UNIVERSIDADE FEDERAL DE VIÇOSA
DEPARTAMENTO DE INFORMÁTICA
INF332 – Projeto e Análise de Algoritmos

Trabalho Computacional

Problema do Caixeiro Viajante (PCV):

O problema do caixeiro viajante (PCV), do inglês *Travelling Salesman Problem* (TSP), é um problema clássico de Otimização Combinatória formulado da seguinte maneira. Existem n cidades (locais ou pontos) a serem visitadas. Para cada par de cidades (i, j), existe um caminho (estrada) com distância $D[i, j]$ conhecida, ou seja, tem-se um grafo completo com n vértices cuja matriz de adjacências é a matriz de distâncias D. Um caixeiro viajante (por exemplo, um entregador de encomendas) pretende visitar todas as cidades uma única vez, retornando à cidade origem ao final do percurso.

O objetivo é encontrar o trajeto mais curto (rota com a menor distância possível), que passa por todas as cidades uma única vez, começando e terminando na cidade origem. Essa rota é conhecida como *ciclo Hamiltoniano* de menor distância. Na Figura 1b mostra-se uma rota formada com 20 cidades: [1, 3, 18, 9, 14, 2, 8, 13, 15, 10, 5, 16, 11, 7, 19, 6, 4, 17, 20, 12]. Note que, para fechar o ciclo, da última cidade visitada deve-se retornar à cidade origem 1. A *distância* (distancia total) correspondente a essa rota é calculada da seguinte maneira:

Distancia = $D[\mathbf{1}, 3] + D[3, 18] + D[18, 9] + \ldots + D[17, 20] + D[20, 12] + D[12, \mathbf{1}]$.

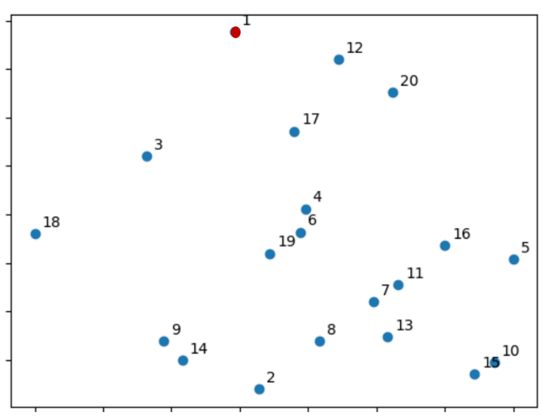

Figura 1a: Conjunto de n pontos (cidades).

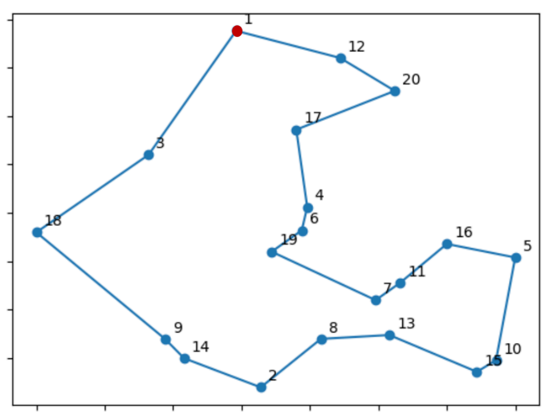

Figura 1b: Rota (ciclo Hamiltoniano) obtida.

Problema do Caixeiro Viajante Multi-objetivo (PCV-MO):

Neste problema, numa viagem da cidade i para uma cidade j, consideram-se a distância percorrida ($D[i,j]$) e o tempo gasto ($T[i,j]$). Nem sempre, a menor distância é percorrida no menor tempo (por exemplo, estradas com muitas curvas sinuosas e asfalto ruim).

O objetivo do PCV-MO é determinar o trajeto ou rota de tal forma que a viagem seja realizada percorrendo a menor distância e no menor tempo. Neste problema tem-se dois objetivos: minimizar distância e minimizar tempo (otimização multi-objetivo).

Geralmente, não existe uma solução (rota) que tenha a menor distância e o menor tempo. Por exemplo, na Figura 1b tem-se a rota com menor distância total. Esta rota não necessariamente será percorrida no menor tempo. Os objetivos são conflitantes, ou seja, a solução que tem a menor distância, geralmente, tem o maior tempo, e vice-versa.

Um problema de otimização multi-objetivo (por exemplo, com dois objetivos a serem minimizados) tem como solução um **conjunto de soluções minimais**.

