UNIVERSIDADE FEDERAL DO TOCANTINS
CÂMPUS DE PALMAS
CIÊNCIA DA COMPUTAÇÃO

PABLO HENRIQUE, LUANA LORENA, MARCOS GILMÁRIO E KHARLOS DANYELL

TRABALHO - PESQUISA OPERACIONAL

PROBLEMA DO CAIXEIRO VIAJANTE

PALMAS/TO
2022

1 PROBLEMA DO CAIXEIRO VIAJANTE

O problema do caixeiro viajante é um problema de otimização combinatória que identifica a menor rota para percorrer uma série de cidades, retornando à cidade de origem. Suponha que um caixeiro viajante tenha de visitar **n** cidades diferentes, iniciando e encerrando sua viagem na primeira cidade. Suponha, também, que não importa a ordem com que as cidades são visitadas e que de cada uma delas pode-se ir diretamente a qualquer outra. O problema do caixeiro viajante consiste em descobrir a rota que torna mínima a viagem total.

1.1 O PROBLEMA DO CAIXEIRO VIAJANTE APLICADO A PROGRAMAÇÃO LINEAR

A formulação do problema em torno da programação linear envolve descrever as restrições de acordo com um conjunto de regras do problema do caixeiro viajante, para então otimizar a função objetivo. Problema do Caixeiro Viajante (PCV), do inglês, Traveling Salesman Problem, é um dos mais tradicionais e conhecidos problemas de otimização combinatória. Ele consiste em um conjunto de nós V={1,...,n} e um conjunto de arestas E, onde os nós os representam cidades e as arestas representam pares ordenados de cidades pelos quais uma ligação direta é possível. Além disto, para cada par ordenado, um peso, proporcional à distância ou ao tempo de viagem, é atribuído. O problema é encontrar um caminho, começando da cidade 1, que, além de visitar cada cidade exatamente uma vez, retorne para a cidade 1 com o tempo total de viagem mínimo.

Resumindo, o problema é encontrar um caminho Hamiltoniano de menor tempo, ou de mínimo custo total, num grafo G=(V,E). Além da função objetiva que mensura o custo de passagem nos arcos, o modelo é formado por quatro conjuntos de restrições. Os dois primeiros conjuntos de restrições obrigam que chegue e saia um, e somente um, arco em cada nó do grafo G=(N,E) que serve como suporte para o modelo. Outro conjunto de restrições elimina ciclos de tamanho menor que | N | e, o último conjunto impõe a condição de integralidade para o problema. O número de restrições que inibem ciclos cresce de forma exponencial em relação ao aumento do número de nós do grafo.

