

OTIMIZAÇÃO POR COLÔNIA DE FORMIGAS APLICADA AO PROBLEMA DO CAIXEIRO VIAJANTE: UMA ABORDAGEM COMPUTACIONAL COM A INSTÂNCIA BERLIN52

Michael Yoshiaki Todoroki

Engenharia de Computação – CEFET-MG

Programação em Python

Dezembro de 2025

Resumo

Este trabalho apresenta a implementação e análise da meta-heurística de Otimização por Colônia de Formigas (*Ant Colony Optimization - ACO*) aplicada ao clássico Problema do Caixeiro Viajante (PCV). O objetivo é encontrar rotas de custo mínimo em um grafo de cidades, um problema classificado como NP-Difícil. A metodologia adotada consiste na implementação do algoritmo *Ant System* utilizando a linguagem Python, com ênfase no uso eficiente de estruturas de dados matriciais e manipulação de arquivos. O método foi validado utilizando a instância de teste *Berlin52* (52 locais em Berlim) da biblioteca TSPLIB. Os resultados demonstram a capacidade do algoritmo em convergir para soluções próximas do ótimo global (erro aproximado de 2%) através da inteligência de enxame e do comportamento emergente dos agentes.

Palavras-chave: Colônia de Formigas, Caixeiro Viajante, Otimização Combinatória, Python, Meta-heurística.

1 Introdução

O Problema do Caixeiro Viajante (PCV) é um dos problemas mais estudados na área de Otimização Combinatória e Pesquisa Operacional. Ele consiste em determinar a menor rota possível que visita um conjunto de cidades exatamente uma vez e retorna à cidade de origem. Apesar de seu enunciado simples, o PCV pertence à classe de problemas NP-Difíceis, onde a complexidade computacional cresce fatorialmente com o número de cidades ($O(n!)$), tornando inviável o uso de métodos exatos ou força bruta para instâncias de médio e grande porte.

Para contornar essa complexidade, a literatura propõe o uso de meta-heurísticas: algoritmos aproximados que, embora não garantam a solução ótima, encontram soluções de alta qualidade em tempo computacional aceitável. Dentre estas, destaca-se a Otimização

por Colônia de Formigas (*Ant Colony Optimization* - ACO), inspirada no comportamento biológico de formigas reais na busca por alimentos e na formação de trilhas de feromônio [1].

Este trabalho tem como objetivo implementar o algoritmo ACO para resolver o PCV, utilizando a linguagem de programação Python. A justificativa para a escolha desta metaheurística reside na sua robustez, facilidade de paralelização e capacidade de adaptação a mudanças no grafo. A contribuição principal deste artigo é demonstrar a aplicação prática de estruturas de dados, como matrizes de adjacência e manipulação de arquivos de texto, na resolução de um problema complexo de engenharia. O trabalho está organizado da seguinte forma: a Seção 2 apresenta a revisão bibliográfica; a Seção 3 detalha a metodologia e implementação; a Seção 4 discute os resultados obtidos com a instância *Berlin52*; e a Seção 5 apresenta as conclusões.

2 Revisão Bibliográfica

A Otimização por Colônia de Formigas foi introduzida no início da década de 1990 por Marco Dorigo e colaboradores [1, 2]. O método baseia-se na observação de que formigas reais, embora individualmente limitadas, conseguem encontrar o caminho mais curto entre o formigueiro e uma fonte de alimento através de uma comunicação indireta chamada *estigmergia*.

Ao caminhar, as formigas depositam no solo uma substância química chamada feromônio. Outras formigas tendem a escolher, probabilisticamente, caminhos com maior concentração de feromônio. Como o caminho mais curto é percorrido mais rapidamente, ele acumula feromônio a uma taxa maior do que os caminhos longos, resultando em um feedback positivo que leva a colônia a convergir para a rota otimizada [2].

