UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO

Ueslei Aparecido Moreira Santos Pina - 11837081
João Otávio da Silva - 12563748
Nicolas de Góes - 12563780
Eduardo Maciel de Matos - 12563821

PROPOSTA DE DESENVOLVIMENTO:
Problema do Caixeiro Viajante

São Carlos
2022

Sumário

1. Introdução

O problema do caixeiro viajante é um problema clássico da otimização em que se deseja encontrar o caminho mais curto para percorrer todas as cidades (ou localidades) de um conjunto, visitando cada uma delas apenas uma vez e retornando ao ponto de partida. Esse problema é comumente usado como um exemplo para ilustrar o uso de algoritmos de otimização e é amplamente aplicável em diversas áreas, incluindo a logística.

O seguinte relatório tem como objetivo documentar a solução de um problema de otimização inteira, a respeito de busca de melhor rota . O problema escolhido pelo grupo tem como base o problema do caixeiro viajante em grafo não direcionado completo, visto em aula.

O nosso cenário consiste em uma empresa de logística precisa entregar mercadorias em diversas cidades, o objetivo é encontrar a rota mais eficiente para que o caminhão percorra todas as cidades, minimizando o tempo e o custo total da viagem. Tendo conhecimento de todas as distâncias entre as cidades, buscamos encontrar a melhor rota que sai do centro de distribuição, passa uma única vez por todas as outras cidades e retorna, de forma a minimizar a distância total.

2. Descrição

Temos uma lista com vinte cidades Araraquara, onde se encontra o Centro de Distribuição, Barretos, Batatais, Bauru, Botucatu, Franca , Itu , Jaboticabal, Lins, Marília, Mogi das Cruzes, Piracicaba, Presidente Prudente', 'Ribeirão Preto, São Carlos, São José do Rio Preto, São José dos Campos, São Paulo, Sorocaba e Taubaté.

Em um matriz está a distância em quilômetros entres as cidades:

```
distancias = [
    [  0, 116,  91, 205, 123, 246, 179, 222, 204, 141, 101, 130, 290, 215,  97, 180,  78,  63, 170, 148],
    [116,   0, 193,  95, 200, 369, 302, 345, 327, 264, 224, 253, 413, 338, 120, 303, 201, 186, 293, 271],
    [ 91, 193,   0, 309,  26, 249, 182, 225, 207, 144, 104, 133, 293, 218, 100, 183,  81,  66, 173, 151],
    [205,  95, 309,   0, 315, 432, 365, 408, 390, 327, 287, 316, 476, 401, 183, 366, 264, 249, 356, 334],
    [123, 200,  26, 315,   0, 235, 168, 211, 193, 130,  90, 119, 279, 204,  86, 169,  67,  52, 159, 137],
    [246, 369, 249, 432, 235,   0,  67, 110,  92,  29,  11,  58, 218, 143,  25, 108,   6,   9,  94,  72],
    [179, 302, 182, 365, 168,  67,   0,  43,  25,  38,  78,  49, 181, 106,  20,  93,  81,  96,   3,  19],
    [222, 345, 225, 408, 211, 110,  43,   0,  82,  45,   5,  34, 194, 119, 101,  84,  28,  43,  50,  28],
    [204, 327, 207, 390, 193,  92,  25,  82,   0,  57,  17,  46, 166,  91,  73,  56,  50,  65,  32,  10],
    [141, 264, 144, 327, 130,  29,  38,  45,  57,   0,  40,  11, 123,  48,  30,  13,  83,  98,  45,  23],
    [101, 224, 104, 287,  90,  11,  78,   5,  17,  40,   0,  29,  83,   8,  20,   3, 115, 130,   5,  17],
    [130, 253, 133, 316, 119,  58,  49,  34,  46,  11,  29,   0, 112,  37,  19,   2,  86, 101,  34,  12],
    [290, 413, 293, 476, 279, 218, 181, 194, 166, 123,  83, 112,   0, 125, 107, 190,  88,  73, 180, 158],
    [215, 338, 218, 401, 204, 143, 106, 119,  91,  48,   8,  37, 125,   0,  82, 165,  63,  48, 155, 133],
    [ 97, 120, 100, 183,  86,  25,  20, 101,  73,  30,  20,  19, 107,  82,   0,  83,  29,  44,  61,  39],
    [180, 303, 183, 366, 169, 108,  93,  84,  56,  13,   3,   2, 190, 165,  83,   0,  98, 113, 366, 300],
    [ 78, 201,  81, 264,  67,   6,  81,  28,  50,  83, 115,  86,  88,  63,  29,  98,   0,  72, 295, 229],
    [ 63, 186,  66, 249,  52,   9,  96,  43,  65,  98, 130, 101,  73,  48,  44, 113,  72,   0, 223, 157],
    [170, 293, 173, 356, 159,  94,   3,  50,  32,  45,   5,  34, 180, 155,  61, 366, 295, 223,   0,  76],
    [148, 271, 151, 334, 137,  72,  19,  28,  10,  23,  17,  12, 158, 133,  39, 300, 229, 157,  76,   0]]
```