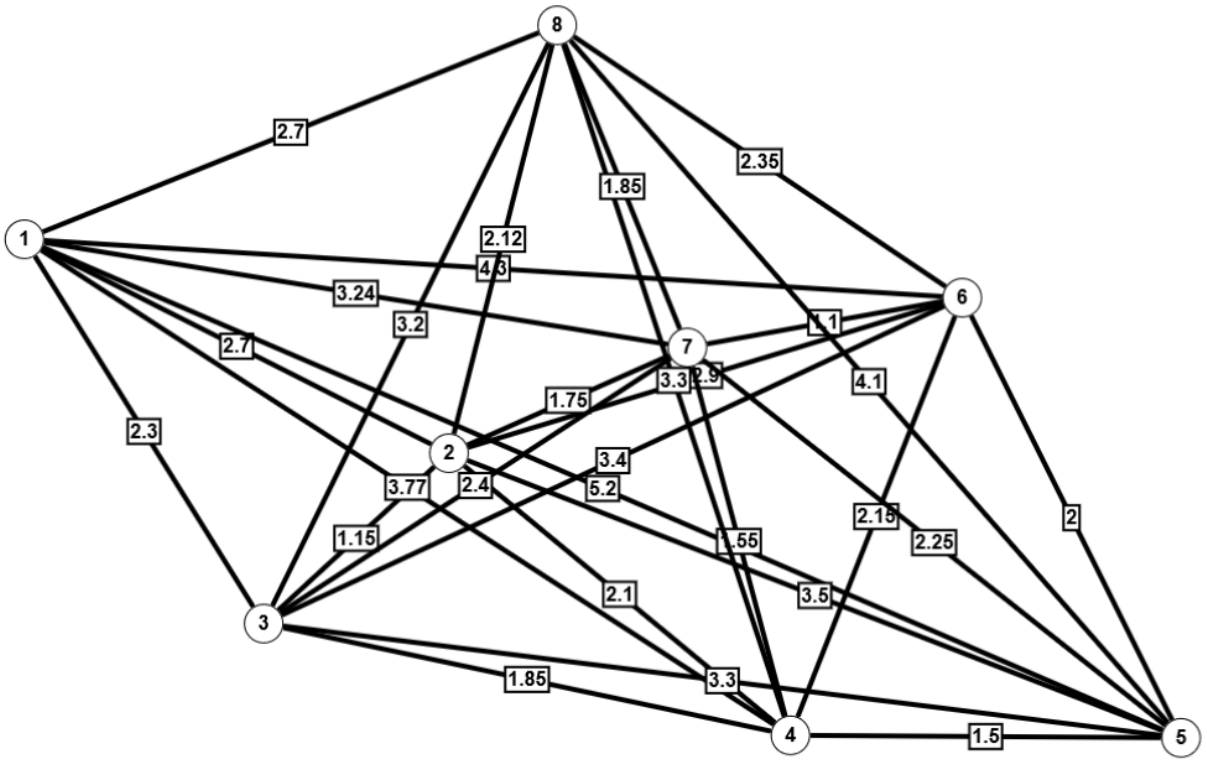
Nesta formulação, o caixeiro viajante deve visitar as outras n-1 cidades, salvo a cidade 1, exatamente uma vez. Durante seu trajeto ele deve retornar à cidade origem exatamente t vezes, incluindo o retorno final, e não deve visitar mais de p cidades diferentes em um ciclo.

2 MODELO DO PROBLEMA

Sabendo que um problema de fluxo de rede pode ser formulado como um problema de otimização combinatória, é possível capturar um grafo não direcionado como exemplar e, então, traduzi-lo para um problema de programação linear de acordo com a definição da seção 1.1 deste relatório. Para este modelo, é mais usado a aplicação em grafos, no qual cada cidade é um vértice ou nó do grafo e as estradas são os arcos. Cada vértice do problema não tem conexão direta com outros, possuindo até 7 arestas.

Portanto, a estrutura matemática é um gráfico onde cada cidade é denotada por um ponto (ou nó) e linhas são desenhadas conectando cada dois nós (chamados arcos ou arestas). Associado com cada linha é uma distância (ou custo). Quando a pessoa pode ir de cada cidade para outra cidade diretamente, diz-se que o grafo está completo. Uma viagem de ida e volta pelas cidades corresponde a algum subconjunto das linhas e é chamado de passeio ou ciclo hamiltoniano em teoria dos grafos. A duração de um passeio é a soma dos comprimentos das linhas na ida e volta.

Diante do exposto, considera-se o seguinte problema do caixeiro viajante para resolver utilizando a programação linear:



As variáveis do caixeiro viajante são determinadas por x_{ij} no qual **i** representa o primeiro nó do caminho ao segundo nó **j** das distâncias entre as cidades. Seus respectivos valores são:

$x_{11} = 0$ $x_{12} = 2.7$ $x_{13} = 2.3$ $x_{14} = 3.77$ $x_{15} = 5.2$ $x_{16} = 4.3$

$x_{17} = 3.24$ $x_{18} = 2.7$ $x_{21} = 2.7$ $x_{22} = 0$ $x_{23} = 1.15$ $x_{24} = 2.1$

$x_{25} = 3.3$ $x_{26} = 2.9$ $x_{27} = 1.75$ $x_{28} = 2.12$ $x_{31} = 2.3$ $x_{32} = 1.15$

