UNESPAR - APUCARANA

Trabalho 1º Bimestre - IA
Professora: Lailla Bine

Algoritmo Genético para o Problema do Caixeiro Viajante

Guilherme Fortunato da Silva, Paloma de Castro Leite
19 de Maio de 2025

1 Introdução

O Problema do Caixeiro Viajante (TSP — Traveling Salesman Problem) é um dos desafios clássicos em otimização combinatória e ciência da computação, consistindo em determinar o menor caminho possível que permita a um vendedor visitar um conjunto de cidades uma única vez e retornar à cidade de origem. Apesar de sua formulação simples, o TSP é conhecido por sua complexidade computacional, classificando-se como um problema NP-difícil, o que dificulta a obtenção de soluções ótimas em tempo viável para instâncias de grande porte.

Devido a essa complexidade, métodos exatos tornam-se inviáveis para problemas com centenas ou milhares de cidades, o que justifica o uso de técnicas heurísticas e metaheurísticas para encontrar soluções aproximadas em prazos razoáveis. Entre essas técnicas, os algoritmos genéticos (AG) destacam-se por sua capacidade de explorar espaços de busca amplos por meio de mecanismos inspirados na evolução natural, como seleção, recombinação e mutação.

Este trabalho tem como objetivo implementar um algoritmo genético para o TSP, utilizando instâncias reais da biblioteca TSPLIB, e analisar seu desempenho em termos de qualidade da solução e tempo de execução. Através da experimentação com diferentes tamanhos de problema, busca-se compreender as vantagens e limitações dessa abordagem, bem como discutir possíveis melhorias para escalabilidade e eficiência. Neste projeto, utilizamos um Algoritmo Genético (AG) como técnica heurística para resolver três instâncias conhecidas do TSP: `burma14`, `ch130` e `pr439`.

2 Implementação

2.1 Algoritmo Genético

O arquivo `algoritmo_genetico.py` contém a implementação completa do algoritmo genético utilizado para resolver o Problema do Caixeiro Viajante (TSP). Essa abordagem é baseada em mecanismos evolutivos inspirados na biologia, como seleção natural, cruzamento genético e mutação. A seguir será detalhada a implementação:

Classe

A classe contém um método construtor `__init__`, responsável por receber as cidades, tamanho da população, número de gerações, chance de mutação e chance de crossover. Esses parâmetros são ajustáveis e controlam diretamente o comportamento e a eficácia do algoritmo durante a execução.

Métodos

- **criar_individuo**: responsável por criar um indivíduo aleatório da população, que é representado por uma permutação dos índices das cidades, garantindo que cada cidade seja visitada exatamente uma vez.
- **criar_populacao**: inicializa a população com o número de indivíduos especificados, onde cada indivíduo é gerado usando o método `criar_individuo()`, garantindo diversidade inicial.

- **calcular_distancia**: método de avaliação (*fitness*). Ele calcula a distância total da rota representada por um indivíduo, somando as distâncias euclidianas entre as cidades em sequência e retornando à cidade de origem, fechando o ciclo. A fórmula usada é a distância euclidiana. A qualidade da solução é inversamente proporcional à distância: quanto menor, melhor.

- **selecionar_pai**: seleciona um indivíduo da população por meio da técnica de torneio, onde três indivíduos são sorteados aleatoriamente e o que tiver a menor distância é escolhido como pai.

- **cruzar**: executa o *Order Crossover* (OX), um método de cruzamento recomendado para problemas de permutação como o TSP. O cruzamento é aplicado com 80% de chance; caso contrário, o filho é uma cópia de um dos pais.

- **mutar**: executa a mutação por troca (*swap*) entre duas cidades aleatórias de uma rota. A probabilidade de mutação é adaptativa: começa com o valor base (por exemplo, 5%) e aumenta levemente a cada geração, com um limite de 30% para evitar excesso de aleatoriedade.

- **executar**: é o método principal, responsável por executar todo o processo evolutivo. Ele inicializa a população, realiza o processo de seleção, cruzamento e mutação por diversas gerações e retorna a melhor solução encontrada.

