Problema do Caixeiro Viajante Usando Algoritmo Genético

Este código implementa um Algoritmo Genético (GA) para resolver o problema do Caixeiro Viajante (TSP). O objetivo do TSP é encontrar o caminho mais curto que visita todas as cidades em um conjunto e retorna à cidade inicial. O GA é uma técnica de otimização metaheurística inspirada na seleção natural, onde os indivíduos (soluções) com melhor aptidão (menor custo) têm mais chances de serem selecionados para reprodução e gerar novas soluções.

Documentação do Código:

Criação de Indivíduos:

* A classe `GA` representa um indivíduo do GA.
* O método `***criar_caminho(n)***` gera um caminho aleatório de tamanho 'n'.
* O método `***calcular_fitness()***` calcula o valor de aptidão (custo total do caminho) do indivíduo.

Operadores Genéticos:

- O método `***cruzamento(outro)***` realiza o cruzamento entre dois indivíduos, gerando um novo indivíduo.
- O método `***cruzamento_multiponto(outro, pontos)***` realiza o cruzamento multiponto entre dois indivíduos, gerando um novo indivíduo.
- O método `***two_opt()***` aplica o heurístico 2-opt para melhorar o indivíduo.
- O método `***two_opt_ganho(i, j)***` calcula o ganho de aplicar o 2-opt entre as cidades i e j.
- O método `***mutacao(taxa)***` aplica mutação ao indivíduo com uma taxa especificada, trocando cidades aleatórias.

➢ O método `***mutacao_inversao(taxa)***` aplica mutação por inversão ao indivíduo, invertendo a ordem de um segmento do caminho.

Leitura de Dados:

➢ O método `**ler_arquivo(caminho_arquivo)**` lê os dados do TSP de um arquivo de texto contendo as coordenadas das cidades.

Visualização:

➢ O método `***plot_convergencia()***` gera um gráfico que mostra a convergência do GA, ou seja, como a melhor aptidão (menor custo) muda ao longo das gerações.

Evolução da População:

➢ O método `***calcular_fitness_paralelo(individuo)***` calcula a aptidão de um indivíduo de forma paralela usando multiprocessing.

➢ O método `***evoluir(populacao, geracoes)***` evolui a população de indivíduos por um número especificado de gerações:

- Seleciona os melhores indivíduos para reprodução.
- Aplica cruzamento e mutação para gerar novos indivíduos.
- Avalia a aptidão de todos os indivíduos.
- Seleciona os melhores indivíduos para a próxima geração.
- Registra a melhor aptidão em cada geração.
- Retorna a população final e o número de gerações.

Estrutura do Código:

O código está organizado em classes e funções:

Classe `GA`: Representa um indivíduo do GA, com atributos para o caminho (gene), custo total (preco), aptidão (fit) e lista tabu para evitar ciclos repetidos na mutação.

Funções Auxiliares:

- `***criar_caminho(n)***`: Gera um caminho aleatório de tamanho n.
- `***calcular_distancia(cidade1, cidade2, matriz_distancias)***`: Calcula a distância entre duas cidades.
- `***ler_arquivo(caminho_arquivo)***`: Lê os dados do TSP de um arquivo de texto.
- `***plot_convergencia()***`: Gera um gráfico que mostra a convergência do GA.
- `***calcular_fitness_paralelo(individuo)***`: Calcula a aptidão de um indivíduo de forma paralela usando multiprocessing.

Evolução da População:

- `***evoluir(populacao, geracoes)***`: Evolui a população de indivíduos por um número especificado de gerações.

Resultados:

Utilizando o algoritmo para cerca de 38 cidades, obteve-se o seguinte resultado:

```
Geracao:  99  Melhor caminho: [33, 38, 37, 35, 32, 30, 29, 21, 14, 10, 1, 2, 4, 3, 5, 6, 7, 8, 9, 12, 11, 19, 18, 17, 16, 13, 15, 20, 23, 26, 25, 22, 24, 28, 2
7, 31, 36, 34]
 Custo:  6659

O tempo de execução foi:  13.762014389038086  segundos
```

Para comparação, a fonte: https://www.math.uwaterloo.ca/tsp/world/djtour.html

Utilizando o algoritmo para cerca de 194 cidades, obteve-se o seguinte resultado:

```
Geracao:  99  Melhor caminho: [175, 184, 177, 181, 178, 180, 170, 167, 168, 165, 159, 158, 162, 155, 151, 147, 152, 141, 144, 150
, 146, 149, 156, 145, 140, 137, 134, 132, 126, 125, 127, 130, 111, 104, 101, 99, 94, 89, 90, 98, 86, 85, 65, 20, 63, 36, 59, 62,
, 74, 69, 60, 57, 45, 37, 27, 22, 29, 28, 33, 18, 21, 24, 26, 17, 7, 11, 14, 25, 23, 13, 16, 8, 6, 1, 4, 2, 3, 5, 9, 10, 12, 15,
, 42, 50, 49, 55, 54, 46, 48, 52, 53, 56, 58, 43, 40, 34, 39, 47, 51, 61, 67, 73, 66, 68, 64, 70, 77, 84, 81, 79, 83, 88, 92, 97,
, 113, 114, 119, 122, 118, 106, 105, 107, 108, 100, 110, 112, 115, 116, 117, 121, 120, 123, 124, 128, 133, 135, 129, 131, 136, 14
3, 188, 189, 191, 192, 190, 187, 183, 186, 194, 182, 176, 169, 161, 163, 164, 172, 179, 174, 173]
 Custo:  9354

O tempo de execução foi:  292.3051645755768  segundos
```

Para comparação, a fonte: https://www.math.uwaterloo.ca/tsp/world/qatour.html