Prática 4 - Bioinspirados - Problema do Caixeiro Viajante

Rodrigo José Zonzin Esteves

9 de maio de 2025

1 Introdução

Algoritmos Genéticos (AGs) são técnicas de otimização inspiradas em processos biológicos, amplamente aplicadas em áreas como otimização estrutural, planejamento urbano e redes moleculares. Além de funções em domínio real, também são úteis em problemas combinatórios, como o do caixeiro viajante (TSP, na sigla em inglês). Este trabalho visa implementar, de forma simplificada, um AG para resolver o TSP.

2 Metodologia

O código está estruturado em torno de duas classes principais: *Gene*, que representa uma solução candidata (ou indivíduo), e *AG*, que implementa o funcionamento do algoritmo genético propriamente dito. armazena uma permutação dos vértices do grafo de distâncias. Essa permutação define a ordem de visita às cidades. A classe fornece também métodos auxiliares como copy, para duplicar o indivíduo, e conversão para string, para representação textual (Código 1).

```
class Gene:
    def __init__(self, alelos):
        self.alelos = alelos

    def __str__(self):
        return str(self.alelos)

class AG:
    def __init__(self, matriz_dist, pop_size=200, taxa_mutacao=0.01,
   taxa_cruzamento=1.0, metodo_selecao="torneio"):
        self.matriz_dist = matriz_dist
        self.pop_size = pop_size
        self.gene_size = matriz_dist.shape[0]
        self.taxa_mutacao = taxa_mutacao
        self.taxa_cruzamento = taxa_cruzamento
```

```
15          self.metodo_selecao = metodo_selecao
16          self.pop = self._gerar_populacao()
17
18      def _gerar_populacao(self):
19          return [Gene(random.sample(range(self.gene_size), self.
    gene_size)) for _ in range(self.pop_size)]
20
21      def fitness(self, gene):
22          caminho = gene.alelos
23          dist = sum(self.matriz_dist[caminho[i]][caminho[i + 1]] for i
    in range(len(caminho) - 1))
24          #add distancia do ultimo ate o primeiro
25          dist += self.matriz_dist[caminho[-1]][caminho[0]]
26          return dist
```

Listing 1: Implementação do Gene e do AG

O método *_gerar_populacao* da classe AG cria uma população inicial de indivíduos, cada um contendo uma permutação aleatória dos nós. Essa população é armazenada na lista self.pop.

O método *fitness* calcula a aptidão de um indivíduo com base na distância total do percurso representado por sua permutação (Equação 1). Essa distância é obtida somando-se os pesos dos arcos entre os vértices consecutivos, considerando ainda o retorno ao ponto inicial para fechar o ciclo.

$$f(\pi) = \sum_{i=1}^{n-1} \rho(\pi(i), \pi(i+1)) + \rho(\pi(n), \pi(1)) \tag{1}$$

Além disso, dois operadores de cruzamento são implementados: Order Crossover (OX) e Cycle Crossover (CX). No OX, um segmento do primeiro pai é copiado diretamente para o filho. Os genes restantes são inseridos na ordem em que aparecem no segundo pai, excluindo elementos já presentes no vetor. Já no CX, ciclos de posição entre os pais são identificados e usados para determinar quais genes cada filho herda, garantindo a formação de permutações válidas sem repetições.

Foram realizados testes para avaliar o desempenho do AG em diferentes cenários, com variações em parâmetros como método de seleção de pais, tamanho da população, taxa de mutação, taxa de cruzamento e presença de elitismo. Os parâmetros padrão utilizados nos testes foram os seiugntes:

- Método de seleção: Torneio ou Roleta
- Tamanho da população: 100 ou 200 indivíduos
- Taxa de mutação: 0.01
- Taxa de cruzamento: 1.0
- Elitismo: sim
- Número de gerações: 500
- Entradas: 15 cidades

Quanto ao método de seleção, foram realizados testes comparando abordagens por Torneio e Roleta, mantendo os demais parâmetros inalterados. Sabe-se que a presença do elitismo proporciona maior estabilidade do resultado ao longo das gerações, de forma que, na sua ausência, os resultados tendem a apresentar flutuações significativas. Dessa forma, optou-se por analisar apenas a variação dos parâmetros diante do elitismo.