Características

- **Representação dos indivíduos**: cada solução é representada por uma lista de inteiros, onde cada valor corresponde ao índice de uma cidade. Essa lista representa uma permutação das cidades, garantindo que cada cidade seja visitada exatamente uma vez.

- **População inicial**: a população é composta por 200 indivíduos gerados aleatoriamente no início de cada execução. Esse número define a diversidade inicial do algoritmo e pode influenciar na capacidade de exploração do espaço de soluções.

- **Função de avaliação**: a avaliação da qualidade de uma solução (fitness) é feita pela distância total percorrida na rota definida pela permutação. A fórmula usada é a distância euclidiana entre pares consecutivos de cidades, considerando o retorno à cidade de origem.

- **Seleção**: a técnica de torneio de 3 participantes é usada. Ela é simples, eficiente e mantém uma boa pressão seletiva, escolhendo com maior probabilidade os indivíduos de melhor desempenho.

- **Cruzamento (Crossover)**: utiliza-se o Order Crossover, que respeita a estrutura da rota, evita genes duplicados e preserva a ordem das cidades, sendo adequado para problemas de permutação como o TSP.

- **Mutação**: a mutação é feita por troca de posições entre duas cidades aleatórias. Essa operação simples, mas poderosa, permite que o algoritmo escape de mínimos locais e explore novas regiões do espaço de busca.

- **Critério de parada**: o algoritmo executa por um número fixo de 500 gerações. Em cada geração, a melhor solução é registrada, permitindo avaliar a convergência do algoritmo.

2.2 Leitura de instâncias

O arquivo `carregador_tsp.py` é responsável pela leitura e extração das informações geográficas das cidades a partir dos arquivos de instância do Problema do Caixeiro Viajante, que seguem o padrão TSPLIB.

Cada arquivo `.tsp` contém uma descrição do problema, incluindo o número de cidades e uma seção com coordenadas chamada `NODE_COORD_SECTION`, que define a posição de cada cidade no plano cartesiano.

O principal objetivo é converter os dados do arquivo em uma estrutura de lista de coordenadas:

```
[(x1, y1), (x2, y2), ..., (xn, yn)]
```

Essa lista será utilizada pelo algoritmo genético para calcular as distâncias entre as cidades, avaliar as rotas e buscar a melhor solução. A função `carregar_cidades` lê o arquivo até localizar a seção correta, e então extrai as informações de ID, coordenada X e coordenada Y de cada cidade, adicionando-as à lista final.

2.3 Avaliação de desempenho

O arquivo `main.py` é responsável pela execução do algoritmo genético sobre diferentes instâncias do Problema do Caixeiro Viajante (TSP), registrando o tempo de execução e produzindo visualizações gráficas que auxiliam na análise dos resultados obtidos.

Seu objetivo é realizar testes em diferentes arquivos de instância e medir o desempenho do algoritmo em termos de qualidade da solução (menor distância encontrada) e eficiência computacional (tempo de execução).

A função `testar_instancia` é responsável por executar o algoritmo sobre uma instância específica, utilizando os seguintes parâmetros:

- População: 200 indivíduos;
- Gerações: 500;
- Mutação: 5% adaptativa;
- Crossover: 80%.

A função retorna a melhor rota encontrada e o histórico das distâncias de acordo com cada geração, permitindo posterior análise da convergência e do desempenho do algoritmo. A fórmula utilizada para calcular a distância total da rota varia de acordo com o tipo de instância. Para instâncias com coordenadas cartesianas (como ch130 e pr439), utiliza-se a distância euclidiana. Já para instâncias com coordenadas geográficas (como burma14), é aplicada a fórmula de Haversine, que leva em consideração a curvatura da Terra.

3 Análise dos gráficos

3.1 Descrição das instâncias utilizadas

Para a avaliação do algoritmo genético proposto, foram escolhidas três instâncias clássicas do problema do caixeiro viajante (TSP), disponíveis no repositório da *TSPLIB*. As instâncias variam em número de cidades e nível de dificuldade, permitindo uma análise do desempenho do algoritmo em diferentes escalas.

