Relatório de Execução de Código - Preparação para Algoritmo Genético (TSP)

Desenvolvido por: Maria Eduarda S. e Fredson Arthur

Objetivo

O código Python apresentado visa configurar as estruturas de dados e funções básicas (carregamento de matriz de distâncias, validação de rota e cálculo de fitness) necessárias para um problema do Caixeiro Viajante (TSP) a ser resolvido com um Algoritmo Genético (AG).

1. Código Fonte

O programa tenta carregar uma matriz de distâncias de uma URL externa e, em caso de falha, utiliza uma matriz de fallback de 5 cidades (N=5). Define as funções para validação de rota e cálculo de distância (fitness) e utiliza uma classe Individuo para encapsular o cromossomo e seu desempenho.

```python
import numpy as np
import requests
import warnings # Usaremos warnings para suprimir a mensagem de aviso

# 1. Instância do TSP (Matriz de Distâncias)
URL_TSP_DATA =
"https://gist.github.com/rodrigoclira/1cc3dfc603740decb4269096aa7ac122/raw
/"

def carregar_matriz_distancias(url):
    """
    Baixa a matriz de distâncias no formato JSON/Array.
    """
    # ⚠️ Desabilita os avisos (warnings) gerados por 'verify=False'
    warnings.filterwarnings('ignore', message='Unverified HTTPS request')

    try:
        # AQUI ESTÁ A CORREÇÃO: Usar verify=False
        response = requests.get(url, verify=False)
```

```
        response.raise_for_status()

        # O restante da função permanece igual
        matriz_distancias_list = response.json()
        num_cidades = len(matriz_distancias_list)

        print(f"Dados TSP carregados com sucesso: {num_cidades} cidades.")

        return np.array(matriz_distancias_list)

    except requests.exceptions.RequestException as e:
        # Se a conexão falhar mesmo com verify=False (improvável), usa a
fallback
        print(f"Erro ao carregar os dados TSP (após ignorar SSL): {e}")
        print("Usando matriz de fallback (5 cidades: A, B, C, D, E).")
        return np.array([
            [ 0, 20, 42, 35, 10],  # A
            [20,  0, 30, 34, 25],  # B
            [42, 30,  0, 12, 42],  # C
            [35, 34, 12,  0, 24],  # D
            [10, 25, 42, 24,  0]   # E
        ])

# Carrega a matriz global
DISTANCIA_MATRIX = carregar_matriz_distancias(URL_TSP_DATA)

# Mapeamento para cidades (se for a matriz de 5x5 do fallback, senão usa
índices)
# Como a matriz carregada tem 10 cidades, vamos trabalhar apenas com
índices (0 a 9)
NUM_CIDADES = DISTANCIA_MATRIX.shape[0]

#
================================================================================
====
# Funções de Validação e Fitness
#
================================================================================
====

def validar_rota(cromossomo):
    """
    Valida se a rota (cromossomo) é válida para o TSP:
    1. Não pode ter cidades repetidas.
    2. Deve conter todas as cidades intermediárias (cidades 1 até N-1).
```

```
    O cromossomo é uma lista/array/tupla dos índices das cidades
intermediárias.
    Ex: [1, 3, 2] para um problema de 4 cidades (0 -> 1 -> 3 -> 2 -> 0)
    """

    # 1. Verificar o tamanho (deve ter N-1 cidades)
    cidades_no_cromossomo = len(cromossomo)
    cidades_esperadas = NUM_CIDADES - 1

    if cidades_no_cromossomo != cidades_esperadas:
        return False, f"Tamanho incorreto: Esperado {cidades_esperadas},
Encontrado {cidades_no_cromossomo}."

    # 2. Verificar se há cidades repetidas ou faltando
    cidades_esperadas_indices = set(range(1, NUM_CIDADES)) # Cidades de 1
até N-1 (Cid 0 é fixada)
    cidades_no_cromossomo_set = set(cromossomo)

    if cidades_no_cromossomo_set != cidades_esperadas_indices:
        return False, "Cidades repetidas ou faltando."

    return True, "Rota válida."


def calcular_fitness_distancia(cromossomo):
    """
    Calcula a distância total da rota (Fitness no contexto de AG para
minimização).
    Quanto menor a distância, melhor o fitness.

    Rota completa: Cidade 0 -> Cromossomo[0] -> ... -> Cromossomo[N-2] ->
Cidade 0
    """

    # A rota sempre começa e termina na cidade 0 (índice)
    rota_completa = [0] + list(cromossomo) + [0]

    distancia_total = 0