3 Resultados

O plot do fitness ao longo das gerações indica que o 500 gerações foi capaz de obter o mínimo local para um dos métodos – torneio (Figura 1).

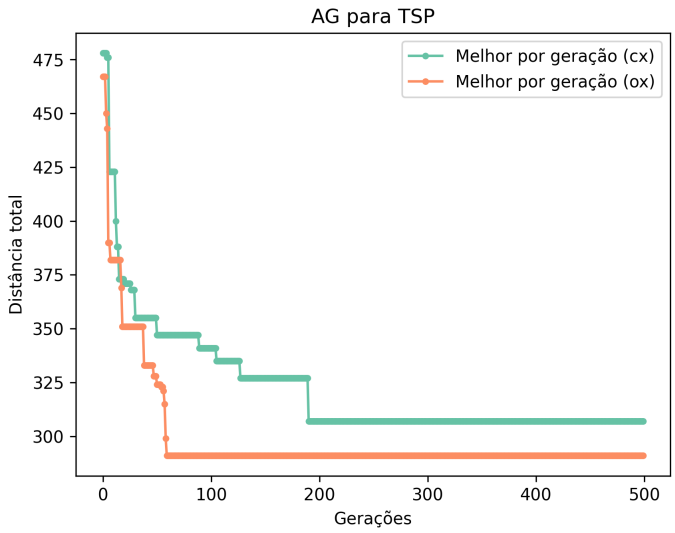

Figura 1: Fitness oa longo das gerações

Para complementar essa análise, a Figura 2 apresenta a frequência de obtenção do mínimo global por geração.

Os resultados indicaram que o método do Torneio apresentou desempenho superior, tanto na qualidade do melhor indivíduo encontrado quanto no número de gerações necessárias para encontrá-lo (Figura 3).

Já a comparação entre os métodos de cruzamento não foi conclusiva. O método OX apresentou uma média de desmpenho melhor (301), enquanto o método CX apresentou um resultado médio ligeiramente pior (304). Cabe ressaltar, no entanto, que o desvio-padrão das execuções do CX foram maiores, o que demanda uma análise de significância estatística mais aprofundada (Figura 4).

A Figura 5 apresenta a comparação global obtida. Como se nota, o método do torneio foi mais eficiente em ambos os casos. Ainda sim, todas as abordagens aproximaram-se do mínimo global conhecido: 291.