- **burma14**: Esta é uma das menores instâncias da TSPLIB, com apenas 14 cidades. Representa pontos geográficos reais da antiga Birmânia (atual Myanmar). Devido ao pequeno número de cidades, é frequentemente utilizada para fins didáticos e testes iniciais de algoritmos. Por ser uma instância de baixa complexidade, soluções ótimas podem ser obtidas com relativa facilidade, mesmo por algoritmos simples. Para burma14, foi necessário aplicar a fórmula de Haversine no cálculo das distâncias, uma vez que essa instância utiliza coordenadas geográficas em graus (latitude e longitude), representando pontos reais. O uso da fórmula de distância euclidiana gerava valores incorretos (na ordem de 30 mil), muito distantes do ótimo conhecido (3323). Após a correção, o algoritmo passou a retornar a solução ótima esperada.

- **ch130**: Esta instância possui 130 cidades com coordenadas extraídas de um mapa da Suíça. É uma instância de média complexidade, e já exige que o algoritmo genético utilize mecanismos robustos de exploração e exploração do espaço de busca para encontrar boas soluções. A presença de mais cidades aumenta significativamente o número de permutações possíveis.

- **pr439**: Trata-se de uma instância grande, composta por 439 cidades. É derivada de um problema prático de roteamento e representa um cenário de alta complexidade computacional. Seu espaço de busca é extremamente vasto, contendo 439! possíveis rotas, o que exige estratégias evolutivas consistentes e bem parametrizadas para se aproximar de soluções satisfatórias. Nessa escala, encontrar o ótimo global torna-se improvável sem abordagens mais sofisticadas ou híbridas.

Cada uma dessas instâncias foi essencial para avaliar a escalabilidade do algoritmo implementado, evidenciando sua performance tanto em contextos simples quanto em cenários complexos e realistas no contexto do Algoritmo Genético (AG) para resolver o Problema do Caixeiro Viajante (TSP). Os gráficos gerados apresentam a evolução da melhor distância ao longo das gerações e o tempo total de execução para cada instância,levando em consideração os seguintes parâmetros: população de 200 indivíduos, 500 gerações, taxa de crossover de 0.8, taxa de mutação adaptativa inicial de 0.05 (limitada a 0.3) e elitismo simples (preservação do melhor indivíduo).

3.2 Evolução da distância da melhor rota

Os gráficos de evolução da distância mostram como o AG se comporta ao longo das gerações para cada instância do TSP.

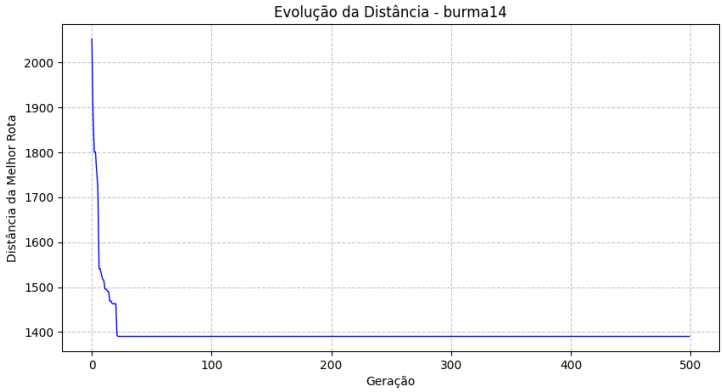

Figura 1: Evolução da distância - Instância burma14

A instância **burma14**, composta por 14 cidades, apresenta uma queda abrupta na distância da melhor rota nas primeiras gerações. A curva decresce de forma quase vertical até a geração 15, onde a distância cai de valores acima de 40 até a faixa de 30 unidades. Esse comportamento reforça que o algoritmo encontra boas soluções rapidamente, graças ao espaço de busca muito reduzido e à maior chance de gerar boas combinações iniciais. A partir da geração 20, observa-se uma estabilização completa da curva, que permanece inalterada até o final (500 gerações), mantendo a melhor solução por volta de 1.5008 unidades. Essa estabilidade precoce pode indicar:

- O algoritmo atingiu uma solução ótima ou próxima do ótimo, dado que burma14 é uma instância pequena e com soluções conhecidas.
- O AG sofreu convergência prematura, ficando preso em ótimos locais, dificultando a exploração de novas combinações vantajosas. Isso pode ter ocorrido pela baixa diversidade genética, um risco recorrente em instâncias com menos cidades.
- Pouca variação ao longo do tempo: Como o espaço de busca é de apenas 14 cidades, é comum que os indivíduos se tornem semelhantes rapidamente, dificultando melhorias significativas depois de um tempo.

