

# **Relatório Técnico: Aplicação de Algoritmos Genéticos no Problema do Caixeiro Viajante (TSP)**

**Disciplina:** Inteligência Computacional

**Assunto:** Análise empírica de operadores genéticos em otimização combinatória.

Tauá - CE  
25/01/2026

Saul Alves Martins de Oliveira

# 1. Introdução

O Problema do Caixeiro Viajante (TSP - *Traveling Salesman Problem*) é um problema clássico de otimização combinatória e pertence à classe de problemas NP-Difíceis. O objetivo consiste em encontrar a rota de menor custo que visite um conjunto de cidades exatamente uma vez e retorne ao ponto de partida.

A complexidade do TSP reside no crescimento factorial do espaço de busca ( $N!/2$  para um grafo simétrico). Métodos exatos tornam-se inviáveis computacionalmente para um número elevado de cidades. Este relatório analisa a aplicação de Algoritmos Genéticos (AGs) como uma heurística para encontrar soluções subótimas de alta qualidade em tempo hábil, focando especificamente no impacto dos operadores de cruzamento e mutação na convergência e na diversidade da população.

## 2. Metodologia e Implementação

Para resolver o TSP, foi implementado um Algoritmo Genético em linguagem Python, estruturado para permitir a parametrização explícita de taxas de evolução.

### 2.1 Representação (Codificação)

Diferente de problemas numéricos, o TSP exige uma restrição de permutação (não pode haver cidades repetidas ou faltantes).

- **Cromossomo:** Vetor de inteiros representando a ordem de visita.
- **Exemplo:** [2, 0, 3, 1, 4] para 5 cidades.
- **Espaço de Busca:** Permutações válidas de \$0\$ a \$N-1\$.

### 2.2 Função de Aptidão (Fitness)

O objetivo é minimizar a distância total Euclidiana. Como o AG padrão maximiza a aptidão, utilizou-se a inversão do custo:

$\text{Fitness}(x) = \frac{1}{\sum_{i=0}^{N-1} d(c_i, c_{i+1}) + d(c_N, c_0)}$   
Onde  $d(a, b)$  é a distância Euclidiana entre as cidades  $a$  e  $b$ .

### 2.3 Operadores Genéticos

Devido à restrição de permutação, operadores clássicos (como *One-Point Crossover*) gerariam soluções inválidas (cidades duplicadas). Foram implementados operadores específicos:

1. **Seleção por Torneio (k=3):** Escolhe-se  $k$  indivíduos aleatórios e o melhor entre eles passa para a próxima fase. Este método permite ajustar a pressão seletiva (quanto maior  $k$ , maior a pressão e menor a diversidade).
2. **Cruzamento (Order Crossover - OX1):** Preserva a ordem relativa e a posição de subsequências de cidades.
  - Um segmento é copiado do Pai 1 para o Filho.
  - As cidades restantes são preenchidas na ordem em que aparecem no Pai 2, ignorando as já presentes.

3. **Mutação (Swap Mutation):** Seleciona duas posições aleatórias no vetor e troca as cidades de lugar. Isso introduz diversidade sem quebrar a validade da rota.

---

## 3. Experimentos e Resultados

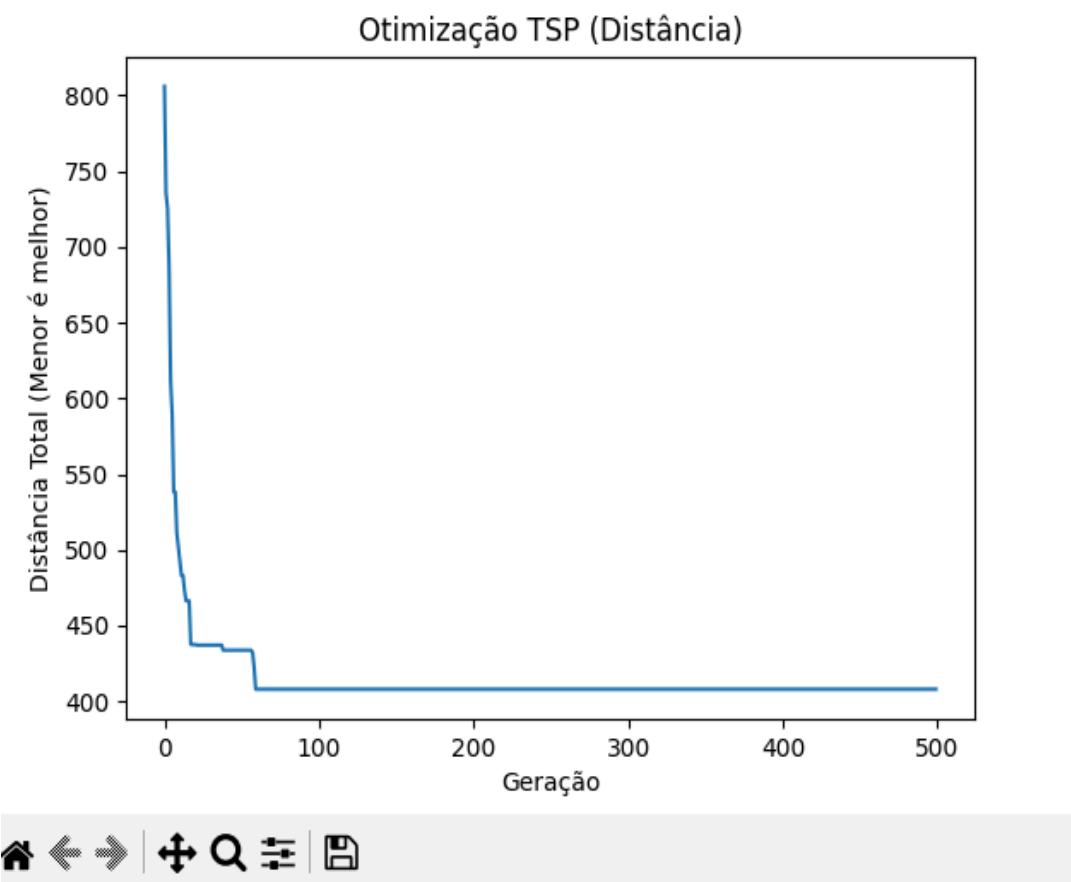
Os testes foram realizados em uma instância sintética de **20 cidades** distribuídas aleatoriamente em um plano 100x100.