    # Percorre de par em par
    for i in range(len(rota_completa) - 1):
        cidade_atual = rota_completa[i]
        proxima_cidade = rota_completa[i+1]

        # Acessa a distância na matriz global
        distancia_total += DISTANCIA_MATRIX[cidade_atual, proxima_cidade]
```

```
    return distancia_total

#
================================================================================
====
# Classe Indivíduo para Organização
#
================================================================================
====

class Individuo:
    def __init__(self, cromossomo):
        """
        O cromossomo é a lista/array das cidades intermediárias (B, C, D,
E, etc.).
        Ex: [1, 3, 2, 4]
        """
        self.cromossomo = np.array(cromossomo)

        # Validação
        self.valido, self.mensagem_validacao =
validar_rota(self.cromossomo)

        # Fitness (só calcula se for válido, embora AGs muitas vezes punam
rotas inválidas)
        if self.valido:
            self.distancia_total =
calcular_fitness_distancia(self.cromossomo)
            # Para o AG, o Fitness real é o inverso, pois queremos
MAXIMIZAR o 'bom'
            self.fitness = 1 / self.distancia_total
        else:
            self.distancia_total = float('inf') # Distância máxima
            self.fitness = 0.0 # Fitness mínimo

    def get_rota_completa(self):
        """Retorna a rota completa incluindo a cidade inicial/final
(0)."""
        return [0] + list(self.cromossomo) + [0]

    def __repr__(self):
        status = "Válido" if self.valido else f"INVÁLIDO:
{self.mensagem_validacao}"
        return (f"Cromossomo: {self.cromossomo} | Rota:
{self.get_rota_completa()} | "
                f"Distância: {self.distancia_total:.2f} km | Fitness
(1/D): {self.fitness:.5f} | Status: {status}")
```

```
#
================================================================================
====
# Teste e Validação
#
================================================================================
====

if __name__ == '__main__':

    print("\n--- TESTE DE PREPARAÇÃO TSP/AG ---")
    print(f"Matriz de Distâncias carregada. Número de cidades:
{NUM_CIDADES} (índices 0 a {NUM_CIDADES-1})")

    # --- CORREÇÃO AQUI ---
    # Se N=5, o cromossomo deve ter 4 elementos: cidades 1, 2, 3, 4.
    cidades_intermediarias = list(range(1, NUM_CIDADES)) # Ex: [1, 2, 3,
4]

    print(f"Cidades Intermediárias esperadas: {cidades_intermediarias}")
    # --- FIM DA CORREÇÃO ---

    print("\n--- TESTE DE VALIDAÇÃO DE ROTAS ---")

    # 1. Rota Válida (exemplo: uma rota ordenada)
    rota_valida_exemplo = cidades_intermediarias
    valido, msg = validar_rota(rota_valida_exemplo)
    print(f"Rota {rota_valida_exemplo}: {valido} - {msg}")

    # 2. Rota Inválida (cidade 1 repetida, uma cidade faltando)
    # Exemplo para 5 cidades (N=5): [1, 1, 3, 4] -> 2 está faltando, 1
está repetida
    rota_invalida_repetida = [1, 1, 3, 4]
    valido, msg = validar_rota(rota_invalida_repetida)
    print(f"Rota {rota_invalida_repetida}: {valido} - {msg}")

    # 3. Rota Inválida (tamanho incorreto)
    rota_invalida_tamanho = cidades_intermediarias[:-1] # Remove a última
cidade. Ex: [1, 2, 3]
    valido, msg = validar_rota(rota_invalida_tamanho)
    print(f"Rota {rota_invalida_tamanho}: {valido} - {msg}")

    print("\n--- TESTE DE CÁLCULO DE FITNESS (DISTÂNCIA) ---")

    # Rota válida de exemplo (cromossomo ordenado)
    ind_valido = Individuo(rota_valida_exemplo)
```

```
    print(ind_valido)

    # Rota inválida (o cálculo do fitness é tratado na classe)
    ind_invalido = Individuo(rota_invalida_repetida)
    print(ind_invalido)

    # Exemplo de cálculo passo-a-passo (0 -> 1):
    print("\nCálculo Exemplo (0 -> 1):")
    dist_0_1 = DISTANCIA_MATRIX[0, 1]

    # Para 5 cidades, a última cidade intermediária é N-1 = 4.
    # Então, a última aresta da rota ordenada é (N-1) -> 0, ou seja, 4 ->
0.
    dist_4_0 = DISTANCIA_MATRIX[4, 0]

    print(f"Distância 0 -> 1: {dist_0_1}")
    print(f"Distância 4 -> 0: {dist_4_0}")
    print(f"Distância Total da rota ordenada (0-1-2-3-4-0):
{ind_valido.distancia_total:.2f} km")
```