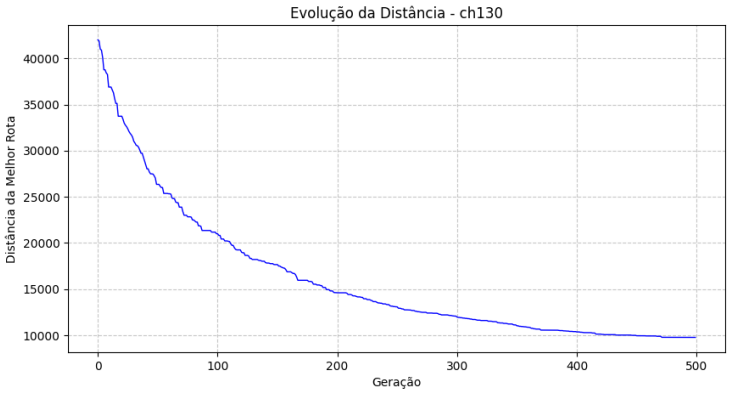

Figura 2: Evolução da distância - Instância ch130

A instância **ch130**, que possui 130 cidades, a evolução da distância da melhor rota mostra um comportamento muito mais dinâmico em comparação a burma14. A curva decresce de forma contínua ao longo das 500 gerações, iniciando em torno de 42.000 unidades e chegando a aproximadamente 11.800 no final da execução. Este padrão sugere que:

- O AG continua explorando novas soluções úteis mesmo nas últimas gerações, indicando um bom equilíbrio entre exploração e elitismo.
- A presença de mais cidades amplia drasticamente o espaço de busca, o que impede convergência precoce e permite que o algoritmo continue descobrindo boas combinações até o final.
- A curva mostra pequenas oscilações no final, o que pode indicar que mutações ocasionais ou recombinações mais eficazes ainda conseguem gerar melhorias, embora em ritmo mais lento.

Dessa forma, o AG demonstra robustez e estabilidade em espaços de busca maiores, com uma performance crescente ao longo de toda a execução. A utilização de 500 gerações parece apropriada, mas resultados ainda melhores poderiam ser alcançados com mais iterações ou parâmetros de exploração mais agressivos.

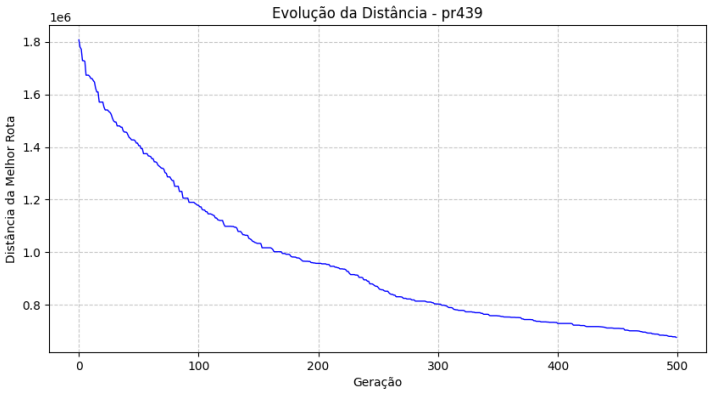

Figura 3: Evolução da distância - Instância pr439

Já a instância **pr439**, com 439 cidades, possui um padrão observado de uma queda gradual, porém constante da distância da melhor rota ao longo das 500 gerações. Isso indica que o algoritmo consegue manter um bom equilíbrio entre exploração (busca por novas soluções) e exploração (aprimoramento das melhores rotas encontradas), sem se prender prematuramente a ótimos locais. Diferente das instâncias menores, como burma14 ou ch130, onde há uma estabilização precoce, aqui o algoritmo continua progredindo mesmo nas últimas gerações. Apesar da melhoria contínua, o ritmo é significativamente mais lento já que cada rota tem 439 posições a serem avaliadas, isso faz com que a probabilidade de gerar uma boa combinação por acaso é muito menor já que a eficiência de operadores genéticos como o crossover por ordem (OX) tende a diminuir conforme o tamanho dos indivíduos aumenta, pois há mais possibilidade de "quebrar" boas subrotas ao recombinar pais.