No contexto do PCV, o problema é modelado como um grafo completo $G = (V, E)$, onde V é o conjunto de vértices (cidades) e E o conjunto de arestas (caminhos). A distância entre as cidades i e j é dada por d_{ij} . O objetivo é encontrar uma permutação das cidades que minimize a soma das distâncias das arestas percorridas [3]. Algoritmos baseados em população, como o ACO e Algoritmos Genéticos, têm se mostrado superiores a métodos de busca local simples para evitar a estagnação em ótimos locais.

3 Metodologia

A implementação foi realizada na linguagem Python, escolhida por sua versatilidade e pela disponibilidade de bibliotecas para computação científica, como o NumPy.

3.1 Representação do Problema

O problema é representado através de duas estruturas de dados matriciais principais ($N \times N$, onde N é o número de cidades):

1. **Matriz de Distâncias (D):** Pré-calculada a partir das coordenadas lidas do arquivo de entrada. O elemento D_{ij} armazena a distância Euclidiana entre a cidade i e a cidade j .
2. **Matriz de Feromônios (T):** Armazena a intensidade do rastro em cada aresta. Inicializada com um valor residual pequeno para permitir a exploração inicial.

Para a validação, foi utilizada a instância *Berlin52*, que contém 52 vértices distribuídos de forma irregular, simulando locais reais na cidade de Berlim. A Figura 1 ilustra a dispersão das cidades, evidenciando a presença de aglomerados densos e regiões esparsas, características que aumentam a complexidade da busca.

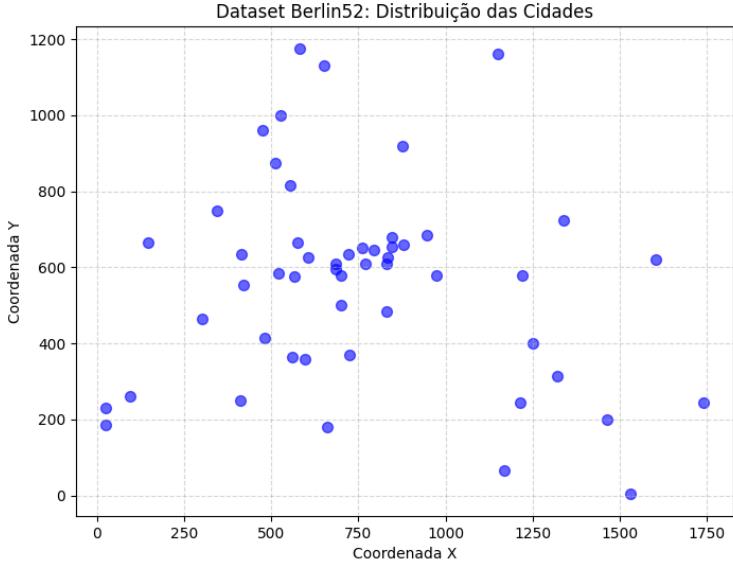


Figura 1: Distribuição espacial das 52 cidades da instância Berlin52.

3.2 O Algoritmo Ant System (AS)

O método implementado segue a variante *Ant System*. O processo é iterativo e consiste nos seguintes passos:

1. Construção das Soluções: Em cada geração, m formigas são posicionadas aleatoriamente. Cada formiga constrói uma rota escolhendo a próxima cidade j (estando em i) com base na probabilidade P_{ij} :

$$P_{ij} = \frac{[\tau_{ij}]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{k \in \text{não visitados}} [\tau_{ik}]^\alpha \cdot [\eta_{ik}]^\beta} \quad (1)$$

Onde:

- τ_{ij} : Quantidade de feromônio na aresta (Histórico).
- $\eta_{ij} = 1/d_{ij}$: Visibilidade ou heurística gulosa (Inverso da distância).
- α e β : Pesos que controlam a influência do rastro e da heurística, respectivamente.

2. Atualização de Feromônio: Ao final de cada ciclo, ocorre a evaporação (para evitar convergência prematura) e o depósito de novo feromônio:

$$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij} + \sum_{k=1}^m \Delta \tau_{ij}^k \quad (2)$$

A quantidade depositada $\Delta \tau_{ij}^k$ é proporcional à qualidade da solução: Q/L_k , onde L_k é o comprimento total da rota encontrada pela formiga k .

