



UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL REI
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO - DCOMP
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
UNIDADE CURRICULAR: GRAFOS

Trabalho prático:
Algoritmo Genético para o Problema
do Caixeiro Viajante

BERNARDO MAIA DETOMI - 202050054
LEONARDO PATIAS JOHANSSON - 202050055

Documentação do Algoritmo Genético para o Problema do Caixeiro Viajante (TSP)

Conceitos Principais

1. Problema do Caixeiro Viajante (TSP):

O **TSP** é definido como: dado um conjunto de cidades e as distâncias entre cada par de cidades, encontre o menor caminho possível que percorra todas as cidades exatamente uma vez e retorne à cidade inicial.

2. Algoritmo Genético (AG):

Um **Algoritmo Genético** é inspirado nos processos de seleção natural e evolução biológica. O AG usado aqui evolui uma população de soluções candidatas (rotas) através de seleção, crossover e mutação para encontrar a melhor rota.

3. Passos do AG:

- **Inicialização:** Criar uma população inicial de soluções aleatórias.
- **Seleção:** Escolher soluções boas (rotas com menor distância) para se reproduzirem.
- **Cruzamento:** Combinar duas soluções para criar uma nova solução (filhos).
- **Mutação:** Fazer pequenas mudanças aleatórias em uma solução.
- **Avaliação:** Calcular a distância total de uma rota.
- **Critério de Parada:** O algoritmo para quando atinge um número máximo de gerações ou não encontra melhorias após um certo número de gerações (estagnação).

- Parâmetros

O código suporta a variação dos seguintes parâmetros:

- Tamanho da população: Ajustável nas listas de experimentos no arquivo [main.py](#). Padrão em 50, 100 e 200.
- Taxa de cruzamento: Padrão em 0.7, 0.8 e 0.9.
- Taxa de mutação: Variável entre 0.01, 0.05 e 0.1.

- Critérios de Parada

O algoritmo possui três critérios de parada:

1. Número máximo de gerações (definido como 500 por padrão).
2. Estagnação: Se a solução não melhorar após um número pré-definido de gerações.
3. Convergência: Quando todas as rotas na população convergem para uma solução semelhante.

Este repositório contém a implementação de um **Algoritmo Genético (AG)** para resolver o **Problema do Caixeiro Viajante (TSP - Traveling Salesman Problem)**. O código utiliza matrizes de distâncias fornecidas em arquivos de texto para encontrar a rota ótima.

Repositório: (<https://github.com/BernardoDetomi/Algoritmo-genetico-para-solucao-do-Problema-do-Caixeiro-Viajante>).






Estrutura de Arquivos

- `main.py`: 🚀 Arquivo principal que executa o algoritmo genético, incluindo a leitura da matriz de distâncias e experimentos fatoriais.
- `algoritmo_genetico.py`: 💡 Implementação do núcleo do algoritmo genético, com seleção de pais, crossover, mutação, e critérios de parada por estagnação.
- `utils.py`: 🛠️ Funções utilitárias como cálculo de distância, criação de rotas, mutação e crossover.
- `matriz_distancias4.txt`, `matriz_distancias8.txt`, `matriz_distancias15.txt`: 📊 Arquivos de teste contendo matrizes de distâncias para diferentes números de cidades (4, 8 e 15 cidades, respectivamente).



Pasta testes

Dentro da pasta `testes`, você encontrará subpastas organizadas pelas seguintes instâncias de matriz de distâncias:

- `matriz_distancias4`:  Contém testes com matrizes de distâncias de 4 cidades.
- `matriz_distancias8`:  Contém testes com matrizes de distâncias de 8 cidades.
- `matriz_distancias15`:  Contém testes com matrizes de distâncias de 15 cidades.

Cada subpasta armazena os resultados dos experimentos para a respectiva matriz.



Como Executar o Código

1. Escolha a matriz de distâncias desejada (por exemplo, `matriz_distancias4.txt`, `matriz_distancias8.txt`, ou `matriz_distancias15.txt`).
2. Salve a matriz de distâncias no arquivo `matriz_distancias.txt` ou ajuste o caminho no código.