O resultado permite que qualquer uma das cidades seja interpretada como o centro de distribuição, simplesmente utilizando a cidade pretendida como ponto de partida. Porém utilizamos a cidade de Araraquara como centro de distribuição.

Podemos escolher o número total de cidades que serão utilizadas, por exemplo, para um *n* = 8. As 8 primeiras cidades serão incluídas no modelo (o problema é relativamente custoso, então a medida que n cresce, o tempo de execução torna a resolução inviável). Ao final, o programa nos retorna qual a rota escolhida e a distância total percorrida na mesma.

3. Modelagem

O problema foi modelado da mesma forma que foi feito na aula.

$$C_{ij} = \text{distância entre as cidades } i \text{ e } j$$

$$X_{ij} = \begin{cases} 1 & \text{se usamos o caminho da cidade } i \text{ até a } j \\ 0 & \text{caso contrário} \end{cases}$$

$$\begin{aligned}
& min.: \sum_{i=1}^{n} \sum_{j=1}^{n} C_{ij} X_{ij} \\
& s.a.: \sum_{i=1}^{n} X_{ij} \geq 1 && \forall j \in \{1, 2, \ldots, n\} && (R1) \\
& \sum_{i=1}^{n} X_{ij} = \sum_{i=1}^{n} X_{ji} && \forall j \in \{1, 2, \ldots, n\} && (R2) \\
& \sum_{(i_{ij}) \in S} X_{ij} \leq |S| - 1 && \forall s : |S| \leq \left\lceil \frac{n}{2} \right\rceil && (R3)
\end{aligned}$$

A função objetiva retorna a distância total percorrida na rota, queremos minimizá-la.

A restrição (R1) serve para garantir que para toda cidade, há, pelo menos, uma aresta que chega nela.

A restrição (R2) serve para garantir que para cada aresta que chega em uma cidade, também há uma correspondente que sai da mesma.

A restrição (R3) serve para garantir que o modelo não aceite ciclos. Para os subcaminhos de tamanho *n* menor ou igual a metade do total de cidades, impõe-se que deve haver no máximo *n - 1* arestas, seria o equivalente a remover uma aresta do caminho para que não se feche o ciclo.

Obs: informações do modelo completamente montado, com os valores, podem ser vistas no arquivo *modelo.lp*

4. Código

O problema foi resolvido utilizando a linguagem de programação Python e duas de suas bibliotecas: *numpy* para utilização de vetores e *MIP*, o solver propriamente dito. Utilizamos o código feito em sala como base.

O programa começa com a importação das bibliotecas, com a definição da lista das cidades, da matriz de distâncias e do número de cidades a serem incluídas no problema.

```python
import mip
import numpy as np

cidades = ['Araraquara', 'Barretos', 'Batatais', 'Bauru', 'Botucatu',
           'Franca', 'Itu', 'Jaboticabal', 'Lins', 'Marília']

distancias = [
    [0, 116, 91, 205, 123, 246, 179, 222, 204, 141],
    [116, 0, 193, 95, 200, 369, 302, 345, 327, 264],
    [91, 193, 0, 309, 26, 249, 182, 225, 207, 144],
    [205, 95, 309, 0, 315, 432, 365, 408, 390, 327],
    [123, 200, 26, 315, 0, 235, 168, 211, 193, 130],
    [246, 369, 249, 432, 235, 0, 67, 110, 92, 29],
    [179, 302, 182, 365, 168, 67, 0, 43, 25, 38],
    [222, 345, 225, 408, 211, 110, 43, 0, 82, 45],
    [204, 327, 207, 390, 193, 92, 25, 82, 0, 57],
    [141, 264, 144, 327, 130, 29, 38, 45, 57, 0]]

# Quantidade de cidades a serem utilizadas no modelo.
n = 10
```

