Universidade
Estadual de Londrina

ATIVIDADE RESULTADOS

Londrina

2023

VINICIUS DOURADO SILVA

ATIVIDADE RESULTADOS

Atividades TCC

Docente: Ernesto Ferreyra Ramirez

Sumário

1 METODOLOGIA

A fim de realizar comparações, dois métodos para solução do Problema do Caixeiro viajante (PCV) foram montadas, sendo uma solução por meio da busca exaustiva e outra por meio do algoritmo genético. Todos os códigos abaixo foram executados em uma máquina com as seguinte configurações:

- **Processador:** Processador Intel(R) Core(TM) i3-4330 CPU @ 3.50GHz 3.50 GHz
- **Memória RAM:** 10,0 GB
- **Tipo de Sistema:** Windows 10 Pro, Sistema operacional de 64 bits, processador baseado em x64

1.1 Busca exaustiva

A maneira mais eficiente de encontrar uma solução para o Problema do Caixeiro Viajante (PCV) é através da avaliação de todas as possibilidades de rota. Portanto, o código necessário deve ser o mais simples possível. Nesse sentido, a metodologia proposta envolve a utilização de uma matriz de distâncias das cidades selecionadas para os testes, além das permutações das cidades, que representam as diferentes rotas a serem consideradas. Com essas informações disponíveis, é possível calcular o custo de cada rota e, em seguida, identificar a rota de menor e maior custo, bem como determinar o tempo necessário para a realização da simulação.

```
%clc
clear all
tic

% Dados Cidades N = 10
```

```
% 1  - UEL, Londrina
% 2  - Ibiporã
% 3  - Maringá
% 4  - Cascavel
% 5  - Paranagua
% 6  - Pérola
% 7  - Palmas
% 8  - Brasilandia do Sul
% 9  - Sengés
% 10 - Santo Antonio do Sudoeste

DistMatriz = [0      21     98     378    485    309    525    319    308    534;
              21     0      111    391    498    322    538    332    285    547;
              98     111    0      283    523    212    508    222    399    439;
              378    391    283    0      603    171    310    106    556    163;
              485    498    523    603    0      693    450    697    371    657;
              309    322    212    171    693    0      497    63     638    327;
              525    538    508    310    450    497    0      409    493    221;
              319    332    222    106    697    63     409    0      648    262;
              308    285    399    556    371    638    493    648    0      622;
              534    547    439    163    657    327    221    262    622    0];

VetorIndi = [1 2 3 4 5 6 7 8 9 10];

%permutação dos indices
Pi = perms(VetorIndi);

% Variavel da Distancia
distanciamenor  = inf;
distanciamaior = 0;

% Calculo da rota
VetorDistancias = zeros(length(Pi),1);
Indice_Rota_menor = 0;
Indice_Rota_maior = 0;

for i = 1:length(Pi)
    rota = Pi(i, :); % Armazena a rota atual
    distancia   = 0;

    for j = 1:length(rota)
        Cidade_partida = rota(j);
        Cidade_chegada = rota(mod(j, length(rota)) + 1);
```

```
        distancia = distancia + DistMatriz(Cidade_partida, Cidade_chegada);
    end

    % Armazenador de distancias
    VetorDistancias(i,1) = distancia;

    % Comparadores
    if distancia < distanciamenor
        distanciamenor = distancia;
        Indice_Rota_menor = i;
        Rota_menor = rota;
    end
    if distancia > distanciamaior
        distanciamaior = distancia;
        Indice_Rota_maior = i;
        Rota_maior = rota;
    end

end

% Exibi as distâncias
disp(['A distância total da menor rota é e a rota é a:']);
distanciamenor
Indice_Rota_menor
Rota_menor
disp(['A distância total da maior rota ée a rota é a:']);
distanciamaior
Indice_Rota_maior
Rota_maior


toc
```

1.2 Algoritmo Genético

O algoritmo genético proposto busca solucionar o Problema do Caixeiro Viajante (PCV) aplicando a técnica de otimização baseada em processos evolutivos. O código abaixo descreve a implementação dessa metodologia.

Na etapa de inicialização, é necessário definir as cidades a serem testadas, bem

como os parâmetros necessários. Isso inclui determinar o tamanho da população inicial e o número máximo de gerações. Para cada cromossomo na população inicial, um vetor de 11 posições é gerado, com os índices das cidades distribuídos aleatoriamente.

Em seguida, para o processo de evolução, em cada população, o cálculo do percurso total é realizado. Os dois melhores indivíduos são selecionados como pais da próxima geração. Cada pai gera dois filhos através da permutação de duas posições. Posteriormente, ocorre a atualização da população.