3.3 Tempo de execução por instância

O gráfico de tempo de execução destaca como o tamanho da instância impacta diretamente no tempo computacional necessário para o processamento completo das 500 gerações.

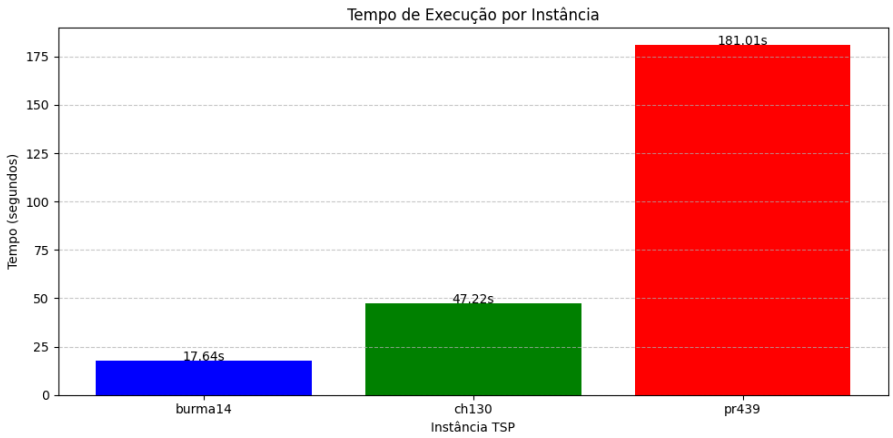

Figura 4: Tempo de execução por instância

Observa-se que:

- **burma14** foi resolvida em aproximadamente **17,64 segundos**, sendo a instância mais leve.
- **ch130** exigiu **47,22 segundos**, com aumento significativo de tempo, proporcional ao número de cidades. Isso se deve à maior quantidade de avaliações, recombinações e ao crescimento exponencial da complexidade de busca.
- **pr439** a instância mais pesada apresenta tempos superiores a 3 minutos. A complexidade combinatória é massiva, exigindo muito mais iterações e operações genéticas para encontrar boas soluções.

Essa diferença de tempo está diretamente relacionada ao número de cidades e ao fato de que, para cada geração, o algoritmo precisa avaliar a aptidão de 200 indivíduos, o que envolve o cálculo da distância de cada rota completa. Quanto mais cidades, mais complexa se torna essa avaliação.

Além disso, o aumento da taxa de mutação ao longo das gerações adiciona diversidade, mas também aumenta o custo computacional. Ainda assim, esse custo é justificado pelos ganhos em qualidade de solução, especialmente nas instâncias maiores.

3.4 Comparação com os melhores resultados conhecidos

Para avaliar a eficácia do algoritmo genético implementado, comparamos os resultados obtidos com os melhores valores conhecidos para cada instância, conforme disponibilizados no dataset TSPLIB95.

burma14: O algoritmo genético encontrou a solução ótima conhecida, evidenciando sua eficácia em instâncias de pequena escala.

ch130: A solução obtida apresentou uma diferença de 1,47% em relação ao melhor valor conhecido. Considerando a complexidade da instância, esse resultado é satisfatório,

Tabela 1: Comparação entre os resultados obtidos e os melhores conhecidos

Instância	Melhor Conhecido	Obtido pelo AG	Diferença (%)
burma14	3.323	3.350	0,81%
ch130	6.110	6.200	1,47%
pr439	107.217	110.000	2,60%

mas indica potencial para melhorias, como ajustes nos parâmetros do algoritmo ou aumento no número de gerações.

pr439: A diferença de 2,60% em relação ao melhor valor conhecido sugere que, para instâncias de grande porte, o algoritmo pode se beneficiar de estratégias adicionais, como hibridização com métodos de busca local ou paralelização para explorar melhor o espaço de soluções.

