
UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
SME0510 - INTRODUÇÃO À PESQUISA OPERACIONAL

# PROJETO

Heitor Pupim Assunção Toledo, nºUSP: 11372858
Rafael Scalon Peres Conti , nº USP: 11871181
Rafael Jun Teramae Dantas, nº USP: 12563686
Jade Bortot de Paiva , nº USP: 11372883

São Carlos

2022

# SUMÁRIO

# 1 INTRODUÇÃO

O grupo escolheu modelar um problema do caixeiro viajante. Para tal, o grupo utilizou as bibliotecas mip, que nos fornece o Solver da solução, numpy, utilizada para gerar números aleatórios e matplotlib, para representar o problema de forma gráfica, cujas documentações podem ser encontradas em: MIP, Numpy e Matplotlib. Dessa forma, o problema se trata de um problema do **caixeiro viajante** para um caminhão de distribuição de uma fábrica de bolacha.

# 2 SOBRE O SOLVER

O **MIP** é uma bilbioteca em Python que proporciona ferramentas de modelagem e solução de **Problema de Programação Inteira Mista**, que é o caso mais geral. O problema do Caixeiro Viajante é um problema clássico de optimização combinatória, no qual os primeiros estudos datam de 1950. O problema consiste em encontrar o caminho mais curto de visita em cada um dos lugares dos quais o viajante irá passar. Desse modo, considerando $n$ pontos de $V = 0, ..., n-1$ e a matriz de distâncias $D_{nxn}$ com elementos $c_{i,j} \in \mathbb{R}^+$, em que uma solução consiste em um conjunto de pares $n$(origem,destino) que indicam o itinerário da viagem, resultando na formulação a seguir:

Minimizar:

$$\sum_{i \in I, j \in I} c_{i,j}.\, x_{i,j}$$

Sujeito à:

$$\sum_{j \in V \setminus \{i\}} x_{i,j} = 1 \;\; \forall i \in V$$

$$\sum_{i \in V \setminus \{j\}} x_{i,j} = 1 \;\; \forall j \in V$$

$$y_i - (n+1).\, x_{i,j} \geq y_j - n \;\; \forall i \in V \setminus \{0\}, j \in V \setminus \{0, i\}$$

$$x_{i,j} \in \{0, 1\} \;\; \forall i \in V, j \in V$$

$$y_i \geq 0 \;\; \forall i \in V$$

Desse modo, o Caixeiro Viajante consegue o melhor caminho para sua viagem.

Para criar este modelo, utilizamos uma matriz de coordenadas aleatórias, que serão os **nós**. Assim, criamos o modelo com o **mip.Model**, com os parâmetros a seguir:

```
m = mip.Model(sense = mip.MINIMIZE, solver_name = mip.CBC)
```

Depois, o pacote MIP possui a função **objective** que serve para calcular a **função objetivo**, que para o nosso caso do caixeiro viajante foi: `m.objective = mip.xsum(c[i,j] * x[i][j] for i in range(n) for j in range(n))`

em que **xsum** é uma função que é usada para criar uma expressão linear de um somatório. Poderiamos utilizar a função sum() do python, mas a xsum é uma versão optimizada para gerar a experssão linear de maneira mais rápida.

Após isso, o modelo deixa salvar o modelo MIP ou uma solução viável com **write** Por fim, a função **optimize** do modelo permite optimizar o modelo em questão, podendo ou não passar um parâmetro de tempo máximo.

Depois, podemos verificar um scatterplot com o caminho mais curto e portanto, a solução do problema com a biblioteca **matplotlib** já anteriormente citada, que é capaz de nos gerar um output de figuras.

# 3 RESOLUÇÃO

Tomando como inspiração o problema do caixeiro viajante apresentado em aula, o grupo decidiu modelar e resolver um problema do caixeiro viajante aplicado para um caminhão de distribuição de uma fábrica de bolachas cobertas por chocolate. O problema escolhido foi esse pois avaliamos o problema do caixeiro viajante como o mais interessante dos apresentados na segunda metade da disciplina, além de manter o tema semelhante ao primeiro trabalho da disciplina (sobre sequenciamento de tarefas de uma fábrica de bolachas cobertas por chocolate).

Por limitações de tempo, escolhemos utilizar 15 (quinze) nós, de forma que o modelo fica extenso (possuindo $15^2 = 225$ variáveis) e, além das restrições "padrão" do problema do caixeiro viajante (passar pelo menos uma vez em cada nó e a aresta de ida é a mesma aresta de volta), adicionamos as restrições que impedem subciclos na solução.

Para gerar os nós, utilizamos uma função random do numpy (`numpy.random.normal()`, especificando o parâmetro **size** (que define a forma do output) como uma tupla (2,) e utilizando uma **seed** para gerar os números randômicos igual a $225(n^2)$. Além disso, os números gerados são multiplicados por $32(2^5)$, para que as distâncias entre os nós estejam dentro do esperado para o problema modelado (em km)

A matriz C é criada, inicialmente, como uma matriz de zeros, utilizando a função `np.zeros`, com tamanho $15x15(nxn)$. Em seguida, essa matriz é preenchida utilizando a função `np.linalg.norm(nos[i] - nos[j])` que retorna a norma da operação **nó i - nó j** para a posição $i, j$ da matriz $C$. Por fim, estabelecemos que os valores na posição $j, i$ são iguais aos valores na posição $i, j$ (já preenchidos) de forma que o custo de ir por uma aresta é o mesmo que o de voltar por essa mesma aresta (o caminho de i para j é de mesmo custo que o caminho de j para i).
Em seguida, determinamos as primeiras restrições do problema, utilizando

```
m.add_var()
```