$x_{33} = 0$ $x_{34} = 1.85$ $x_{35} = 3.3$ $x_{36} = 3.4$ $x_{37} = 2.4$ $x_{38} = 3.2$

$x_{41} = 3.77$ $x_{42} = 2.1$ $x_{43} = 1.85$ $x_{44} = 0$ $x_{45} = 1.5$ $x_{46} = 2.15$

$x_{47} = 1.55$ $x_{48} = 3.3$ $x_{51} = 5.2$ $x_{52} = 3.5$ $x_{53} = 3.3$ $x_{54} = 1.5$

$x_{55} = 0$ $x_{56} = 2$ $x_{57} = 2.25$ $x_{58} = 4.1$ $x_{61} = 4.3$ $x_{62} = 2.9$

$x_{63} = 3.4$ $x_{64} = 2.15$ $x_{65} = 2$ $x_{66} = 0$ $x_{67} = 1.1$ $x_{68} = 2.35$

$x_{71} = 3.24$ $x_{72} = 1.75$ $x_{73} = 2.4$ $x_{74} = 1.55$ $x_{75} = 2.25$ $x_{76} = 1.1$

$x_{77} = 0$ $x_{78} = 1.85$ $x_{81} = 2.7$ $x_{82} = 2.12$ $x_{83} = 3.2$ $x_{84} = 3.3$

$x_{85} = 4.1$ $x_{86} = 2.35$ $x_{87} = 1.85$ $x_{88} = 0$

O objetivo consiste em minimizar o caminho de 1 a 8. Dessa forma, a função objetivo será a minimização do custo total da rota do caixeiro, em que x_{ij} é o custo da distância do arco **i**, **j** variando de 1 a 8 e $B_{i,j}$ é a variável binária (0 ou 1) que decide se a rota i,j entrará ou não na solução do problema.

$$Min = \sum_{i=1}^{8} \sum_{j=1}^{8} \in E \; [\, Xi,j * Bi,j \,]$$

As restrições são:

$$\sum_{i=1}^{8} \sum_{j=1}^{8} \in E \; Bi,j = 1 \; , \; \forall \; i \; , \; \forall \; j$$

Para a saída e entrada dos nós, onde **B** representa uma variável binária indicadora da rota (0 ou 1) que decide a rota i,j se entrará ou não na solução do problema. Significa dizer que se a cidade é visitada, ela é determinada pelo valor binário 1, e se não por 0 , e **E** o conjunto de arcos do grafo. A restrição garante que cada nó tenha uma entrada e uma saída para o grafo otimizado. Essa restrição indica que cada um dos pontos é visitado somente uma vez, ou seja, tanto como início quanto como final.

As restrições de conservação são:

$$X_{i,j} \in \{0;1\},\ \forall\ i,j \in E$$

A última restrição indica que entre quaisquer dois subconjuntos complementares de pontos há pelo menos um arco de ligação. Essa restrição evita que um modelo apresente como solução dois subconjuntos não conectados.

$$\sum_{i \in S} \sum_{j \in S} \in E\ Bi,j \geq 1\ ,\ \forall\ S\ \subset \text{conjunto total dos pontos ta faltando o que?}$$

2.1 MODELO EM MATHPROG

O modelo em MathProg foi baseado em um exemplo da própria GLPK e consiste em um Problema do Caixeiro Viajante Seja dado um grafo direcionado G = (V, E), onde V = {1, ..., n} é um conjunto de nós, E <= V x V é um conjunto de arcos. Deixe também cada arco **e** = (i,j) ser atribuído um número c[i,j], que é o comprimento do arco **e**. O problema é encontrar um caminho fechado de comprimento mínimo indo através de cada nó de G exatamente uma vez.

https://github.com/firedrakeproject/glpk/blob/master/examples/tsp.mod

Código fonte do modelo em MathProg

Além disso, o código nos fornece de maneira formatada o fluxo que percorre cada vértice.

```
printf  " Tour ideal tem comprimento %d\n " ,
   soma {(i,j) em E} c[i,j] * x[i,j];
printf ( " Distância do nodo ao nodo\n " );
printf {(i,j) em E : x[i,j]} "       %3 d        %3 d    %8 g\n " ,
   i, j, c[i,j];

dados ;
```

