

Problema do Caixeiro Viajante

Seminário – Algoritmos em Grafos

Disciplina: Projeto e Análise de Algoritmos

Professor: Leonardo Nogueira Matos

Integrantes:

- Gustavo Dias
- Raquel Martins

Introdução

O Problema do Caixeiro Viajante (Travelling Salesman Problem - TSP) é um dos mais clássicos da ciência da computação e da matemática combinatória.

Consiste em encontrar o menor caminho possível que visite todas as cidades exatamente uma vez e retorne ao ponto de partida.

Este problema é NP-difícil, ou seja, não existe algoritmo conhecido que o resolva em tempo polinomial para todos os casos.

Problema em Grafos: Caixeiro Viajante (TSP) aplicado à Coleta de Lixo

O problema da coleta de lixo pode ser modelado como um grafo completo ponderado:

- Cada vértice representa um ponto de coleta (bairros, ruas, depósitos).
- Cada aresta representa a distância entre dois pontos.

O objetivo é encontrar o menor percurso que parte do depósito, passa por todos os pontos de coleta exatamente uma vez e retorna ao depósito.

Esse é o mesmo desafio do Problema do Caixeiro Viajante (TSP), que é NP-difícil — ou seja, para instâncias grandes, não há algoritmo eficiente conhecido que sempre encontra a solução ótima em tempo polinomial.

Formulação do Problema



Redução de custos (menos combustível e tempo de percurso).



Menor impacto ambiental (menos emissão de CO₂).



Maior eficiência na operação.

Desenvolvimento



No código foi utilizado a força bruta (testa todas as permutações possíveis de rotas). Para 5 pontos de coleta, isso ainda é viável. Ele garante a solução ótima, porque avalia todas as possibilidades e escolhe a menor distância.

