

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

Desenvolvido por: Maria Eduarda S. e Fredson Arthur

Objetivo

O presente relatório descreve a execução e validação das estruturas de dados e funções básicas em Python para a solução do Problema do Caixeiro Viajante (TSP) usando um Algoritmo Genético (AG). O foco inicial é o carregamento da instância (matriz de distâncias), a validação da rota e o cálculo da função de aptidão (*fitness*).

1. Código Fonte (Estrutura e Funcionalidades)

O código implementa três componentes principais para configurar a base do AG:

1. Carregamento da Matriz de Distâncias (carregar_matriz_distancias): Tenta baixar os dados de uma URL externa. Em caso de falha (erro de conexão ou formato de dados, como observado na saída), utiliza uma matriz de fallback simétrica de 5 x 5 (para 5 cidades).
2. Validação de Rota (validar_rota): Garante que o cromossomo (a sequência de cidades intermediárias) represente uma rota TSP válida. Verifica se o tamanho do cromossomo é N-1 (onde N é o número total de cidades) e assegura que todas as cidades intermediárias (índices de 1 a N-1) estejam presentes e sem repetição.
3. Cálculo de Fitness (calcular_fitness_distancia) e Classe Indivíduo: Calcula a distância total da rota, considerando o retorno à cidade inicial (índice 0). A classe Individuo encapsula o cromossomo, a distância total e o valor de fitness, que é definido como o inverso da distância (1/D), transformando o problema de minimização em um problema de maximização. Rotas inválidas recebem fitness 0.

O trecho de código abaixo resume as funções principais e a classe Individuo:

```
```python
Trecho simplificado do código:
A função 'carregar_matriz_distancias' tenta a URL, mas falha e usa a matriz de 5x5.

def validar_rota(cromossomo):
 # Verifica o tamanho (deve ser N-1) e se há cidades repetidas ou faltando.
 # ... retorna True/False e mensagem.

def calcular_fitness_distancia(cromossomo):
 # Rota completa: [0] + list(cromossomo) + [0]
 # Soma as distâncias na matriz global para a rota completa.
 # ... retorna a distancia_total.

class Individuo:
```

```

def __init__(self, cromossomo):
 self.cromossomo = np.array(cromossomo)
 self.valido, self.mensagem_validacao = validar_rota(self.cromossomo)

 if self.valido:
 self.distancia_total = calcular_fitness_distancia(self.cromossomo)
 self.fitness = 1 / self.distancia_total # Maximização
 else:
 self.distancia_total = float('inf')
 self.fitness = 0.0 # Penalização
...

```

## 2. Saída da Execução e Análise

A execução do código demonstrou uma falha no carregamento dos dados da URL, que retornou 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 o número total de cidades, N, como 5. As cidades intermediárias esperadas para o cromossomo são os índices [1, 2, 3, 4].

A seguir, a análise detalhada dos testes realizados:

### Validação de Rotas

O sistema de validação funcionou corretamente, diferenciando rotas válidas de inválidas, conforme exemplificado:

Rota Válida: O cromossomo ordenado [1, 2, 3, 4] foi identificado como válido.

Rota Inválida (Repetição/Falta): A rota [1, 1, 3, 4] foi corretamente identificada como inválida com a mensagem "Cidades repetidas ou faltando" (o '1' se repete e o '2' está ausente).

Rota Inválida (Tamanho): A rota [1, 2, 3] foi corretamente identificada como inválida devido ao "Tamanho incorreto: Esperado 4, Encontrado 3".

### Cálculo de Fitness

O cálculo da aptidão foi testado em dois cenários:

1. Indivíduo Válido: A rota completa da solução ordenada é 0 \to 1 \to 2 \to 3 \to 4 \to 0.

Distância Total: A soma das distâncias na matriz de 5 x 5 (fallback) foi verificada como  $D(0\text{to } 1) + D(1\text{to } 2) + D(2\text{to } 3) + D(3\text{to } 4) + D(4\text{to } 0) = 20 + 30 + 12 + 24 + 10 = 96.00\text{km}$ .

Fitness: O valor de fitness, sendo 1/Distância, resultou em = 0.01042.

2. Indivíduo Inválido: Para a rota inválida ([1, 1, 3, 4]), a classe Individuo corretamente atribuiu à distancia\_total o valor inf (infinito) e ao fitness o valor 0.00000, penalizando adequadamente o indivíduo para que não seja selecionado em um AG de maximização.

## Conclusão

O código de preparação para o Algoritmo Genético no problema TSP foi executado com sucesso e suas funções básicas foram validadas. Apesar da falha no carregamento da URL, a matriz de fallback garantiu a disponibilidade de uma instância para testes.

As implementações de validação de rota e cálculo de fitness são robustas e corretas: elas garantem que apenas rotas válidas sejam avaliadas e que o problema de minimização de distância seja corretamente traduzido para a maximização de aptidão ( $\text{Fitness} = 1/\text{Distância}$ ). As estruturas estão prontas para a próxima fase do Algoritmo Genético, que incluirá a geração da população inicial, operadores de cruzamento e mutação, e o mecanismo de seleção.