Documentação Técnica: Algoritmo ACO Distribuído com Interface Gráfica

Dalton de Oliveira Cardoso,
João Victor dos Santos,
Leonardo de Oliveira Carvalho,
Victor Monteiro Martinelli Gataroli
Projeto de Computação Distribuída

17 de junho de 2025

1 Introdução

Este projeto apresenta uma solução distribuída para o problema do Caixeiro Viajante (*Traveling Salesman Problem* - TSP) baseada no algoritmo de Otimização por Colônia de Formigas (*Ant Colony Optimization* - ACO). O sistema é composto por um servidor central que coordena a distribuição de tarefas para múltiplos clientes, responsáveis por simular formigas explorando diferentes rotas. Ao final da execução, o resultado pode ser visualizado em uma interface gráfica desenvolvida com `tkinter` e `matplotlib`, permitindo comparar visualmente o grafo original com o percurso otimizado.

2 Arquitetura do Sistema

O sistema foi organizado em quatro módulos principais:

- **server.py**: Responsável pela criação do grafo, distribuição das tarefas, coleta dos caminhos gerados pelos clientes e atualização dos níveis de feromônio.
- **client.py**: Cada cliente representa uma formiga e executa localmente o algoritmo ACO, comunicando-se com o servidor para enviar suas soluções.
- **aco_core.py**: Implementa toda a lógica do ACO, incluindo o cálculo das probabilidades, seleção dos próximos nós e as regras de atualização de feromônio.
- **gui_result.py**: Permite a visualização gráfica dos resultados, facilitando a comparação entre o grafo original e o melhor percurso encontrado.

3 Decisões de Projeto

3.1 Escolha do Algoritmo

A opção pelo ACO se deve à sua eficácia em resolver problemas combinatórios complexos como o TSP. O algoritmo permite explorar diversas soluções e reforçar, por meio

do feromônio, os caminhos mais promissores ao longo das iterações. Foram utilizados parâmetros clássicos, como `alpha`, `beta` e `rho`, conforme descrito por Dorigo.

3.2 Distribuição via Sockets

A comunicação entre servidor e clientes ocorre via sockets TCP, o que possibilita a execução paralela de múltiplas formigas. Dessa forma, a exploração dos caminhos é feita de maneira independente, aumentando a diversidade de soluções e ilustrando a viabilidade da abordagem distribuída.

3.3 Configuração dos Parâmetros

Optou-se por valores `ALPHA = 1.0` e `BETA = 3.0`, de modo a privilegiar a heurística nas primeiras iterações e evitar convergência prematura. O parâmetro de evaporação (`RHO = 0.1`) garante que soluções antigas percam influência gradualmente, abrindo espaço para novas possibilidades.

3.4 Representação do Grafo

As arestas são modeladas por meio de uma classe `Edge`, contendo tanto a distância euclidiana quanto o nível de feromônio. O grafo é armazenado como uma matriz de adjacência, o que facilita o acesso eficiente entre pares de nós.

3.5 Visualização dos Resultados

A interface gráfica utiliza o `matplotlib` integrado ao `tkinter` via `FigureCanvasTkAgg`. São exibidos dois gráficos: o primeiro mostra o grafo original, e o segundo destaca a melhor rota encontrada pelo algoritmo. Essa abordagem torna o funcionamento do algoritmo mais intuitivo para qualquer usuário. Para uma melhor visualização dos dados obtidos realizamos uma modificação para definir uma seed fixa para o grafo original.