Para o PCV-MO, uma solução é definida pela *rota ou trajeto* da viagem e pelos valores dos objetivos (*distância*, *tempo*) correspondente a esta rota. Assim, cada rota R terá associado um ponto

$p(R) = (d_R, t_R)$, onde d_R e t_R são os valores da *distância* e *tempo*, respectivamente. Para comparar a qualidade das soluções é usada a seguinte definição:

Uma solução A com valores (d_A, t_A) *domina* ("é melhor" que) outra solução B com valores (d_B, t_B), $A \neq B$, se: $(d_A \leq d_B$ e $t_A \leq t_B)$ e $(d_A < d_B$ ou $t_A < t_B)$.
Resolver o PCV-MO, consiste em determinar o **conjunto de soluções minimais**, dentre todas as soluções (rotas) possíveis. Ou seja, deve ser determinado o conjunto de soluções que não sejam dominados por nenhuma solução existente. Na Figura 2 ilustrasse os **valores dos objetivos** de um conjunto de 23 soluções minimais. Note que o ponto (**70.509**, 108.115) corresponde à rota com a **menor distância**, no entanto ela tem o maior tempo. O ponto (123.345, **61.697**) corresponde à rota com o menor **tempo**, no entanto ela tem a maior distância.

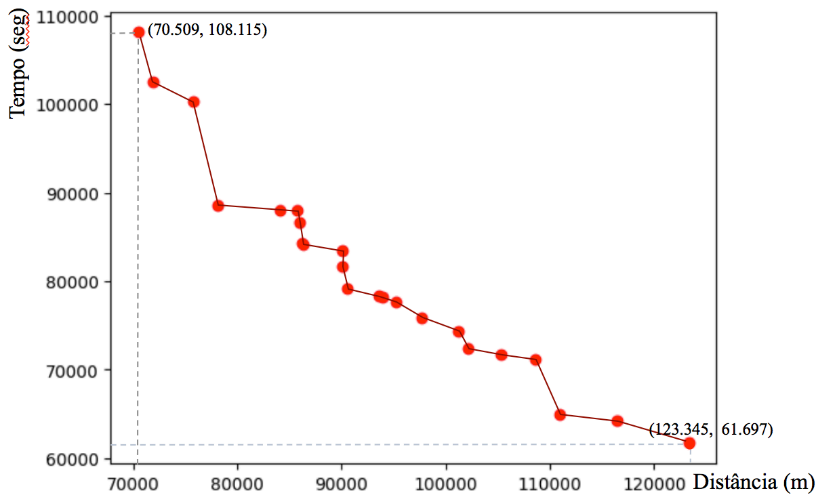

Figura 2. Curva com os valores dos objetivos (distância, tempo) de um conjunto com 23 soluções minimais.

Implementação Computacional

Para determinar uma solução aproximada (um conjunto de soluções minimais) do Problema do Caixeiro Viajante Multi-objetivo, com n cidades, implemente o seguinte método.

Método Heurístico Multi-objetivo (MHMO):

1) Gere n soluções utilizando o algoritmo guloso do Vizinho Mais Próximo (VMP) e minimizando apenas *distância*. Para cada solução a ser gerada utilize uma cidade origem diferente (considere todas as cidades i para ser origem, tal que i =1, 2, ..., n). Após obter uma solução (rota) com *distância* otimizada, deve-se calcular o *tempo* da rota. Todas as soluções geradas devem ser armazenadas em um conjunto **S**. O algoritmo do VMP é apresentado abaixo.
2) Gere n soluções utilizando o algoritmo VMP e minimizando apenas o *tempo*. Para cada solução a ser gerada utilize uma cidade origem diferente (considere todas as cidades i para ser origem, tal que i =1, 2, ..., n). Após obter uma solução (rota) com *tempo* otimizado, deve-se calcular a *distância* da rota. Todas as soluções geradas devem ser armazenadas no conjunto **S**.
3) Gere $6n$ soluções utilizando o algoritmo VMP_*D*_*T*, minimizando *distância* ou *tempo*. Para determinar uma solução, a cidade origem deve ser escolhida de forma aleatória (entre 1 e n). Todas as soluções geradas devem ser armazenadas no conjunto **S**. O algoritmo do VMP_*D*_*T* é apresentado abaixo.
4) Gere $2n$ soluções (rotas) de forma totalmente aleatória (algoritmo aleatório). Para cada rota R gerada calcule a *distância* e o *tempo*: (d_R, t_R). Todas as soluções geradas devem ser armazenadas no conjunto **S**.
5) Das $10n$ soluções obtidas (conjunto **S**), determine o **conjunto das soluções minimais** (conjunto **M**). Use um algoritmo baseado em **Divisão e Conquista** para determinar o conjunto **M**.

6) Plote os **valores dos objetivos** (d_R, t_R) das soluções minimais obtidas no conjunto **M** (gere gráficos similares à Figura 2). Sugere-se que os pontos (d_R, t_R) do conjunto **M** sejam ordenados em ordem crescente de *distância*.

Algoritmo Guloso VMP:

*#Algoritmo para obter **uma solução** minimizando apenas Distância.*

<u>Entrada</u>: Matriz de distâncias D,
Lista de índices das cidades I = {1, 2, ..., *n*},
Cidade inicial (origem) a.