Para instâncias grandes, a complexidade cresce muito rápido ($O(n!)$), mas para fins didáticos funciona bem.

```
import itertools

pontos = ['Deposito', 'Centro', 'Atalaia', 'Jardins', 'Santos Dumont']

distancias = {
    ('Deposito', 'Centro'): 5,
    ('Deposito', 'Atalaia'): 10,
    ('Deposito', 'Jardins'): 8,
    ('Deposito', 'Santos Dumont'): 12,
    ('Centro', 'Atalaia'): 6,
    ('Centro', 'Jardins'): 4,
    ('Centro', 'Santos Dumont'): 9,
    ('Atalaia', 'Jardins'): 5,
    ('Atalaia', 'Santos Dumont'): 7,
    ('Jardins', 'Santos Dumont'): 6
}
```



```
import itertools

pontos = ['Depósito', 'Centro', 'Atalaia', 'Jardins', 'Santos Dumont']

distancias = {
    ('Depósito', 'Centro'): 5,
    ('Depósito', 'Atalaia'): 10,
    ('Depósito', 'Jardins'): 8,
    ('Depósito', 'Santos Dumont'): 12,
    ('Centro', 'Atalaia'): 6,
    ('Centro', 'Jardins'): 4,
    ('Centro', 'Santos Dumont'): 9,
    ('Atalaia', 'Jardins'): 5,
    ('Atalaia', 'Santos Dumont'): 7,
    ('Jardins', 'Santos Dumont'): 6
}
```



```
def distancia_total(rota):
    custo = 0
    for i in range(len(rota)):
        cidade1 = rota[i]
        cidade2 = rota[(i+1) % len(rota)]
        if (cidade1, cidade2) in distancias:
            custo += distancias[(cidade1, cidade2)]
        else:
            custo += distancias[(cidade2, cidade1)]
    return custo

def tsp_forca_bruta(pontos):
    permutacoes = itertools.permutations(pontos[1:])
    melhor_rota = None
    menor_distancia = float('inf')

    for rota in permutacoes:
        rota_completa = ('Depósito',) + rota
        custo = distancia_total(rota_completa)
        if custo < menor_distancia:
            menor_distancia = custo
            melhor_rota = rota_completa
    return melhor_rota, menor_distancia

rota, custo = tsp_forca_bruta(pontos)

print("Melhor rota de coleta:", rota)
print("Distância total:", custo, "km")
```

```
import networkx as nx
import matplotlib.pyplot as plt

pontos = ['Deposito', 'Centro', 'Atalaia', 'Jardins', 'Santos Dumont']

distancias = {
    ('Deposito', 'Centro'): 5,
    ('Deposito', 'Atalaia'): 10,
    ('Deposito', 'Jardins'): 8,
    ('Deposito', 'Santos Dumont'): 12,
    ('Centro', 'Atalaia'): 6,
    ('Centro', 'Jardins'): 4,
    ('Centro', 'Santos Dumont'): 9,
    ('Atalaia', 'Jardins'): 5,
    ('Atalaia', 'Santos Dumont'): 7,
    ('Jardins', 'Santos Dumont'): 6
}
```



```
import networkx as nx
import matplotlib.pyplot as plt

pontos = ['Depósito', 'Centro', 'Atalaia', 'Jardins', 'Santos Dumont']

distancias = {
    ('Depósito', 'Centro'): 5,
    ('Depósito', 'Atalaia'): 10,
    ('Depósito', 'Jardins'): 8,
    ('Depósito', 'Santos Dumont'): 12,
    ('Centro', 'Atalaia'): 6,
    ('Centro', 'Jardins'): 4,
    ('Centro', 'Santos Dumont'): 9,
    ('Atalaia', 'Jardins'): 5,
    ('Atalaia', 'Santos Dumont'): 7,
    ('Jardins', 'Santos Dumont'): 6
}
```



```
def obter_distancia(c1, c2):
    if (c1, c2) in distancias:
        return distancias[(c1, c2)]
    else:
        return distancias[(c2, c1)]

rota = ['Depósito', 'Centro', 'Jardins', 'Atalaia', 'Santos Dumont', 'Depósito']

G = nx.Graph()
for (c1, c2), d in distancias.items():
    G.add_edge(c1, c2, weight=d)

pos = {
    'Depósito': (0, 0),
    'Centro': (1, 1.5),
    'Jardins': (2.5, 1.2),
    'Atalaia': (3.5, 0),
    'Santos Dumont': (2, -1)
}
```

```
def obter_distancia(c1, c2):
    if (c1, c2) in distancias:
        return distancias[(c1, c2)]
    else:
        return distancias[(c2, c1)]


rota = ['Deposito', 'Centro', 'Jardins', 'Atalaia', 'Santos Dumont', 'Deposito']


G = nx.Graph()
for (c1, c2), d in distancias.items():
    G.add_edge(c1, c2, weight=d)


pos = {
    'Deposito': (0, 0),
    'Centro': (1, 1.5),
    'Jardins': (2.5, 1.2),
    'Atalaia': (3.5, 0),
    'Santos Dumont': (2, -1)
}
```



```
plt.figure(figsize=(8,6))
nx.draw(G, pos, with_labels=True, node_size=1800, node_color="#14b8a6", font_size=10, font_color="white")


nx.draw_networkx_edge_labels(G, pos, edge_labels=nx.get_edge_attributes(G, 'weight'))


edge_list = [(rota[i], rota[i+1]) for i in range(len(rota)-1)]
nx.draw_networkx_edges(G, pos, edgelist=edge_list, width=3, edge_color="red")


plt.title("Rota de Coleta de Lixo em Aracaju (TSP)", fontsize=14)
plt.show()
```

Exemplo de Código (Python - Colab)



<https://colab.research.google.com/drive/15O9js2rtSqZSYgWfxS3q2DoX8rqmlS9i?usp=sharing>

Conclusão

O estudo mostrou como o Problema do Caixeiro Viajante (TSP) pode ser aplicado de forma prática ao planejamento da rota de coleta de lixo em Aracaju. A modelagem do problema em forma de grafo possibilitou representar os pontos de coleta e as distâncias entre eles, permitindo que algoritmos de busca encontrassem a melhor rota.

Com a utilização do algoritmo de força bruta, foi possível determinar a rota ótima, garantindo o menor percurso para visitar todos os pontos e retornar ao depósito. Apesar da limitação de escalabilidade dessa abordagem, ela cumpre o objetivo em instâncias pequenas e serve como base para compreender a complexidade do problema.

Conclusão

Além disso, a aplicação desse tipo de solução traz benefícios práticos e reais, como:

- Redução de custos operacionais com combustível e manutenção de veículos.
- Maior eficiência na execução da coleta, otimizando o tempo de trabalho.
- Impacto ambiental positivo, com menor emissão de poluentes.

Portanto, a abordagem apresentada reforça a importância dos grafos e da teoria da computação na solução de problemas urbanos, mostrando como a matemática e a ciência da computação podem gerar ganhos concretos para a sociedade.