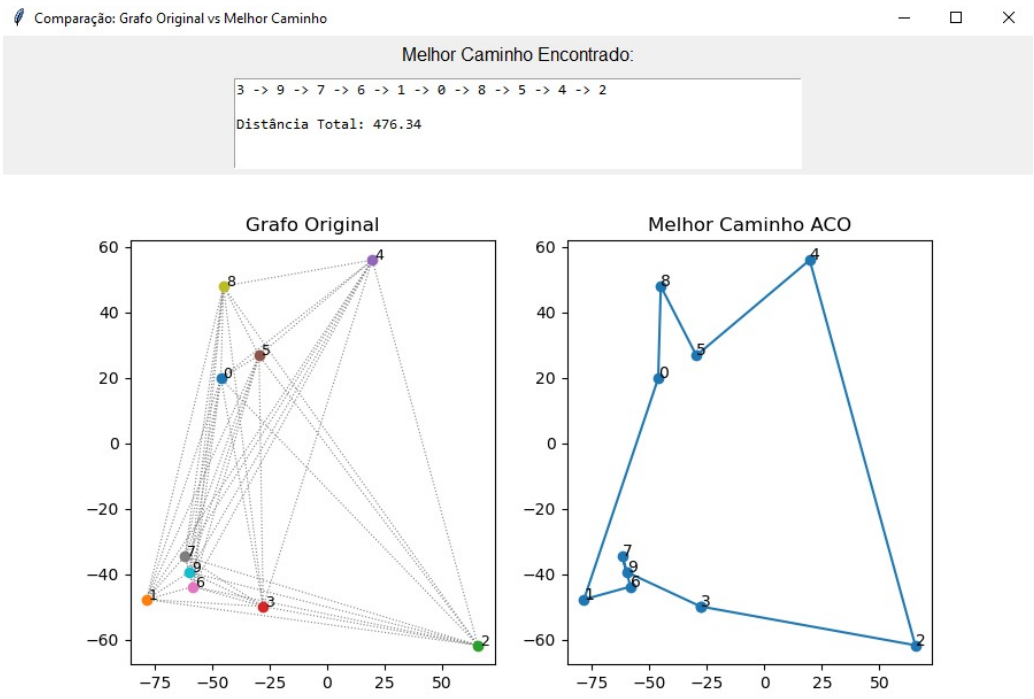

Figura 1: Exemplo 1: Melhor rota encontrada com 2 clientes utilizando seed randômica. Distância total: 476.34

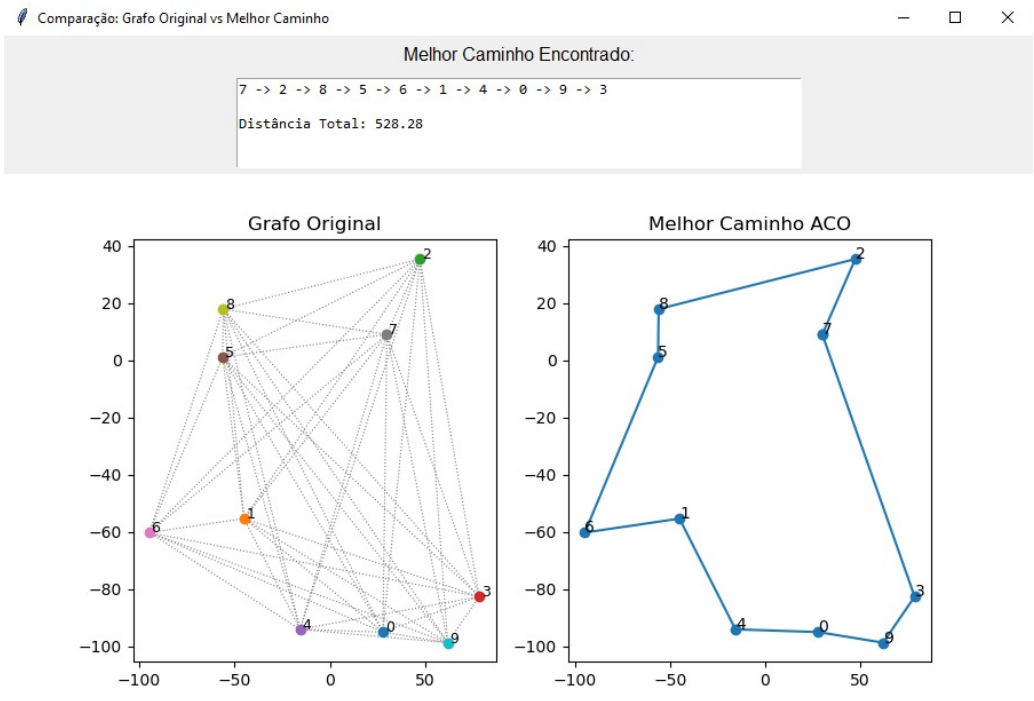

Figura 2: Exemplo 2: Variação da rota ao usar mais formigas. Distância total: 528.28