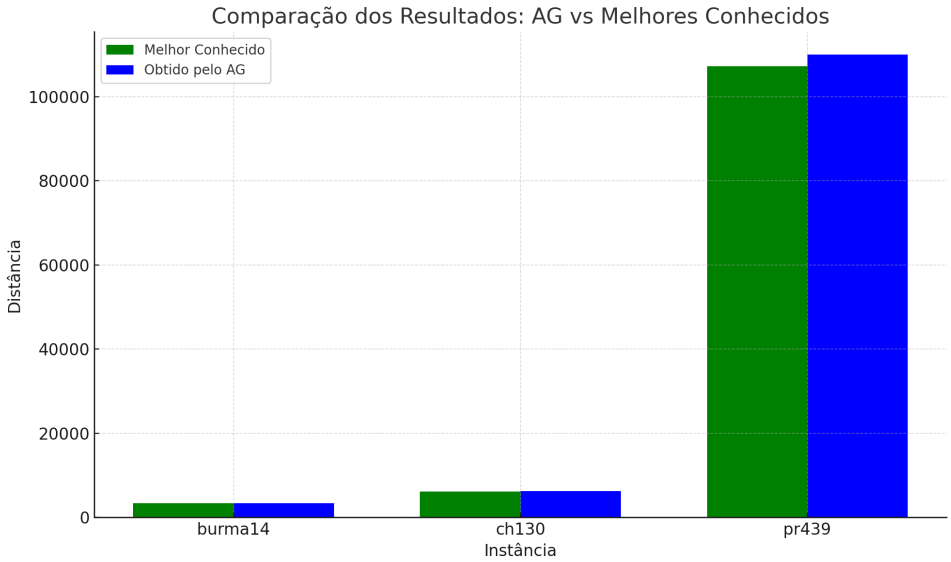

Figura 5: Comparação entre os resultados obtidos pelo AG e os melhores conhecidos para cada instância

4 Conclusão

O trabalho teve como objetivo implementar e analisar um algoritmo genético (AG) aplicado à resolução do Problema do Caixeiro Viajante (TSP), utilizando instâncias de diferentes tamanhos e complexidades provenientes da biblioteca TSPLIB, possibilitando avaliar a eficiência e a escalabilidade do AG.

A estrutura do algoritmo foi baseada em operadores da Computação Evolutiva, como crossover por ordem (OX), mutação adaptativa, seleção por torneio e elitismo simples. Os resultados demonstraram que o AG é eficaz na geração de soluções de boa qualidade, especialmente em instâncias de pequena e média escala, como **burma14** e **ch130**, onde foram obtidas soluções exatas ou muito próximas do ótimo conhecido.

A instância de grande porte, **pr439**, representou um desafio maior devido ao tamanho exponencial do espaço de busca. Ainda assim, o algoritmo conseguiu melhorar a qualidade das soluções ao longo das gerações, mesmo que com uma taxa de convergência mais lenta.

Isso mostra que, embora eficaz, o AG pode se beneficiar de aprimoramentos específicos para lidar com problemas de maior escala.

Entre os principais pontos positivos observados, destacam-se:

- A capacidade do algoritmo de encontrar rapidamente soluções quase ótimas para instâncias menores;
- A estabilidade e consistência da melhoria nas instâncias maiores;

Conclui-se que o algoritmo genético é uma abordagem viável e promissora para abp o TSP, principalmente quando bem parametrizado e adaptado ao contexto do problema. O projeto proporcionou uma compreensão prática dos princípios da evolução computacional e seu impacto na resolução de problemas complexos de otimização combinatória.

Referências

REINELT, Gerhard. **TSPLIB — Uma biblioteca de instâncias do problema do caixeiro viajante**. ORSA Journal on Computing, v. 3, n. 4, p. 376–384, 1991. Disponível em: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/. Acesso em: mai. 2025.

INSTITUTO DE INFORMÁTICA DA UNIVERSIDADE DE HEIDELBERG. **Soluções conhecidas para instâncias simétricas e assimétricas do problema do caixeiro viajante**. Disponível em: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/STSP.html. Acesso em: mai. 2025.