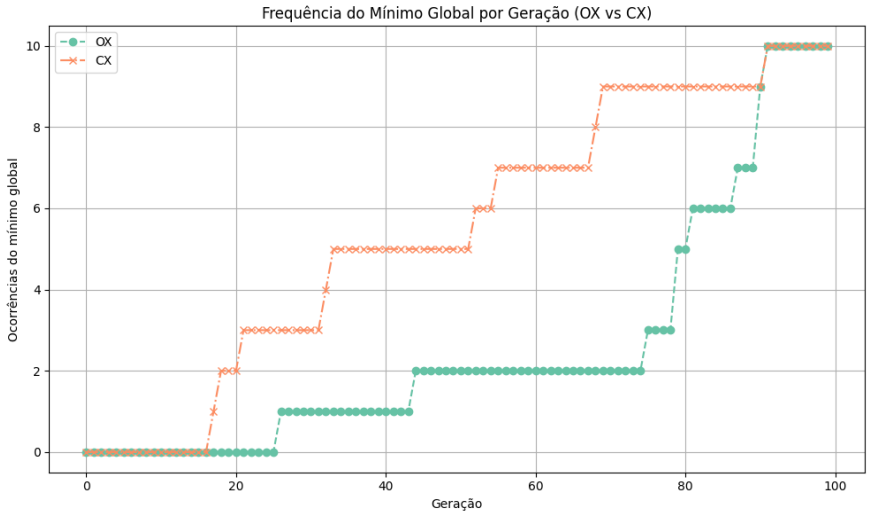

Figura 2: Obtenção do mínimo global

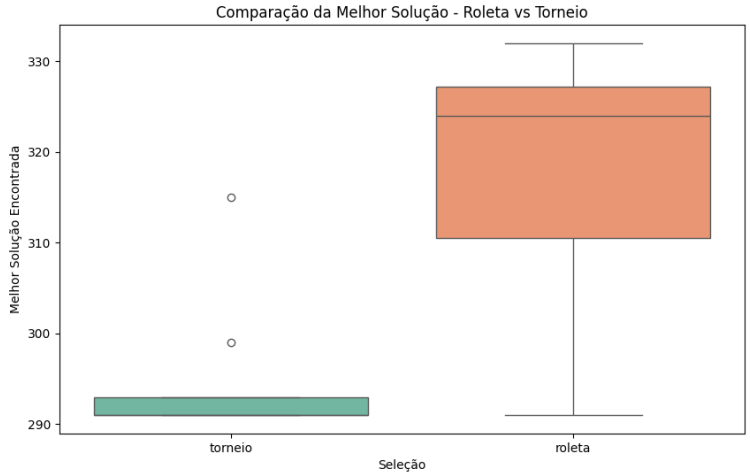

Figura 3: Comparação entre Torneio e Roleta

4 Conclusão

A implementação do AG para a resolução do Problema do Caixeiro Viajante demonstrou-se eficaz na obtenção de soluções próximas ao mínimo global conhecido. O método de seleção por torneio destacou-se por apresentar melhores resultados em termos de qualidade da solução e estabilidade ao longo das gerações, superando a abordagem por roleta. Além disso, o operador de cruzamento OX mostrou-se superior. De modo geral, os resultados obtidos validam a eficiência dos AGs na resolução do TSP, resultado já conhecido da literatura em otimização. Trabalhos futuros podem explorar outras va-

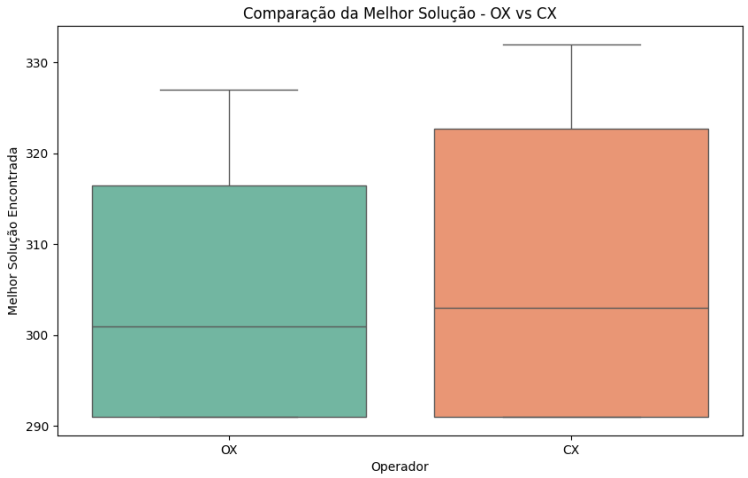

Figura 4: Comparação entre ps métodos OX e CX

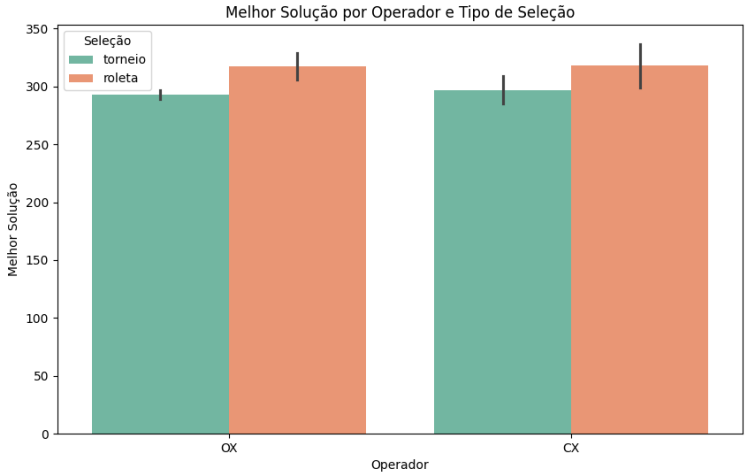

Figura 5: Operador vs e tipo de solução

riantes de operadores genéticos, bem como incorporar técnicas híbridas que combinem busca local com algoritmos evolutivos.