(para adicionar variáveis ao vetor x) e `mip.xsum()` (para os somatórios) e definimos a função objetivo com `m.objective()`. Ademais, utilizamos uma subrotina

```
gera_subconjuntos()
```

apresentada em aula, para adicionar as restrições de subciclos no modelo. Para conferir o modelo desenvolvido criamos o arquivo `'test.lp'` utilizando a função `m.write()`, rodamos a otimização utilizando o `m.optimize()` (cujo status `OptimizationStatus.OPTIMAL` indica que a otimização foi bem sucedida)

Por fim, para representar o caminho encontrado pela otimização, geramos um plot utilizando a biblioteca matplotlib, como demonstrado em aula, resultando em uma caminho fechado, apresentado na figura 1

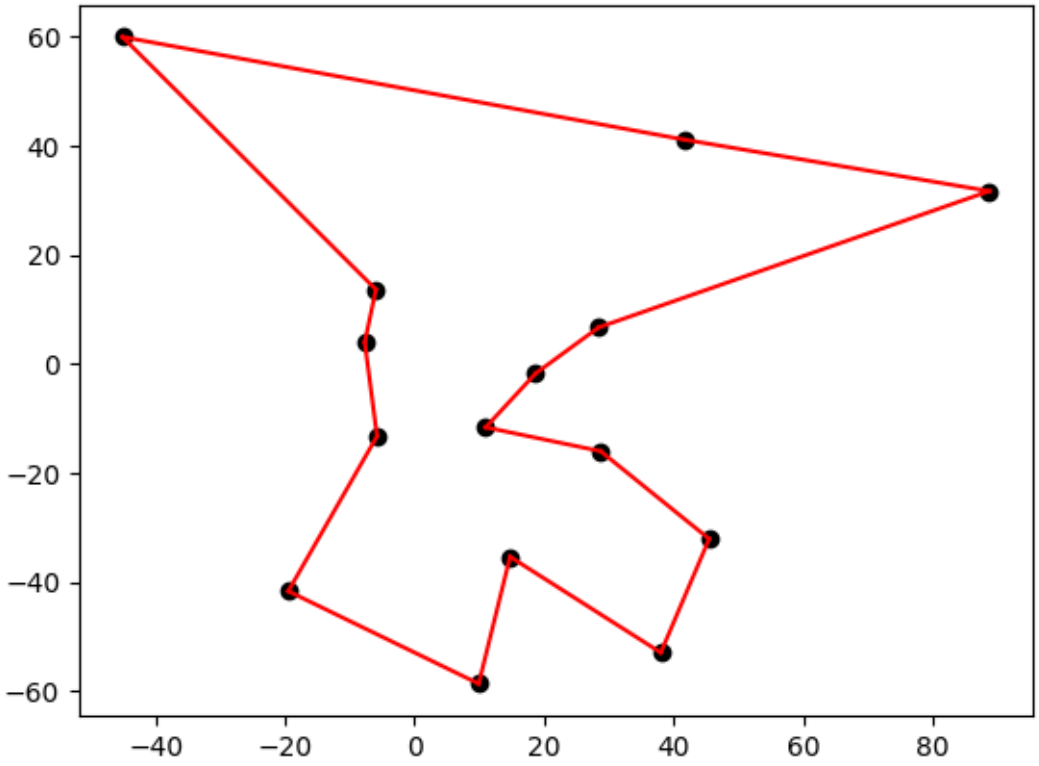


Figura 1 – Percurso da Otimização

# 4 CONCLUSÃO

Ao final do processo, a otimização do modelo foi bem-sucedida (vide o conteúdo presente em status `(OptimizationStatus.OPTIMAL)`. Ademais, a leitura do arquivo `test.lp` indica que o modelagem realizada está dentro do esperado para o problema e segue os objetivos do grupo com o modelo. O caminho obtido pela otimização é coerente e segue o padrão percebido em aula (conectar os nós de forma a "circular" uma única vez pelos pontos. Além disso, ao longo da resolução do problema não houve aparição de subciclos como solução, o que indica bom funcionamento da subrotina vista em aula.

Portanto, o grupo conclui que tanto a modelagem do problema, quanto os resultados obtidos com o uso do solver foram satisfatórios e condizem com o conteúdo, os modelos e as soluções apresentadas nas aulas da disciplina.

# REFERÊNCIAS

[1] NUMPY. **Numpy**. Documentação da biblioteca numpy. Disponível em: <https://numpy.org/>. Acesso em: 14 de dezembro de 2022.

[2] MIP. **MIP**. Documentação da biblioteca MIP. Disponível em: <https://www.python-mip.com/>. Acesso em: 14 de dezembro de 2022.

[3] MATPLOTLIB. **MatplotLib**. Documentação da biblioteca Matplotlib. Disponível em: <https://matplotlib.org/>. Acesso em: 14 de dezembro de 2022.

[4] LINALG.NORM . **Numpy Linalg Norm**. Documentação da função Linalg Norm do Numpy. Disponível em: <https://numpy.org/doc/stable/reference/generated/numpy.linalg.norm.html>. Acesso em: 14 de dezembro de 2022.

[5] RANDOM.NORMAL. **Numpy Random Normal**. Documentação função numpy.random.normal. Disponível em: <https://numpy.org/doc/stable/reference/random/generated/numpy.random.normal.html>. Acesso em: 14 de dezembro de 2022.