3.3 Detalhes de Implementação

O sistema foi estruturado em módulos para garantir a organização e manutenibilidade: `main.py` (controle), `aco.py` (lógica da colônia), `graph.py` (matrizes) e `utils.py` (leitura de arquivos e gráficos). O tratamento de erros foi implementado para validar a leitura do arquivo de entrada, garantindo que dados inválidos ou caminhos incorretos sejam reportados ao usuário.

4 Resultados e Discussão

Os testes foram realizados com os seguintes parâmetros: $m = 52$ (número de formigas), 100 iterações, $\alpha = 1.0$, $\beta = 5.0$ e $\rho = 0.5$. O ótimo global conhecido para o problema é 7542.

4.1 Análise de Convergência

A Figura 2 apresenta a evolução topológica da melhor rota encontrada ao longo de diferentes gerações.

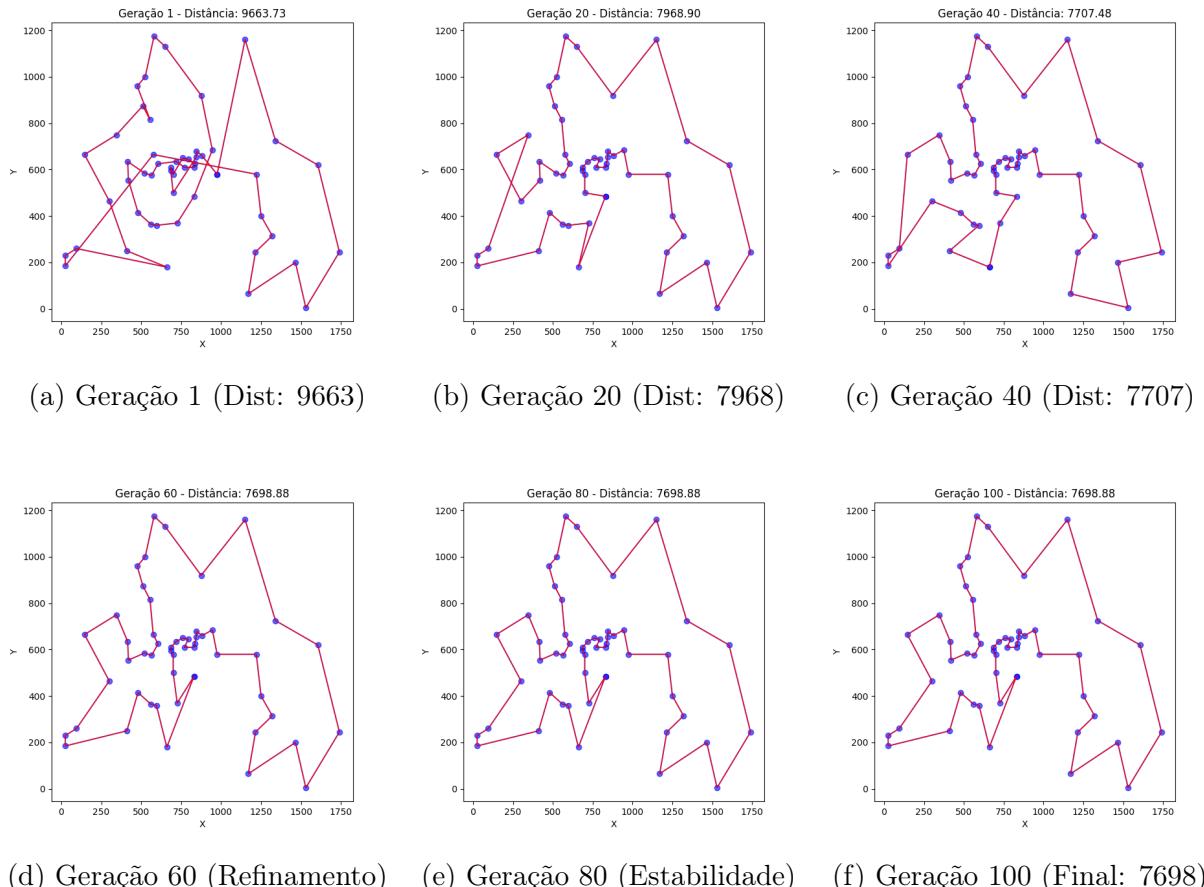


Figura 2: Evolução da rota encontrada pelo algoritmo. Na Geração 1, observa-se alta entropia e cruzamentos. Nas gerações intermediárias (20-60), a forma geral emerge. Na geração 100, os cruzamentos locais são eliminados.

