Relatório do Problema de Múltiplos Caixeiros Viajantes

Leonardo Ângelo Figueiredo Muniz
cc23323@g.unicamp.br

Gabriel Aparecido Mantovani
cc23317@g.unicamp.br

Relatório do trabalho para resolver o problema de múltiplos caixeiros viajantes conhecido também como MCV. O objetivo deste problema é achar o melhor e menor caminho para viajarmos por todas as cidades e por fim voltar para cidade inicial, sem repetir nenhuma cidade.

Cotuca | Colégio Técnico de Campinas,
Abril 2024

1 Introdução

O Problema dos Múltiplos Caixeiros Viajantes (MCV) é um desafio clássico em otimização combinatória, onde múltiplos caixeiros viajantes devem visitar um conjunto de cidades uma vez, sem repetições, e retornar ao ponto de partida, minimizando a distância total percorrida. Ele também é classificado como NP. Problemas NP não tem respostas simples com resultados únicos, há diversas formas diferentes de resolver problemas NP como este do caixeiro viajante. Todas as resoluções desse problema são heurísticas que geram resultados aproximados da solução ideal.

2 Heurística construtiva

Um método comum para resolver o problema é o do vizinho mais próximo, onde começamos em uma cidade e visitamos a mais próxima, repetindo esse processo até retornarmos à cidade inicial. No entanto, este método pode ser limitado, pois foca excessivamente em partes específicas da rota, resultando em caminhos mais longos que passam por várias cidades. Para facilitar a resolução do problema, utilizamos as bibliotecas matplotlib e sklearn para criar um plano cartesiano com as distâncias, tornando mais fácil identificar possíveis problemas em nossa solução.

Como estamos especificamente resolvendo o problema de Múltiplos Caixeiros Viajante (MCV), vamos fazer com que o número de cidades que cada caixeiro viajante tem que passar seja de (Cidades/NCaixeiros), ou seja, se tivermos 8 cidades e 4 caixeiros, cada um terá que passar por 2 cidades, no caso de termos um número quebrado de cidades por caixeiro, o que vamos fazer é o seguinte. Caso o número decimal seja menor que N.5, todos os

caixeiros viajarão por N cidades arredondado para baixo, com exceção do último que viajará por N arredondado para cima. Caso o número decimal seja de exatamente N.5, metade dos caixeiros viajarão por N arredondado para baixo e outra metade por N arredondado para cima. No caso do número decimal ser maior que N.5, o primeiro caixeiro apenas viajará por N arredondado para baixo de cidades, e o restante por N arredondado para cima.

Exemplos:

(Número de cidades / Número de caixeiros) = Cidades por caixeiro -> Lista Final

54/12 = 4.5 -> [4, 4, 4, 4, 4, 4, 5, 5, 5, 5, 5, 5]

27/6 = 4.5 -> [4, 4, 4, 5, 5, 5]

16/3 = 5.33333 -> [5, 5, 6]

15/3 = 5 -> [5, 5, 5]

14/3 = 4.66666 -> [4, 5, 5]

13/3 = 4.33333 -> [4, 4, 5]

Criamos uma heuristica que é uma modificação da heurística do vizinho mais proximo juntamente com uma chamada Three OPT Swap. Começamos o percurso pegando as duas cidades mais próximas da cidade inicial, uma dessas cidades será reservada como sendo a ultima cidade que viajaremos. O restante do percurso segue a mesma lógica do algoritimo de vizinho mais proximo, pegamos a cidade mais proxima da atual e passamos por ela.

Depois disso usamos a técnica do Three OPT Swap, que basicamente consiste em pegar aleatoriamente três rotas diferentes e troca-lás de lugar, caso a distância seja menor que a original, aceitamos essa nova rota. Por cima

disso tudo existe uma váriavel de "temperatura"que possibilita fazer trocas que pioram a distância total. Isso acontece porque em certas ocasiões ficamos "travados"em uma miníma local, e as vezes para que possamos sair dessa minima local precisamos piorar um pouco a rota para depois possivelmente melhorar a distância. A temperatura vai abaixando aos poucos, e conforme a temperatura chega perto de zero vamos aceitando menos rotas arriscadas.

Este algorítimo roda N+1 vezes, N sendo o número de tours. Na primeira vez usamos esse algoritimo para trocar as rotas entre diversos caixeiros viajantes diferentes, depois desse processo melhoramos individualmente a rota de cada caixeiro viajante.