Como resultado, o código proposto foi desenvolvido para fins comparativos com o método de força bruta. Ele produz observações semelhantes, como o tempo de execução e o menor percurso encontrado ao longo das gerações.

```
    %clc
clear all
tic

% Dados Cidades N = 10
% 1  - UEL, Londrina
% 2  - Ibiporã              ; Raio de 20km
% 3  - Maringá              ; Raio de 40km
% 4  - Cascavel             ; Raio de 80km
% 5  - Paranagua            ; Raio maior que 160km
% 6  - Pérola
% 7  - Palmas
% 8  - Brasilandia do Sul
% 9  - Sengés
% 10 - Santo Antonio do Sudoeste

DistMatriz = [0     21    98    378   485   309   525   319   308   534;
              21    0     111   391   498   322   538   332   285   547;
              98    111   0     283   523   212   508   222   399   439;
              378   391   283   0     603   171   310   106   556   163;
              485   498   523   603   0     693   450   697   371   657;
              309   322   212   171   693   0     497   63    638   327;
              525   538   508   310   450   497   0     409   493   221;
              319   332   222   106   697   63    409   0     648   262;
              308   285   399   556   371   638   493   648   0     622;
              534   547   439   163   657   327   221   262   622   0];
```

```
numCidades = 10;

%indices
tam_Pop_ini = 10;               %Tamanho População Inicial
tam_gera    = 100;             %Nmr de gerações

%Geracao da Populacao Inicial
for k = 1:tam_Pop_ini

    %Gera um vetor de 11 posicoes com numeros de 2 a 10 (cidades) distribuidos aleatoriamente

    [a b] = sort(rand(1,9));
    b = b + 1;

    %Cromossomo
    Populacao(k, :) = [1, b, 1];

end


for g = 1:tam_gera

    %Calcula o percurso total de cada vetor solucao
    for k = 1:size(Populacao,1),
        Percurso(k, g) = Calc_Dist(DistMatriz, Populacao(k,:));
    end

    %Escolhe os mais aptos
    clear a b;
    [a b] = sort(Percurso(:,g));

    Pai = Populacao(b(1,1), :);
    Mae = Populacao(b(2,1), :);




    %Cada Genitor vai gerar 2 filhos por permuta
    %Sorteia e permuta as duas posicoes do Pai
    for k = 1:4,
        clear a b;
        [a b] = sort(rand(1,9));
```

```
        b = b + 1;

        FilhosPai(k, :) = Pai;
        FilhosPai(k, b(1, 1)) = Pai(1, b(1, 2));
        FilhosPai(k, b(1, 2)) = Pai(1, b(1, 1));
    end

    %Sorteia e permuta as duas posicoes da Mae
    for k = 1:4,
        clear a b;
        [a b] = sort(rand(1,9));
        b = b + 1;

        FilhosMae(k, :) = Mae;
        FilhosMae(k, b(1, 1)) = Mae(1, b(1, 2));
        FilhosMae(k, b(1, 2)) = Mae(1, b(1, 1));
    end


    %Atualiza a Populacao
    Populacao = [Pai; Mae; FilhosPai; FilhosMae];

end

toc
plot(min(Percurso, [], 1));
```

2 RESULTADOS E DISCUSSÕES

Em um primeiro momento, para ambos os casos foram escolhidas cidades testes arbitrariamente, bem como o tamanho inicial da população e o numero de gerações para o algoritmo genético, sendo 10 e 100 respectivamente. Além de tirar uma média das 10 soluções de cada método para que se tenha um padrão comparativo em cada resultado.

Tabela 1: Comparação do tempo de execução, em segunfos, dos algoritmos em diferentes cenários

	Busca Exaustiva			Algoritmo Genético		
	5 Cidades	10 Cidades	15 Cidades	5 Cidades	10 Cidades	15 Cidades
Matlab	0,0505	5,401	*	0,04002	0,04778	0,04268
Python**	0,00399	113,169	*	0,145	0,1818	0,1312

Legenda:

- *: Não foi possível executar devido a grandes requisitos de memória RAM.
- **: Resultados podem variar com o estado da maquina a ser executado.

Ao analisar os resultados, observa-se que o algoritmo genético, tanto em Python quanto em MATLAB, apresenta uma execução significativamente mais rápida. Além disso, o desempenho do algoritmo mantém um padrão consistente à medida que o número de cidades aumenta.

Ao observar os resultados em relação ao percurso total encontrado, podemos notar que a busca exaustiva é capaz de encontrar o melhor resultado. No entanto, à medida que o número de cidades aumenta, a execução do código se torna mais desafiadora. É importante ressaltar que o valor encontrado para a menor rota através do AG não necessariamente é o menor possível, uma vez que foi calculada uma média das execuções.

Tabela 2: Comparação do menor percurso, em km, dos algoritmos em diferentes cenários

	Busca Exaustiva			Algoritmo Genético		
	5 Cidades	10 Cidades	15 Cidades	5 Cidades	10 Cidades	15 Cidades
Matlab	1503	1990	*	1503	2117	2619
Python	1503	1990	*	1503	2045,8	2500

Legenda:

- *: Não foi possível executar devido a grandes requisitos de memória RAM.

Tabela 3: Comparação do maior percurso, em km, na execução dos algoritmos em diferentes cenários

	Busca Exaustiva			Algoritmo Genético		
	5 Cidades	10 Cidades	15 Cidades	5 Cidades	10 Cidades	15 Cidades
Matlab	1888	5313	*	1529,6	3116,9	4597,3
Python	1888	5313	*	1780,4	3584,2	5106,2

Legenda:

- *: Não foi possível executar devido a grandes requisitos de memória RAM.

Referências