Código responsável por exibir o resultado

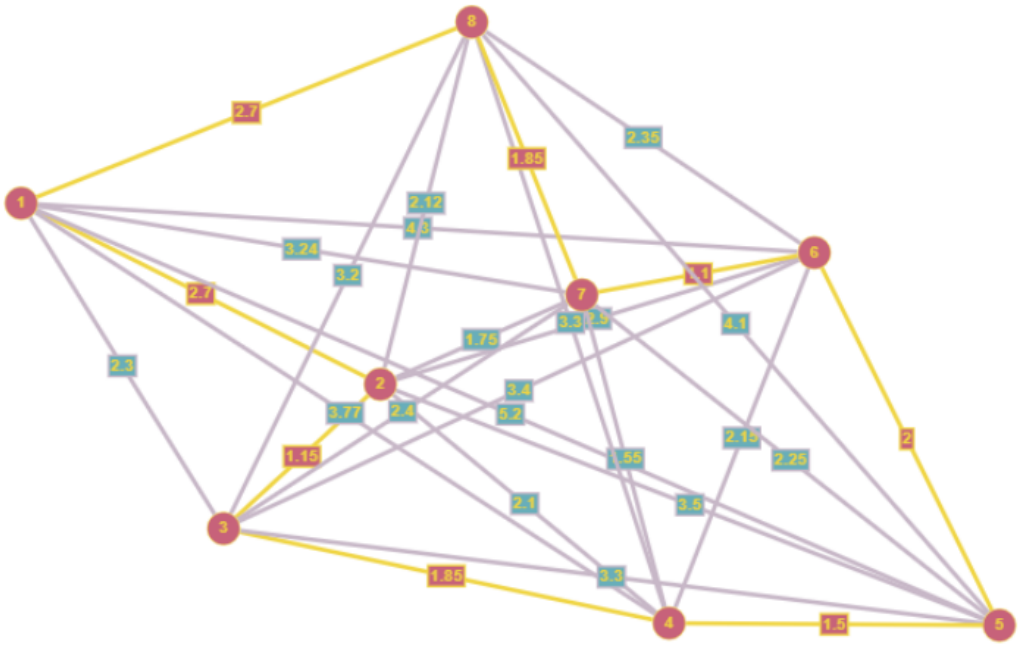
Após o código fonte do programa em si, tem-se a parte de inserção dos dados do problema, onde introduzimos distância do nó i ao nó j e o conjunto de arcos do problema adotado, para que o algoritmo possa calcular o resultado.

```
parâmetro n :=  16 ;

parâmetro  : E : c :=
    1   2    509
    1   3    501
    1   4    312
    1   5   1019
    1   6    736
    1   7    656
    1   8     60
    1   9   1039
    1  10    726
    1  11   2314
    1  12    479
    1  13    448
    1  14    479
    1  15    619
    1  16    150
    2   1    509
    2   3    126
    2   4    474
    2   5   1526
    2   6   1226
    2   7   1133
    2   8    532
    2   9   1449
    2  10   1122
    2  11   2789
    2  12    958
    2  13    941
    2  14    978
    2  15   1127
    2  16    542
    3   1    501
    3   2    126
    3   4    541
    3   5   1516
```

Inserção de dados do problema no modelo em MathProg

3 RESULTADOS



Rodando o programa em MathProg com o seguinte comando: **glpsol -m max_tsp.mod**, onde max_tsp.mod é o arquivo onde se encontra o algoritmo, obtemos o seguinte resultado:

```
Memory used: 0.4 Mb (431872 bytes)
Optimal tour has length 14.70
From node   To node   Distance
        1         3        2.3
        2         4        2.1
        3         2       1.15
        4         5        1.5
        5         6          2
        6         7        1.1
        7         8       1.85
        8         1        2.7
Model has been successfully processed
```

Resultado do algoritmo para o problema proposto