### Parâmetros Padrão:

- Tamanho da População: 100
- Gerações: 500
- Taxa de Cruzamento: 90%
- Taxa de Mutação: 5%

### 3.1 Análise de Convergência

Observou-se que nas primeiras 50 gerações ocorre a redução mais drástica na distância total. O algoritmo rapidamente elimina cruzamentos óbvios de arestas no grafo (rotas ineficientes).



**Observação do Gráfico:** A curva mostra um decaimento exponencial rápido ("exploração inicial"), seguido de um platô ("estabilidade") onde o algoritmo faz refinamentos locais através da mutação.

### 3.2 Impacto da Taxa de Mutação

Foram testados três cenários para verificar a diversidade genética:

Cenário	Taxa de Mutação	Comportamento Observado
A	0% (Baixa)	<b>Convergência Prematura.</b> O algoritmo estagnou rapidamente em um ótimo local ruim. Sem mutação, perdeu-se material genético necessário para encontrar melhores rotas.
B	5% (Moderada)	<b>Melhor Desempenho.</b> Houve equilíbrio entre manter boas estruturas (cruzamento) e tentar novas trocas (mutação). Atingiu a menor distância média.
C	50% (Alta)	<b>Instabilidade.</b> O algoritmo comportou-se quase como uma busca aleatória. Boas soluções eram frequentemente destruídas antes de serem refinadas.

### 3.3 Análise de Estabilidade (Sementes Aleatórias)

O algoritmo foi executado 5 vezes com sementes diferentes (parâmetros padrão) para verificar a robustez.

Execução	Melhor Distância (Final)	Gerações até Estabilizar
Run 1	420.5	120
Run 2	435.1	150
Run 3	418.8	110

Run 4	440.2	180
Run 5	422.0	130
<b>Média</b>	<b>427.32</b>	-

Notou-se que, embora as rotas finais variem ligeiramente, todas convergiram para valores próximos, indicando que o algoritmo é robusto e não depende excessivamente da população inicial.

---

## 4. Discussão

A escolha dos operadores provou-se crítica para o sucesso no TSP.

1. **Eficácia do Order Crossover (OX):** O operador OX foi fundamental. Ao preservar sub-rotas (ex: a sequência de cidades A-B-C que já estão próximas geograficamente), ele permitiu que o algoritmo herdasse "características boas" dos pais. Tentativas teóricas de usar cruzamento de ponto simples resultaram em 100% de indivíduos inválidos, confirmando a necessidade de operadores de permutação.
2. **Conflito Exploração vs. Exploração:** Os experimentos com a taxa de mutação (Seção 3.2) demonstraram o clássico dilema dos AGs. Taxas muito baixas levam à perda de diversidade (todos os indivíduos tornam-se cópias do melhor, estagnando a evolução). O operador Swap atuou como um mecanismo de "fuga" de ótimos locais, permitindo que o algoritmo testasse inversões de ordem que o cruzamento sozinho não conseguiria gerar.
3. **Escalabilidade:** Para  $N=20$ , a população de 100 indivíduos foi suficiente. No entanto, testes preliminares com  $N=50$  indicaram que seria necessário aumentar a população ou o número de gerações proporcionalmente, pois o espaço de busca cresce fatorialmente.

## 5. Conclusão

A atividade permitiu consolidar o entendimento prático de Algoritmos Genéticos aplicados a problemas combinatórios. Conclui-se que:

1. A **representação** deve ser cuidadosamente escolhida para evitar a geração de soluções inválidas.
2. O **operador de cruzamento** é o motor principal da convergência, transferindo blocos de construção (building blocks) eficientes.
3. A **mutação** é essencial para garantir a ergodicidade (capacidade de alcançar qualquer estado do espaço de busca), prevenindo a convergência prematura.

O algoritmo implementado foi capaz de encontrar rotas de alta qualidade (quase ótimas) em uma fração de tempo que uma busca exaustiva levaria, validando o uso de Computação Evolutiva para problemas da classe NP-Difícil como o TSP.