Execute o programa com o seguinte comando:

```
python main.py
```



Ajuste de Parâmetros

No arquivo `main.py`, você pode ajustar os seguintes parâmetros para os experimentos:

- **Tamanho da população:** Controle o número de indivíduos na população.
- **Taxa de mutação:** Defina a probabilidade de ocorrer mutação em uma rota.
- **Taxa de crossover:** Defina a probabilidade de ocorrer crossover entre dois pais.



Exemplo de Resultados

Durante a execução, o algoritmo irá exibir no console a evolução da população e os melhores resultados encontrados:

População: 100, Mutação: 0.01, Cruzamento: 0.8

Geração 1: Melhor distância = 128

Geração 2: Melhor distância = 118

...

Melhor rota: [0, 3, 2, 1, 0]

Os resultados finais e as melhores rotas encontradas serão armazenados em arquivos na pasta `testes`.

Código

Main.py

```
1 import os
2 from datetime import datetime
3 from algoritmo_genetico import algoritmo_genetico
4 import numpy as np
5
6 def ler_matriz_distancias_txt(nome_arquivo):
7     with open(nome_arquivo) as f:
8         matriz = [list(map(int, linha.strip().split())) for linha in f]
9     return np.array(matriz)
10
11 # Função para salvar os resultados em arquivos txt na pasta 'testes'
12 def salvar_resultados(tamanho_pop, taxa_mut, taxa_cruz, melhor_distancia, melhor_rota):
13     # Criar a pasta 'testes' se não existir
14     if not os.path.exists('testes'):
15         os.makedirs('testes')
16
17     # Gerar um nome único para o arquivo
18     timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
19     nome_arquivo = f"testes/teste_{tamanho_pop}_{taxa_mut}_{taxa_cruz}_{timestamp}.txt"
20
21     # Escrever os resultados no arquivo
22     with open(nome_arquivo, 'w') as f:
23         f.write(f"Populacao: {tamanho_pop}\n")
24         f.write(f"Taxa de Mutacao: {taxa_mut}\n")
25         f.write(f"Taxa de Cruzamento: {taxa_cruz}\n")
26         f.write(f"Melhor Distancia: {melhor_distancia}\n")
27         f.write(f"Melhor Rota: {melhor_rota}\n")
28
29     print(f"Resultados salvos em: {nome_arquivo}")
30
31 if __name__ == "__main__":
32     matriz_distancias = ler_matriz_distancias_txt("matriz_distancias8.txt")
33
34     tamanhos_populacao = [50, 100, 200]
35     taxas_mutacao = [0.01, 0.05, 0.1]
36     taxas_cruzamento = [0.7, 0.8, 0.9]
37
38     for tamanho_pop in tamanhos_populacao:
39         for taxa_mut in taxas_mutacao:
40             for taxa_cruz in taxas_cruzamento:
41                 print(f"\nPopulação: {tamanho_pop}, Mutação: {taxa_mut}, Cruzamento: {taxa_cruz}")
42                 melhor_rota, melhor_distancia = algoritmo_genetico(
43                     matriz_distancias,
44                     tamanho_populacao=tamanho_pop,
45                     taxa_mutacao=taxa_mut,
46                     geracoes=500
47                 )
48                 print(f"Melhor distância: {melhor_distancia}")
49                 print(f"Melhor rota: {melhor_rota}")
50
51                 # Salvar os resultados no arquivo
52                 salvar_resultados(tamanho_pop, taxa_mut, taxa_cruz, melhor_distancia, melhor_rota)
```

