Caixeiro Viajante com Grafo Incompleto

Felipe N. Avelino, João P. G. Guedes, Matheus Schiochet

Departamento de Ciência da Computação
Universidade do Estado de Santa Catarina (UDESC) – Joinville, SC – Brazil

Abstract. *This study addresses the Traveling Salesman Problem (TSP) in incomplete graphs, simulating real-world conditions such as blocked roads. The solution employs an ant colony optimization algorithm that improves routes over iterations.*

Resumo. *Este estudo aborda o problema do Caixeiro Viajante (TSP) em grafos incompletos, simulando condições reais como estradas bloqueadas. A solução utiliza um algoritmo heurístico de colônia de formigas, que otiniza trajetos ao longo de iterações.*

1. Introdução

O problema do Caixeiro Viajante (Traveling Salesman Problem, TSP) é um dos problemas mais conhecidos em otimização combinatória. Ele consiste em encontrar o menor percurso que permita visitar todas as cidades exatamente uma vez e retornar à cidade de origem. Este trabalho aborda o TSP em um contexto realista, onde o grafo que representa o problema é incompleto, simulando condições reais de estradas bloqueadas. A solução proposta utiliza um algoritmo heurístico baseado em colônia de formigas.

2. Definição do Problema

O problema do Caixeiro Viajante será resolvido considerando estradas bloqueadas, ou seja, determinadas conexões entre cidades são inexistentes ou inacessíveis. Essa condição é representada pela ausência de arestas no grafo do problema. O objetivo é encontrar um percurso viável que conecte todas as cidades, minimizando a distância total.

Variável Escolhida:

Estradas Bloqueadas - Modelada por meio de valores "`None`" na matriz de distâncias, indicando a inexistência de um caminho direto entre duas cidades

3. Solução Proposta

A solução proposta utiliza um algoritmo de colônia de formigas, implementado na classe **`Colonia`**, que simula o comportamento coletivo das formigas através do depósito e evaporação de feromônio em cada caminho percorrido. O algoritmo pode ser dividido em três etapas principais, todas refletidas nas funções e métodos do código:

Inicialização: Na criação de uma instância da classe **`Colonia`** (método **`__init__`**), cada caminho na matriz de distâncias é convertido em um objeto da classe **`Caminho`**, que contém atributos como a distância, o feromônio inicial (**`ferInicial`**), a taxa de evaporação (**`evap`**) e a quantidade acumulada de feromônio após as iterações. Cidades que não possuem conexão são representadas por valores None.

Busca de Soluções: Durante a execução do método **`rodar`**, que realiza múltiplas iterações (**`numIt`**), as formigas percorrem o grafo. Para cada cidade, são calculados os valores de probabilidade (p) para cada caminho disponível. Essas probabilidades são baseadas no produto entre o feromônio acumulado (**`tn`**) e a atratividade do caminho (inverso da distância). A seleção de um próximo destino é feita usando a função **`metodo_da_roleta`**, que escolhe aleatoriamente um caminho proporcional às probabilidades calculadas. Ao final de cada percurso, é registrada a distância total percorrida.

Atualização de Feromônio: Após completar os percursos, o método rodar realiza a atualização das trilhas de feromônio para todas as conexões (pares de cidades representados na lista conjuntos). O cálculo inclui a evaporação do feromônio antigo e a adição de reforço proporcional ao desempenho (menores distâncias recebem maior reforço). Essa lógica é implementada ao longo do loop que ajusta o atributo **N** de cada objeto Caminho.

4. Testes Realizados e Resultados

Para validar a eficiência do algoritmo de colônia de formigas, foram realizados testes utilizando uma matriz de distâncias obtida do arquivo **`bays29.tsp`**, que representa um grafo com 29 cidades. O arquivo foi processado pela função **`ler_matriz_arquivo`**, que converte os dados em uma matriz com distâncias entre as cidades, considerando conexões inexistentes representadas por valores **`None`**.

Após a leitura da matriz, foi criada uma instância da classe **`Colonia`**, configurada com os seguintes parâmetros:

Taxa de evaporação de feromônio: 0,01

Feromônio inicial em cada caminho: 0,1

Reforço de feromônio para os percursos: 10

O método **`rodar`** foi executado para simular o comportamento das formigas percorrendo o grafo. Durante a execução, os seguintes dados foram coletados:

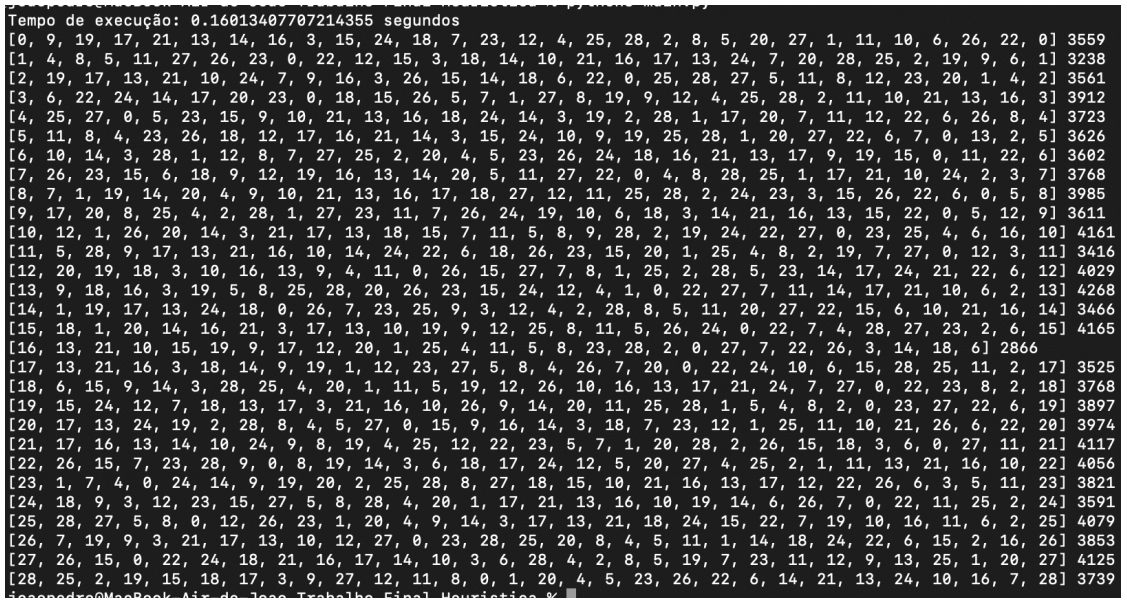
Tempo de execução: Calculado usando as funções do módulo **`time`** para medir o intervalo entre o início e o fim da execução.

Percursos obtidos: Os percursos gerados pelo algoritmo foram exibidos, indicando as sequências de cidades visitadas.

Distância total: A distância total é mostrada no final de cada linha, sendo a distância percorrida no trajeto.

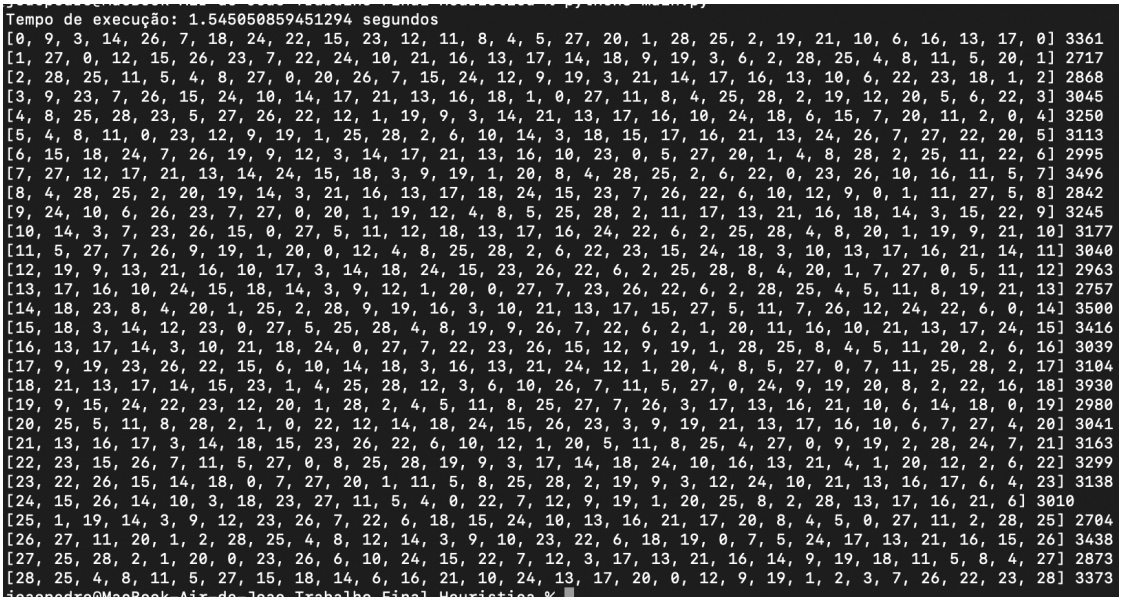
Resultados:

- **Com 10 iterações:**



- **Com 50 Iterações:**

- **Com 100 iterações:**



5. Conclusão

Por fim, após realizar algumas iterações, e encontrar um caminho base inicial, podemos ver que as formigas vão encontrando caminhos melhores conforme vão aumentando o número de iterações. Ao final de cada linha, temos a distância total do caminho percorrido, e os valores da linha indicam os índices das cidades que foram percorridas. A utilização do algoritmo de colônia de

formigas em um grafo incompleto mostrou que essa abordagem é bastante eficaz para resolver o problema do Caixeiro Viajante, mesmo em situações mais próximas do mundo real. Um grafo incompleto representa um cenário onde nem todas as cidades estão conectadas diretamente, o que simula, por exemplo, estradas bloqueadas ou inacessíveis. Apesar desse desafio, o algoritmo foi capaz de encontrar soluções viáveis graças ao fato de que ele utiliza uma estratégia inteligente baseada no comportamento das formigas na natureza: elas depositam feromônio nos caminhos que percorrem, o que ajuda a destacar as melhores rotas ao longo das iterações. Além disso, a evaporação do feromônio impede que soluções ruins sejam reforçadas, ajudando o algoritmo a convergir para opções mais eficientes. Em resumo, o algoritmo de colônia de formigas provou ser uma ferramenta útil para lidar com o problema do Caixeiro Viajante em condições reais e complexas, mostrando eficiência na busca por soluções.

Referências

DORIGO, M.; STÜTZLE, T. Ant Colony Optimization. Cambridge: MIT Press, 2004.

TSPLIB95. Disponível em: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.

PYTHON SOFTWARE FOUNDATION. Python Documentation: Random Module. Disponível em: https://docs.python.org/3/library/random.html.