Distancia = 0, *ini* = *a*.
Rota = {*a*} *#insere na rota o índice da cidade inicial*
I = I – {*a*} *#remove cidade "a" da lista I*
Enquanto tamanho(I) > 0:
    Encontre a cidade $k \in$ I mais próxima de *a* (*k* ={ *j*, tal que *D*[*a* , *j*] é mínimo, $\forall j \in$ I})
    Rota = *Rota* + {*k*} *#a cidade "k" é inserida logo após a última cidade adicionada na rota*
    I = I – {*k*} *#remove cidade "k" da lista I*
    Distancia = *Distancia* + *D*[*a* , *k*]
    a = *k* *# k será a última cidade inserida na rota*
Distancia = *Distancia* + *D*[*a* , *ini*] *# fecha o ciclo*

Tempo = CalculaTempoDaRota(*Rota*)
Retorne (*Rota*, *Distancia, Tempo*)

Algoritmo Guloso VMP_*D_T*:

*#Algoritmo do VMP para obter **uma solução** do PCV minimizando Distância e Tempo.*

<u>Entrada</u>: Matriz de distâncias D e matriz de tempos T,
Lista de índices das cidades I = {1, 2, ..., *n*}

Distancia = *Tempo* = 0,
ini = *a* = random(1, *n*) *#Escolhe aleatoriamente uma cidade inicial*
Rota = {*a*} *#insere na rota o índice da cidade inicial*
I = I – {*a*} *#remove cidade "a" da lista I*
Enquanto tamanho(I) > 0:
  r = random(0, 1)
  Se *r* == 0:
    Encontre a cidade $k \in$ I mais próxima de *a* (*k* ={ *j*, tal que *D*[*a* , *j*] é mínimo, $\forall j \in$ I})
  Senão:
    Encontre a cidade $k \in$ I com menor tempo de *a* (*k* ={ *j*, tal que *T*[*a* , *j*] é mínimo, $\forall j \in$ I})
  Rota = *Rota* + {*k*} *#a cidade "k" é inserida logo após a última cidade adicionada na rota*
  I = I – {*k*} *#remove cidade "a" da lista I*
  Distancia = *Distancia* + *D*[*a* , *k*]
  Tempo = *Tempo* + *T*[*a* , *k*]
  a = *k* *# k será a última cidade inserida na rota*

Distancia = *Distancia* + *D*[*a* , *ini*] *# fecha o ci*clo
Tempo = *Tempo* + *T*[*a* , *ini*]
Retorne (*Rota*, *Distancia, Tempo*)

Dados de Entrada:

Como dados de entrada são fornecidos o número n de cidades e as matrizes de *distâncias* e *tempos* entre cada par de cidades (matrizes D e T).

Serão resolvidas 7 instâncias do problema, 6 com 100 cidades e 1 com 150 cidades. As entradas das instâncias (matrizes D e T) estão, respectivamente, nos seguintes arquivos:

kroAxkroB100.txt, kroAxkroC100.txt, kroAxkroD100.txt, kroBxkroC100.txt, kroBxkroD100.txt, kroCxkroD100.txt e kroAxkroB150.txt

Apresentação de Resultados

Para cada instância (arquivo), execute o método MHMO. Imprima como saída, os valores dos objetivos (d_R, t_R) das soluções minimais obtidas no conjunto **M** e o tempo computacional (em ms) gasto pelo método. Também, plote os pontos (d_R, t_R) e apresente as curvas obtidas.

Apresente os resultados obtidos no formato das Tabela 1 e 2, e apresenta os gráficos: Fig1, Fig2,..., Fig7. Finalmente, escreva sobre as conclusões do trabalho.

Tabela 1.

Instâncias	**nº de soluções mínimas**	**Ponto** (d_R, t_R) **com menor** d_R	**Ponto** (d_R, t_R) **com menor** t_R	**Tempo de CPU total gasto pelo método MHMO (em *ms*)**
kroAxkroB100				
kroAxkroC100				
kroAxkroD100				
kroBxkroC100				
kroBxkroD100				
kroCxkroD100				
kroAxkroB150				

Tabela 2.

Conjunto de pontos (d_R, t_R) das soluções minimais obtidas (ordenadas em ordem crescente de d_R)						
kroAxkroB100	**kroAxkroC100**	**kroAxkroD100**	**kroBxkroC100**	**kroBxkroD100**	**kroCxkroD100**	**kroAxkroB150**

Figuras: Curvas dos pontos (d_R, t_R)

Fig1. kroAxkroB100	Fig2. kroAxkroC100
Fig3. kroAxkroD100	Fig4. kroBxkroC100

Etc.

Conclusões do Trabalho:

Escreva comentários sobre: o problema abordado, análise dos resultados obtidos e desempenho dos algoritmos utilizados para gerar soluções do problema.

As implementações dos algoritmos podem ser feitas nas linguagens C++ ou Python. Apresentar os resultados (tabelas, gráficos e conclusões) em ***um único documento*** *(arquivo) PDF. Anexe/cole no final do documento o código do programa (escreva o código do seu programa da forma mais clara possível). Também envie um arquivo .zip contendo os arquivos fonte do programa.*
Este trabalho pode ser feito em DUPLA.