algoritmo_genetico.py

```
1 import random
2 from utils import criar_populacao, calcular_distancia, selecionar_pais, cruzamento, mutacao
3
4 def algoritmo_genetico(matriz_distancias, tamanho_populacao=100, geracoes=500, taxa_mutacao=0.01, tamanho_torneio=5, max_estagnacao=50):
5     num_cidades = len(matriz_distancias)
6     populacao = criar_populacao(tamanho_populacao, num_cidades)
7     melhor_fitness_geral = float('inf')
8     estagnacao = 0
9
10    for geracao in range(geracoes):
11        fitness = [calcular_distancia(rota, matriz_distancias) for rota in populacao]
12        nova_populacao = []
13
14        # Seleção e reprodução
15        for _ in range(tamanho_populacao // 2):
16            pais = selecionar_pais(populacao, fitness, tamanho_torneio)
17            filho1 = cruzamento(pais[0], pais[1])
18            filho2 = cruzamento(pais[1], pais[0])
19            nova_populacao.append(mutacao(filho1, taxa_mutacao))
20            nova_populacao.append(mutacao(filho2, taxa_mutacao))
21
22        populacao = nova_populacao
23
24        melhor_rota = min(populacao, key=lambda rota: calcular_distancia(rota, matriz_distancias))
25        melhor_distancia = calcular_distancia(melhor_rota, matriz_distancias)
26
27        if melhor_distancia < melhor_fitness_geral:
28            melhor_fitness_geral = melhor_distancia
29            estagnacao = 0
30        else:
31            estagnacao += 1
32
33        if estagnacao >= max_estagnacao:
34            print(f"Parada por estagnação na geração {geracao + 1}")
35            break
36
37        print(f"Geração {geracao + 1}: Melhor distância = {melhor_distancia}")
38
39    return melhor_rota, melhor_distancia
```

utils.py

```
1 import random
2
3 def calcular_distancia(rota, matriz_distancias):
4     distancia_total = 0
5     for i in range(len(rota) - 1):
6         distancia_total += matriz_distancias[rota[i]][rota[i + 1]]
7     distancia_total += matriz_distancias[rota[-1]][rota[0]]
8     return distancia_total
9
10 def criar_rota(num_cidades):
11     rota = list(range(num_cidades))
12     random.shuffle(rota)
13     return rota
14
15 def criar_populacao(tamanho_populacao, num_cidades):
16     return [criar_rota(num_cidades) for _ in range(tamanho_populacao)]
17
18 def selecionar_pais(populacao, fitness, tamanho_torneio):
19     pais = []
20     for _ in range(2):
21         torneio = random.sample(list(enumerate(fitness)), tamanho_torneio)
22         melhor = min(torneio, key=lambda x: x[1])
23         pais.append(populacao[melhor[0]])
24     return pais
25
26 def cruzamento(pai1, pai2):
27     tamanho = len(pai1)
28     filho = [None] * tamanho
29     inicio, fim = sorted(random.sample(range(tamanho), 2))
30     filho[inicio:fim] = pai1[inicio:fim]
31     posicao_filho = fim
32     for cidade in pai2:
33         if cidade not in filho:
34             if posicao_filho == tamanho:
35                 posicao_filho = 0
36             filho[posicao_filho] = cidade
37             posicao_filho += 1
38     return filho
39
40 def mutacao(rota, taxa_mutacao):
41     if random.random() < taxa_mutacao:
42         i, j = random.sample(range(len(rota)), 2)
43         rota[i], rota[j] = rota[j], rota[i]
44     return rota
```

Alguns resultados:

matriz_distancias4.txt

```
Parada por estagnação na geração 51  
Melhor distância: 21  
Melhor rota: [0, 2, 1, 3]  
Resultados salvos em: testes/teste_200_0.1_0.9_20240905_111024.txt
```

matriz_distancias8.txt

```
Parada por estagnação na geração 53  
Melhor distância: 21  
Melhor rota: [0, 1, 7, 2, 5, 4, 3, 6]  
Resultados salvos em: testes/teste_200_0.1_0.9_20240905_111118.txt
```

matriz_distancias15.txt

```
Parada por estagnação na geração 69  
Melhor distância: 291  
Melhor rota: [13, 9, 7, 5, 3, 10, 0, 12, 1, 14, 8, 4, 6, 2, 11]  
Resultados salvos em: testes/teste_200_0.1_0.9_20240905_111201.txt
```



Referências

WIKIPEDIA CONTRIBUTORS. Genetic algorithm. Disponível em:

<https://en.wikipedia.org/wiki/Genetic_algorithm>.

WIKIPEDIA CONTRIBUTORS. Travelling salesman problem. Disponível

em: <https://en.wikipedia.org/wiki/Travelling_salesman_problem>.

Observações:

- Esta implementação é modular e fácil de modificar. Você pode ajustar facilmente os parâmetros ou trocar as funções de mutação/crossover para experimentar diferentes variações do AG.
- As instâncias de teste com diferentes tamanhos de matrizes permitem realizar comparações de desempenho entre as execuções.