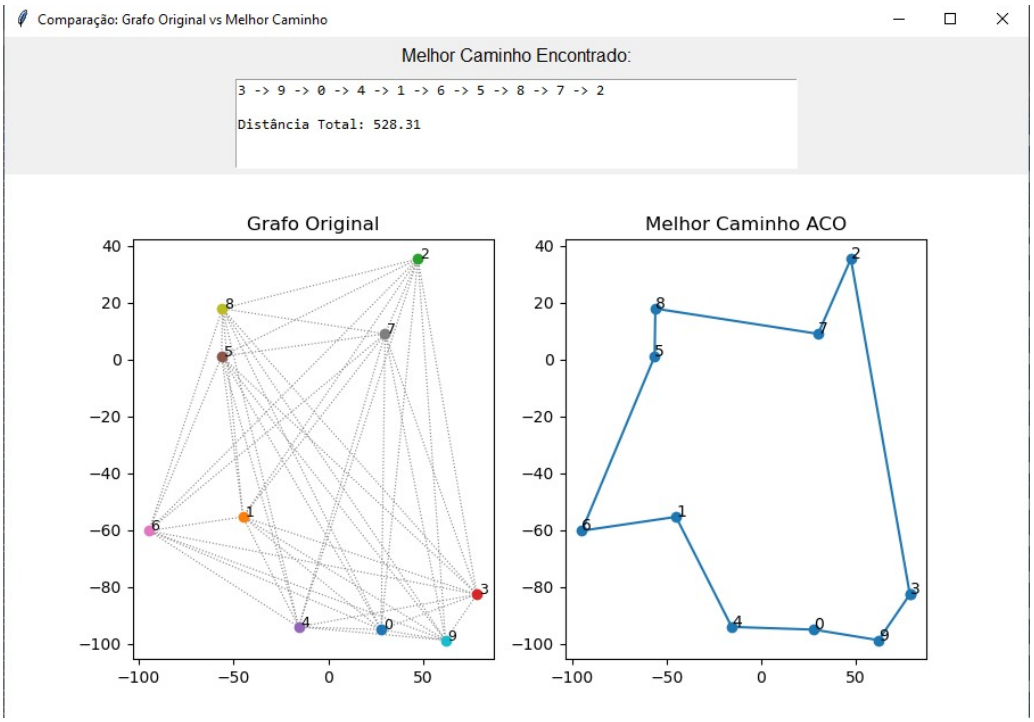

Figura 3: Exemplo 3: Nova solução encontrada em outra execução. Distância total: 528.31

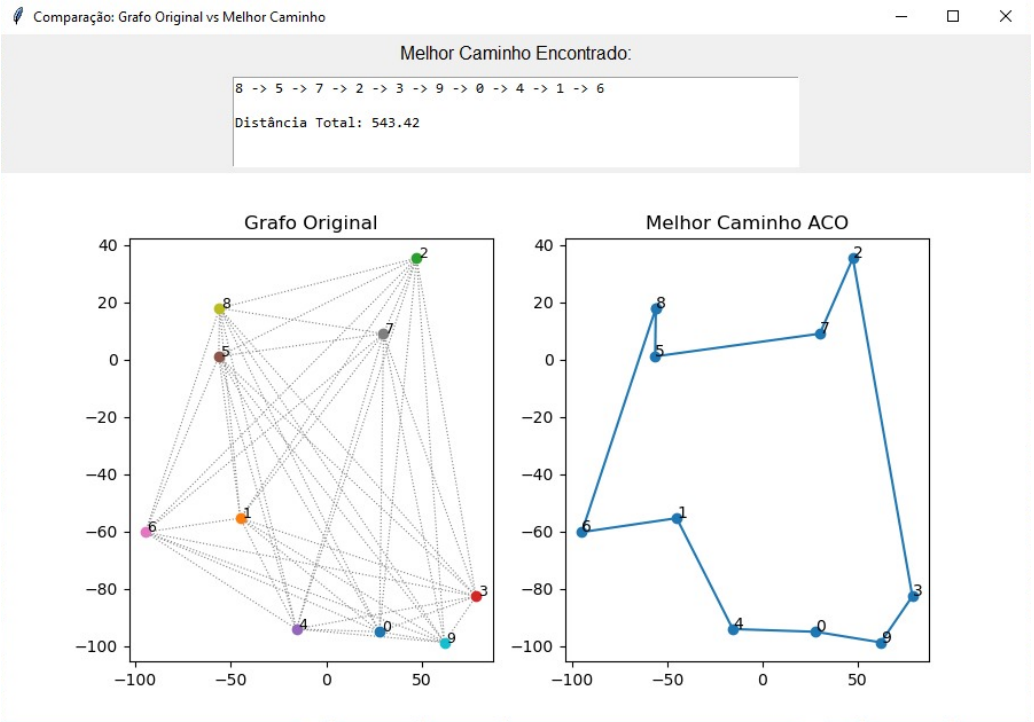

Figura 4: Exemplo 4: Comportamento variável do ACO com múltiplas formigas. Distância total: 543.42

3.6 Persistência dos Dados

Ao final do processamento, o servidor armazena os dados em um arquivo `aco_result.pkl`, incluindo os nós, a melhor rota e a distância total. Dessa forma, é possível reprocessar e visualizar o resultado posteriormente, sem necessidade de nova execução.

4 Escalabilidade

Aumentar o número de clientes traz maior diversidade na exploração das rotas, já que mais formigas percorrem diferentes caminhos. Isso pode resultar em soluções finais distintas, uma vez que o algoritmo é estocástico por natureza. Para garantir testes reprodutíveis, pode-se definir uma semente fixa para o gerador de números aleatórios, utilizando `random.seed()`.

5 Possíveis Melhorias

- Registro detalhado do histórico de tours realizados por cada cliente.
- Registro de medida de eficiência para cada tour.
- Execução remota em máquinas distintas, com configuração de IPs.
- Implementação de métodos comparativos, como PSO e algoritmos genéticos.

6 Conclusão

O projeto demonstra a integração prática de técnicas inspiradas pela natureza com conceitos de computação distribuída. A separação clara entre os módulos de cálculo, coordenação e visualização contribui para a flexibilidade e a facilidade de manutenção do sistema. Os resultados obtidos mostram como a cooperação entre processos pode resolver problemas de alta complexidade de forma eficiente e compreensível.