eulerian_circuit(H, origin))     # Encontra circuito Euleriano
    I = nx.Graph()
    aux = []
    for u,v in euleraux:                            # Adiciona em aux todos os vertices do circuito Euleriano
        aux.append(u)
        aux.append(v)
    h = []
    for i in aux:                                   # Adiciona os h os vértices de aux sem repeticoes
        if (i not in h):
            h.append(i)
    h.append(origin)                                # Adiciona a origem
    for i in range (30):                            # Coloca em
        I.add_edge(h[i],h[i+1])
        I[h[i]][h[i+1]]['weight'] = G[h[i]][h[i+1]]['weight']
    return I                                        # Retorna I

def calcula_peso(T):
    peso = 0
    pesos = nx.get_edge_attributes(T, 'weight')
    for v in T.edges():
        peso += pesos[v]
    return peso
```

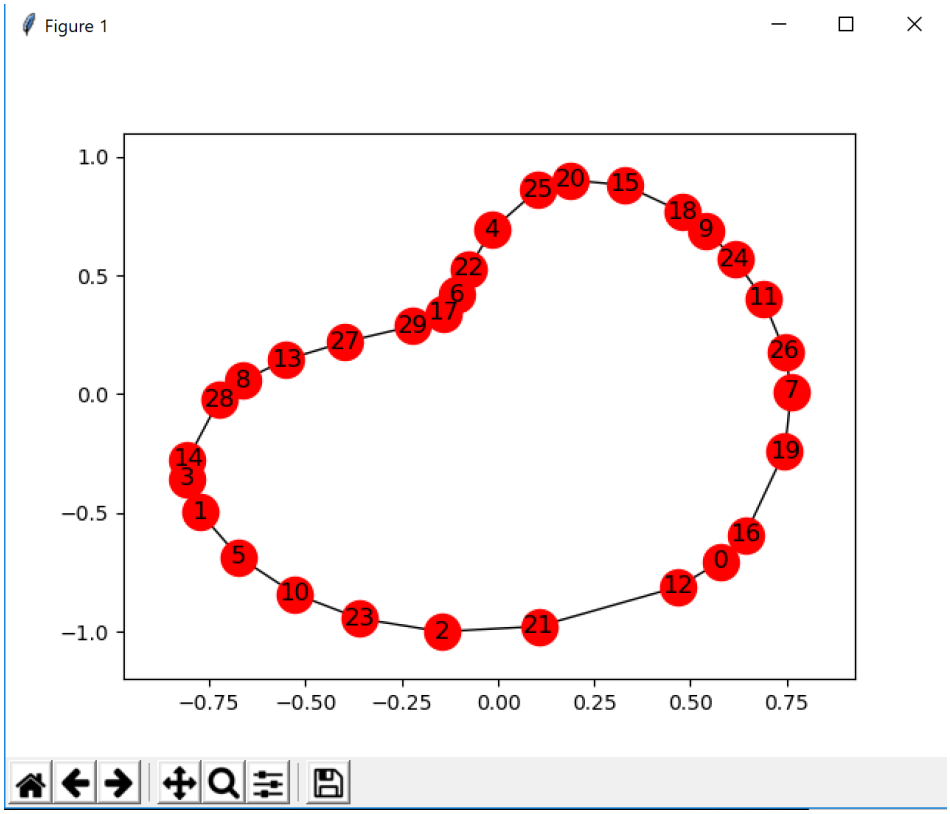
Método Twice Around e para calcular peso

T5grafos.py

```
40  # Main
41  A = np.loadtxt('ha30_dist.txt')
42  G = nx.from_numpy_matrix(A)
43  min_pesos = []
44  max_pesos = []
45  for i in range(30):
46      H = twice_around(G, i)
47      weight = calcula_peso(H)
48      push(min_pesos, (weight, i))
49      push(max_pesos, (-weight, i))
50
51  for i in range(3):
52      peso, inicial = pop(min_pesos)
53      print("Iniciando em ", inicial, ":", peso)
54
55  for i in range(3):
56      peso, inicial = pop(max_pesos)
57      print("Iniciando em ", inicial, ":", -peso)
58
59  #Desenha e exibe o grafo
60  pos = nx.spring_layout(H, k = 0.35, iterations=100)
61  nx.draw_networkx(H, pos)
62  plt.show()
```

Main chamando o Twice Around e Calcula Peso

Rodando o algoritmo, tem-se como resultado o grafo:



Questionamentos

Liste as 3 melhores soluções e as 3 piores obtidas. Qual a diferença de custo entre a melhor e a pior? Discuta como a diferença pode ser significativa.

3 Piores Soluções

Iniciando no vértice 18: 682.0
Iniciando no vértice 15: 682.0
Iniciando no vértice 7: 676.0

3 Melhores Soluções

Iniciando no vértice 29: 575.0
Iniciando no vértice 27: 577.0
Iniciando no vértice 13: 593.0

Escolher um vértice inicial é uma parte muito importante que pode gerar resultados significativamente diferentes posteriormente. Nesse caso, tem-se uma diferença de 10.000km entre o melhor e o pior caminho.