---

2. Saída do Código

A saída reflete a falha no carregamento dos dados da URL e, consequentemente, o uso da matriz de fallback de 5 cidades. Em seguida, os testes de validação de rota e cálculo de fitness são realizados com base nesta matriz de 5 x 5.

```
Erro ao carregar os dados TSP (após ignorar SSL): Expecting value: line 1
column 1 (char 0)
Usando matriz de fallback (5 cidades: A, B, C, D, E).

--- TESTE DE PREPARAÇÃO TSP/AG ---
Matriz de Distâncias carregada. Número de cidades: 5 (índices 0 a 4)
Cidades Intermediárias esperadas: [1, 2, 3, 4]

--- TESTE DE VALIDAÇÃO DE ROTAS ---
Rota [1, 2, 3, 4]: True - Rota válida.
Rota [1, 1, 3, 4]: False - Cidades repetidas ou faltando.
Rota [1, 2, 3]: False - Tamanho incorreto: Esperado 4, Encontrado 3.

--- TESTE DE CÁLCULO DE FITNESS (DISTÂNCIA) ---
```

```
Cromossomo: [1 2 3 4] | Rota: [0, np.int64(1), np.int64(2), np.int64(3),
np.int64(4), 0] | Distância: 96.00 km | Fitness (1/D): 0.01042 | Status:
Válido
Cromossomo: [1 1 3 4] | Rota: [0, np.int64(1), np.int64(1), np.int64(3),
np.int64(4), 0] | Distância: inf km | Fitness (1/D): 0.00000 | Status:
INVÁLIDO: Cidades repetidas ou faltando.

Cálculo Exemplo (0 -> 1):
Distância 0 -> 1: 20
Distância 4 -> 0: 10
Distância Total da rota ordenada (0-1-2-3-4-0): 96.00 km
```

3. Análise da Saída

1. Carregamento de Dados:

* O código tentou carregar a matriz da URL, mas falhou, retornando uma exceção de quebra de formato JSON (`Expecting value: line 1 column 1 (char 0)`).
* Como resultado, a matriz de **fallback de 5 cidades** foi utilizada, definindo `NUM_CIDADES = 5`.
* As cidades intermediárias esperadas para um cromossomo de 5 cidades são os índices `[1, 2, 3, 4]`.

2. Validação de Rotas:

* A rota ordenada `[1, 2, 3, 4]` foi corretamente identificada como **válida**.
* A rota `[1, 1, 3, 4]` foi corretamente identificada como **inválida** por ter "Cidades repetidas ou faltando" (o '1' se repete e o '2' falta).
* A rota `[1, 2, 3]` foi corretamente identificada como **inválida** devido ao "Tamanho incorreto" (esperado 4, encontrado 3).

3. Cálculo de Fitness:

Indivíduo Válido: A rota completa [0, 1, 2, 3, 4, 0] resultou em uma Distância Total de 96.00 km.

Cálculo da Distância (Soma dos elementos na matriz de 5x5):

D(0 \to 1) + D(1 \to 2) + D(2 \to 3) + D(3 \to 4) + D(4 \to 0)

20 + 30 + 12 + 24 + 10 = 96

O Fitness, calculado como o inverso da distância (1/D), é 1/96 \approx 0.01042.

Indivíduo Inválido: A rota inválida tem a distancia_total definida como inf e o fitness como 0.00000, penalizando corretamente o indivíduo para que ele não seja selecionado em um AG de maximização.

Cálculo Exemplo: Os valores de distância entre os pares (0 \to 1) e (4 \to 0) foram verificados na matriz, confirmando a correta indexação.