Trabalho Computacional II - Problema do Caixeiro Viajante

Luiz Filipe Moraes, 112229
Lucas Eduardo Nogueira Gonçalves, 122055

14 de janeiro de 2021

1 Introdução

Um dos problemas mais estudados envolvendo otimização combinatorial é o Problema do Caixeiro Viajante (PCV) também conhecido como the Travelling Salesman Problem (TSP). Esse consiste na determinação de uma rota de menor custo para algo ou alguém que deseja visitar um conjunto finito de cidades (de modo geral queremos percorrer a menor quantidade de pontos com o menor custo possível). De modo geral, queremos ao iniciar uma viagem em uma cidade qualquer, passar por todas as demais cidades exatamente uma vez, e então retornar para a cidade ou ponto onde iniciamos.

Agora, de uma forma mais formal, temos um conjunto de cidades e uma matriz de distâncias entre elas, o Problema do Caixeiro Viajante tem como objetivo encontrar uma rota que:

- parta da cidade origem;
- passe por todas as demais cidades uma única vez;
- retorne à cidade origem ao final do percurso;
- percorra uma rota que dá a menor distância possível.

Dizemos, na teoria dos grafos que um caminho, ou rota que satisfaz as condições anteriores é um ciclo fechado Hamiltoniano (um ciclo Hamiltoniano, circuito Hamiltoniano, passeio em vértices ou grafo ciclo é um ciclo que visita cada vértice exatamente uma vez, exceto o vértice que é tanto o início quanto o fim, e portanto é visitado duas vezes). Visto as observações anteriores temos que o problema do TSP é um problema de otimização de difícil resolução.

Mesmo sendo simples de definir, obter boas soluções não é tarefa fácil. Para altas dimensões, a resolução utilizando métodos clássicos de otimização e programação inteira é limitada por termos de tempos computacionais. Portanto, a abordagem de solução heurística e de aproximação é mais útil para as aplicações que preferem o tempo de execução do algoritmo em relação à precisão do resultado.

2 Método

Uma formulação matemática conhecida na literatura é a de Dantzig-Johnson utilizando grafos. Se considerarmos um Grafo $G = (N, A)$, sendo N os nós (as cidades no nosso caso) e A as arestas (estradas/percurso), temos o seguinte modelo:

$$
\begin{aligned}
\text{Minimizar } z = & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\
\text{s.a. } & \sum_{i=1}^{n} x_{i,j} = 1, && \forall i \in \{1,\ldots,n\} \\
& \sum_{j=1}^{n} x_{i,j} = 1, && \forall i \in \{1,\ldots,n\} \\
& \sum_{i,j\in S} < |S| - 1 && \forall\, S \subset N \\
& x_{i,j} \in \{0,1\}, && \forall i,j \in \{1,\ldots,n\}.
\end{aligned}
$$

No modelo anterior temos que:

- N, representa o número de nós da rede, ou seja o número de cidades;
- $\sum_{i=1}^{n} x_{i,j} = 1$, nos diz que todo fluxo que chega ao nó j é igual a 1;
- $\sum_{j=1}^{n} x_{i,j} = 1$, nos diz que todo fluxo que sai do nó i é igual a 1;
- S, representa subgrafos de G, isto é, sub rotas quando relacionamos com as cidades;
- $|S|$, representa a quantidade de vértices do subgrafo;
- $\sum_{i,j\in S} < |S| - 1 \;\; \forall\, S \subset N$, nos diz que devem ser evitados subciclos;
- $x_{ij} = 1$, se o fluxo passar pelo arco (i, j) e igual a 0, casco contrário.

Como adotado a seguir, temos que $x_{ii} = 0$, uma vez que não queremos passar pela mesma cidade e que não faz sentido considerar o caminho de uma cidade nela mesma.

2.1 Heurística

Considere o Problema do Caixeiro Viajante no grafo $G = (N, A)$, onde N é o conjunto de vértices de G e é dado por $N = \{e_1, \ldots, e_n\}$ e A é o conjunto de arestas entre os vértices de G. Podemos representar N como uma matriz $M \in \mathbb{R}^n \times \mathbb{R}^n$, onde $m_{i,j}$ é o peso da aresta que liga os vértices e_i e e_j, para todo $i, j \in \{1, \ldots, n\}$. Neste trabalho, foi considerado que $m_{i,i} = 0$ e $m_{i,j} = m_{j,i}$ para todo $i, j \in \{1, \ldots, n\}$.

A heurística desenvolvida para resolver o Problema do Caixeiro Viajante no grafo G de modo geral, inicia o caminho em e_1 e a cada passo escolhe um novo vértice e o insere em um caminho de modo que busque minimizar o caminho final.

Se um $Caminho$ é um conjunto ordenado de elementos de G (que indica o resultado obtido pela heurística), o vértice escolhido em cada passo é um vértice e_j de G tal que $j \in \{1, \ldots, n\}$, $e_j \notin Caminho$ e

$$\sum_{e_i \in Caminho} m_{i,j} \leq \sum_{e_i \in Caminho} m_{i,k},$$

para todo $k \in \{i \in \{1, \ldots, n\} \mid e_i \notin Caminho\}$.