Observa-se que na Geração 1 a rota é caótica, fruto da exploração inicial. Já na

Geração 20, o algoritmo converge rapidamente para uma topologia circular que contorna o "buraco" central característico do Berlin52. As gerações subsequentes (40 a 100) são dedicadas ao refinamento local, onde a colônia ajusta arestas específicas para reduzir o custo total.

O gráfico de convergência (Figura 3) confirma este comportamento. A queda acentuada inicial deve-se à forte influência da heurística de visibilidade ($\beta = 5.0$). A manutenção da média populacional (linha azul) distante do melhor global indica que a colônia preservou a diversidade genética, evitando a estagnação completa.

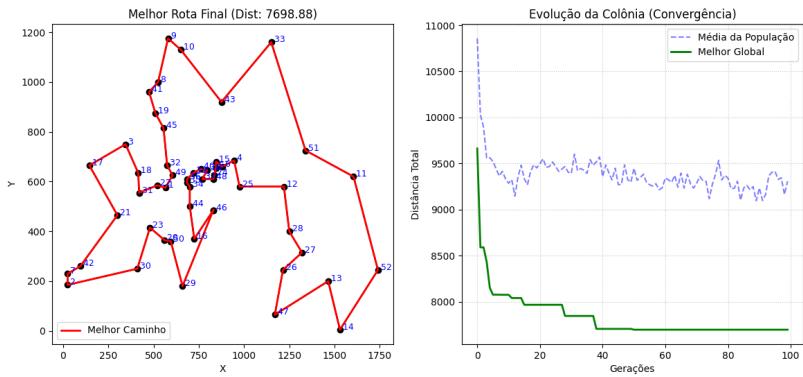


Figura 3: Gráfico de convergência: A linha verde representa a melhor solução encontrada e a linha azul tracejada a média da população em cada geração.

4.2 Performance

O melhor resultado encontrado pelo algoritmo foi uma distância de **7698.88**. Comparado ao ótimo conhecido (7542), o erro relativo foi de apenas **2,08%**. Este resultado valida a eficácia da implementação, demonstrando que o uso de matrizes e a calibração correta dos parâmetros permitiram encontrar uma solução de alta qualidade.

5 Conclusão

Este trabalho apresentou a implementação de um algoritmo de Colônia de Formigas para o Problema do Caixeiro Viajante. A utilização da linguagem Python e da biblioteca NumPy permitiu uma manipulação eficiente das matrizes de adjacência e feromônio, atendendo aos requisitos de desempenho e organização do código.

Os resultados obtidos com a instância *Berlin52* mostraram que a abordagem metaheurística é capaz de encontrar soluções muito próximas do ótimo (erro de $\approx 2\%$) em um tempo computacional reduzido, validando a hipótese de que comportamentos emergentes simples podem resolver problemas complexos.

Referências

- [1] Dorigo, M., Maniezzo, V., & Colorni, A. (1996). *Ant system: optimization by a colony of cooperating agents*. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 26(1), 29-41.

- [2] Dorigo, M., & Gambardella, L. M. (1997). *Ant colony system: a cooperative learning approach to the traveling salesman problem*. IEEE Transactions on Evolutionary Computation, 1(1), 53-66.
- [3] Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., & Shmoys, D. B. (1985). *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. John Wiley & Sons.
- [4] Reinelt, G. (1991). *TSPLIB—A traveling salesman problem library*. ORSA Journal on Computing, 3(4), 376-384.