Agora partimos para as configurações iniciais do modelo, preenchemos o vetor de custos com as distâncias entre as cidades, inicializamos o modelo com a biblioteca MIP, adicionamos as variáveis referentes a quais arestas serão utilizadas e adicionamos a função objetiva.

```python
# Preenche matriz de custos, o elemento Cij corresponde a distância entre as cidades de índice i e j.
C = np.zeros( ( n, n ) )
for i in range( n ):
    for j in range( i + 1, n ):
        C[ i, j ] = distancias[ i ][ j ]
        C[ j, i ] = C[ i, j ]


# Inicializa o modelo no solver.
m = mip.Model( sense = mip.MINIMIZE, solver_name = mip.CBC )

# Adiciona variáveis.
x = []
for i in range( n ):
    x.append( [ m.add_var( var_type = mip.BINARY, name = 'x_(%i,%i)' % ( i + 1, j + 1 ) )  for j in range( n ) ] )

# Adiciona função objetiva.
m.objective = mip.xsum( C[ i, j ] * x[ i ][ j ] for i in range( n ) for j in range( n ) )
```

Com o modelo inicializado, adicionamos as restrições R1, R2 e R3, citadas na seção anterior. Os subconjuntos utilizados para adicionar a restrição R3 são gerados a partir da função recursiva *gera_subconjs.*

```python
# Adiciona restrição (R1).
for j in range( n ):
    m += mip.xsum( x[ i ][ j ] for i in range( n ) ) >= 1


# Adiciona restrição (R2).
for j in range( n ):
    m += mip.xsum( x[ i ][ j ] for i in range( n ) ) == mip.xsum( x[ j ][ i ] for i in range( n ) )

# Gera os subconjuntos de tamanho k, contendo números de 0 até n - 1.
def gera_subconjs( k, n ):

    if k == 1:
        retval = []
        for i in range( n ):
            retval.append( { i } )
        return retval

    subsubs = gera_subconjs( k - 1, n )

    retval = []
    for i in range( n ):
        for s in subsubs:
            tmp = s.copy()
            tmp.add( i )
            if not tmp in retval:
                retval.append( tmp )

    return retval


subconjs = gera_subconjs( int( ( n + 1 ) / 2 ), n )

# Adiciona restrição (R3).
for s in subconjs:
    m += mip.xsum( x[ i ][ j ] for i in s for j in s ) <= len( s ) - 1
```

Finalmente, escrevemos um arquivo com as informações do modelo completamente montado, que será resolvido, executamos o solver e imprimimos informações a respeito da solução.

```
# Escreve arquivo com informações do modelo.
m.write('modelo.lp')

# Executa o solver.
status = m.optimize(max_seconds=300)

# Imprime informações da solução.
for v in m.vars:
    print('{}: {}'.format(v.name, v.x))

print(status)
print(status.value)

# Constrói o vetor com o circuito
circuito = []
anterior = 0
total = 0
circuito.append(cidades[0])
for ponto in range(n):
    for j in range(n):
        if x[anterior][j].x == 1:
            circuito.append(cidades[j])
            total += distancias[anterior][j]
            anterior = j
            break

# Imprime o circuito
print("\nCircuito a ser percorrido: ", circuito, "\n")
print(total, "Km percorridos")
```

O arquivo *instructions.txt* presente no zip, possui instruções para rodar o código.

5. Resultados Obtidos

O resultado do problema do caixeiro viajante é uma rota que visita todas as cidades (ou localidades) de um conjunto, minimizando o tempo e o custo total da viagem. No contexto de uma empresa de logística, o resultado desse problema pode ser usado para planejar rotas de entrega de mercadorias de maneira mais eficiente, minimizando o tempo e o custo da viagem e maximizando a lucratividade da empresa. Além disso, o resultado também pode ser usado para otimizar o planejamento de rotas de transporte público, a distribuição de produtos e outras aplicações.

A saída do programa retorna o percurso e a quilometragem percorrida, como o exemplo a seguir:

Circuito a ser percorrido: ['Araraquara', 'Batatais', 'Botucatu', 'Jaboticabal', 'Itu', 'Lins', 'Franca', 'Marília', 'Bauru', 'Barretos', 'Araraquara']
1055 km percorridos