Ou seja, a soma das distâncias do vértice e_j para os vértices já inseridos no caminho é a menor dentre aqueles que ainda não estão no caminho. Este passo é feito pela função "Minimo" no Algoritmo 5.

Depois disso, a heurística verifica qual é a melhor posição para inserir e_j no conjunto $Caminho$, minimizando a distância total do caminho. Passo realizado pela função "Melhor" no Algoritmo 5. E por fim o vértice é inserido no $Caminho$ (função "Insere" em 5).

Por fim, estes últimos três passos são repetidos $n - 1$ vezes, o conjunto $Caminho$ indica a ordem dos vértices a serem visitados e o ciclo é fechado voltando do último vértice indicado por $Caminho$ para o primeiro.

3 Testes

Como determinado inicialmente no trabalho, contamos com um total de cinco problemas testes que consistem em arquivos do tipo *.txt* contendo a

matriz de distâncias (neste caso simétrica) e a solução ótima para o problema. A posição $m_{i,j}$ da matriz representa a distância da cidade i para a cidade j. Notemos que o modo como a matriz esta definida é a mesma utilizada na heurística definida anteriormente.

O primeiro teste consta com um total de 5 cidades, onde a solução ótima, isto é, a menor distância percorrida pelo viajante ao passar por todas as cidades é igual a 19. O segundo consta com 17 cidades e com solução ótima igual a 2085, o terceiro com 26 cidades, o quarto com 42 cidades e a quinta e última com 48 cidades, com soluções ótimas dadas por 937, $*$ e 33551 respectivamente.

A tabela abaixo contém os dados dos 5 testes, onde:

- a primeira linha representa a distância obtida pela heurística descrita na seção anterior;
- a segunda linha representa a solução ótima para o problema, isto é, a menor distância possível que pode ser percorrida;
- a terceira linha representa a diferença entre a heurística e a solução ótima;
- a quarta e última linha representa a solução obtida utilizando a heurística clássica *Nearest Neighbor*, conhecida também como algoritmo do vizinho mais próximo.

	Teste				
	5	**17**	**26**	**42**	**48**
Heuristica	21	2159	1078	842	37030
Ótima	19	2085	937	*	33551
Diferença	2	74	141	*	3479
NN	21	2187	1112	956	40551

4 Conclusão

Analisando os dados obtidos, temos que a heurística apresentou caminhos com distâncias em média 10% maiores que as soluções ótimas. Além disso, comparando os resultados obtidos pela heurística apresentada neste trabalho com aqueles obtidos pela heurística do vizinho mais próximo, percebe-se que a primeira apresentou melhores resultados em todos os testes exceto o primeiro, no qual os resultados foram iguais.

5 Código

```
# Inicializacao

import numpy as np
from math import *

# Funcoes Auxiliares

def pertence(x, v):
    for i in range(v.size):
        if(v[i] == x):
            return True
    return False

def Minimo(M, v):
    r = inf
    prox = 0

    for j in range(M[0]. size):
        T = 0
        if(not(pertence(j, v))):
            for i in range(M[0].size):
                if(pertence(i, v)):
                    T = T + M[i][j]
            T = T/(M[0].size + 1)
        if(T <= r and T != 0):
            prox = j
            r = T
    return prox

def Melhor(x, M, v):
    melhor = inf
    coord = v.size
    for i in range(v.size+1):
        testar = np.insert(v, i, x)
        T = Total(M, testar)
        if(T <= melhor):
            melhor = T
            coord = i
```

```
    return coord

def Total(M, v):
    total = 0
    for i in range(v. size-1):
        total = total + M[v[i]][v[i+1]]
    total = total + M[v[0]][v[v.size-1]]
    return total

# Codigo principal

# Entradas

# Ler a matriz
matriz = np.loadtxt("Teste_48.txt").astype(int)

# Quantidade de vertices
n = matriz[0].size

# Iniciar caminho no vertice 0
caminho = np.array([0])

# Heuristica

prox = 0
for i in range(n-1):
    r = inf
    prox = Minimo(matriz, caminho)
    caminho = np.insert(caminho, Melhor(prox, matriz, caminho), prox)

caminho = np.insert(caminho, caminho.size, caminho[0])

print(caminho + 1)
```

6 Referências

1. V. E. Wilhelm. Problemas de caixeiro-viajante. *Departamento de Engenharia de Produção - UFPR*, 86:102. *Disponível em: https://bit.ly/3bDROC7.*

2. M. A. P. Rodrigues. Problema do caixeiro viajante: um algoritmo para resolução de problemas de grande porte baseado em busca local dirigida. *Dissertação (Mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico*, *Disponível em: https://bit.ly/3oOm1C7.*

3. L. M. Enami, L. L. da Silva, P. V. Stocco, R. Komatsu, A. Corsi. O Problema do caixeiro viajante através de algoritmo genético. *IX Congresso brasileiro de engenharia de produção*, *Disponível em: https://bit.ly/39sp9NC.*