3 Experimentos computacionais

Para facilitar o processo dos experimentos, criamos um programa que pelo nome pega o arquivo que tem as coordenadas das cidades, e depois transforma essas coordenadas em uma matriz de distâncias automaticamente. Também criamos um programa que pega a matriz de distâncias e as rotas, e cria um plano cartesiano a partir disso, isso nos possibilita vizualizar as rotas.

Vamos utilizar as cidades e a quantidade de caixeiros do arquivo **mTSP-n31-m3** para mostrar o código em ação:

1. Começamos da cidade zero, e pegamos as duas cidades mais proximas, uma delas nós reservamos para ser a penúltima.

2. Para cada caixeiro conseguimos um resultado:
   Tour 1: [0, 28, 24, 21, 2, 12, 27, 15, 16, 30, 4, 0]
   Tour 2: [0, 23, 7, 20, 17, 9, 8, 29, 19, 1, 5, 0]

Tour 3: [0, 25, 13, 14, 6, 3, 18, 10, 11, 26, 31, 22, 0] Distância: 6519m

3. Agora usamos o método de Three OPT Swap em todas as rotas, para fazer isso unimos todas as rotas em uma lista só.

4. Depois disso melhoramos a rota de cada tour individualmente, e como resultado temos:

   Tour 1: [0, 5, 21, 2, 12, 27, 15, 16, 30, 24, 28, 0]

   Tour 2: [0, 7, 23, 20, 9, 17, 8, 29, 19, 25, 4, 0]

   Tour 3: [0, 1, 6, 14, 13, 3, 18, 10, 31, 26, 11, 22, 0]

   Distância melhorada: 5991m

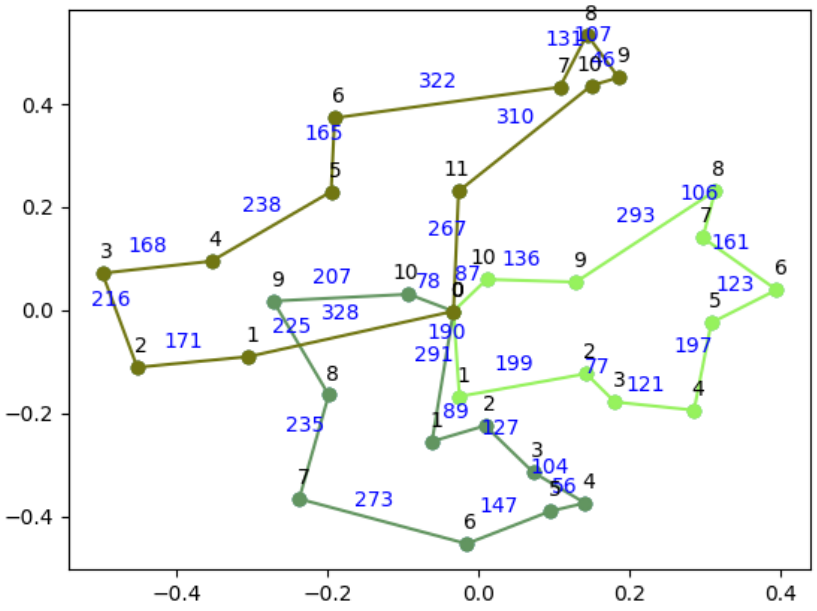

Figura 1: Rota gerada do arquivo "mTSP-n31-m3

3.1 Resultados

Os resultados podem variar já que o processo é bem aleatório.

Instâncias	*N*	*M*	*K*	*f(h)*	*fx*	*GAP%*
mTSP-n13-m1	13	1	13	**3024**	**3024**	0%
mTSP-n17-m1	17	1	17	4068	3948	3%
mTSP-n19-m1	19	1	19	4318	4218	2.3%
mTSP-n31-m3	31	3	11	5991	5841	2.5%
mTSP-n47-m3	47	3	16	7283	6477	11.7%
mTSP-n59-m3	59	3	20	7682	6786	12.3%
mTSP-n71-m5	71	5	15	9720	8618	12%
mTSP-n83-m5	83	5	17	9684	9246	4.6%
mTSP-n91-m5	91	5	19	10493	9586	9%

4 Referências

Noel O'Boyle. Convert distance matrix to 2D projection with Python. 13 de Janeiro de 2014, de https://baoilleach.blogspot.com/2014/01/convert-distance-matrix-to-2d.html.

n Sanity. Traveling Salesman Problem Visualization. 18 de Agosto de 2013, de https://www.youtube.com/watch?v=SC5CX8drAtU.

Wikipedia contributors. (2024, March 30). 2-opt. In Wikipedia, The Free Encyclopedia. 10 de Abril de 2024, de https://en.wikipedia.org/wiki/2-opt