

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 otimiza 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:

```
Tempo de execução: 0.16013407707214355 segundos
[0, 9, 19, 17, 21, 13, 14, 16, 3, 15, 24, 18, 7, 23, 12, 4, 25, 28, 2, 8, 5, 20, 27, 1, 11, 10, 6, 26, 22, 0] 3559
[1, 4, 8, 5, 11, 27, 26, 23, 0, 22, 12, 15, 3, 18, 14, 10, 21, 16, 17, 13, 24, 7, 20, 28, 25, 2, 19, 9, 6, 1] 3238
[2, 19, 17, 13, 21, 10, 24, 7, 9, 16, 3, 26, 15, 14, 18, 6, 22, 0, 25, 28, 27, 5, 11, 8, 12, 23, 20, 1, 4, 2] 3561
[3, 6, 22, 24, 14, 17, 20, 23, 0, 18, 15, 26, 5, 7, 1, 27, 8, 19, 9, 12, 4, 25, 28, 2, 11, 10, 21, 13, 16, 3] 3912
[4, 25, 27, 0, 5, 23, 15, 9, 10, 21, 13, 16, 18, 24, 14, 3, 19, 2, 28, 1, 17, 20, 7, 11, 12, 22, 6, 26, 8, 4] 3723
[5, 11, 8, 4, 23, 26, 18, 12, 17, 16, 21, 14, 3, 15, 24, 10, 9, 19, 25, 28, 1, 20, 27, 22, 6, 7, 0, 13, 2, 5] 3626
[6, 10, 14, 3, 28, 1, 12, 8, 7, 27, 25, 2, 20, 4, 5, 23, 26, 24, 18, 16, 21, 13, 17, 9, 19, 15, 0, 11, 22, 6] 3602
[7, 26, 23, 15, 6, 18, 9, 12, 19, 16, 13, 14, 20, 5, 11, 27, 22, 0, 4, 8, 28, 25, 1, 17, 21, 10, 24, 2, 3, 7] 3768
[8, 7, 1, 19, 14, 20, 4, 9, 10, 21, 13, 16, 17, 18, 27, 12, 11, 25, 28, 2, 24, 23, 3, 15, 26, 22, 6, 0, 5, 8] 3985
[9, 17, 20, 8, 25, 4, 2, 28, 1, 27, 23, 11, 7, 26, 24, 19, 10, 6, 18, 3, 14, 21, 16, 13, 15, 22, 0, 5, 12, 9] 3611
[10, 12, 1, 26, 20, 14, 3, 21, 17, 13, 18, 15, 7, 11, 5, 8, 9, 28, 2, 19, 24, 22, 27, 0, 23, 25, 4, 6, 16, 10] 4161
[11, 5, 28, 9, 17, 13, 21, 16, 10, 14, 24, 22, 6, 18, 26, 23, 15, 20, 1, 25, 4, 8, 2, 19, 7, 27, 0, 12, 3, 11] 3416
[12, 20, 19, 18, 3, 10, 16, 13, 9, 4, 11, 0, 26, 15, 27, 7, 8, 1, 25, 2, 28, 5, 23, 14, 17, 24, 21, 22, 6, 12] 4029
[13, 9, 18, 16, 3, 19, 5, 8, 25, 28, 20, 26, 23, 15, 24, 12, 4, 1, 0, 22, 27, 7, 11, 14, 17, 21, 10, 6, 2, 13] 4268
[14, 1, 19, 17, 13, 24, 18, 0, 26, 7, 23, 25, 9, 3, 12, 4, 2, 28, 8, 5, 11, 20, 27, 22, 15, 6, 10, 21, 16, 14] 3466
[15, 18, 1, 20, 14, 16, 21, 3, 17, 13, 10, 19, 9, 12, 25, 8, 11, 5, 26, 24, 0, 22, 7, 4, 28, 27, 23, 2, 6, 15] 4165
[16, 13, 21, 10, 15, 19, 9, 17, 12, 20, 1, 25, 4, 11, 5, 8, 23, 28, 2, 0, 27, 7, 22, 26, 3, 14, 18, 6] 2866
[17, 13, 21, 16, 3, 18, 14, 9, 19, 1, 12, 23, 27, 5, 8, 4, 26, 7, 20, 0, 22, 24, 10, 6, 15, 28, 25, 11, 2, 17] 3525
[18, 6, 15, 9, 14, 3, 28, 25, 4, 20, 1, 11, 5, 19, 12, 26, 10, 16, 13, 17, 21, 24, 7, 27, 0, 22, 23, 8, 2, 18] 3768
[19, 15, 24, 12, 7, 18, 13, 17, 3, 21, 16, 10, 26, 9, 14, 20, 11, 25, 28, 1, 5, 4, 8, 2, 0, 23, 27, 22, 6, 19] 3897
[20, 17, 13, 24, 19, 2, 28, 8, 4, 5, 27, 0, 15, 9, 16, 14, 3, 18, 7, 23, 12, 1, 25, 11, 10, 21, 26, 6, 22, 20] 3974
[21, 17, 16, 13, 14, 10, 24, 9, 8, 19, 4, 25, 12, 22, 23, 5, 7, 1, 20, 28, 2, 26, 15, 18, 3, 6, 0, 27, 11, 21] 4117
[22, 26, 15, 7, 23, 28, 9, 0, 8, 19, 14, 3, 6, 18, 17, 24, 12, 5, 20, 27, 4, 25, 2, 1, 11, 13, 21, 16, 10, 22] 4056
[23, 1, 7, 4, 0, 24, 14, 9, 19, 20, 2, 25, 28, 8, 27, 18, 15, 10, 21, 16, 13, 17, 12, 22, 26, 6, 3, 5, 11, 23] 3821
[24, 18, 9, 3, 12, 23, 15, 27, 5, 8, 28, 4, 20, 1, 17, 21, 13, 16, 10, 19, 14, 6, 26, 7, 0, 22, 11, 25, 2, 24] 3591
[25, 28, 27, 5, 8, 0, 12, 26, 23, 1, 20, 4, 9, 14, 3, 17, 13, 21, 18, 24, 15, 22, 7, 19, 10, 16, 11, 6, 2, 25] 4079
[26, 7, 19, 9, 3, 21, 17, 13, 10, 12, 27, 0, 23, 28, 25, 20, 8, 4, 5, 11, 1, 14, 18, 24, 22, 6, 15, 2, 16, 26] 3853
[27, 26, 15, 0, 22, 18, 21, 16, 17, 14, 10, 3, 6, 28, 4, 2, 8, 5, 19, 7, 23, 11, 12, 9, 13, 25, 1, 20, 27] 4125
[28, 25, 2, 19, 15, 18, 17, 3, 9, 27, 12, 11, 8, 0, 1, 20, 4, 5, 23, 26, 22, 6, 14, 21, 13, 24, 10, 16, 7, 28] 3739
```

- Com 50 iterações:

```

Joaquim@MacBook-Air de João: Trabalho Final Heurística % python3 main.py
Tempo de execução: 0.7711360454559326 segundos
[0, 1, 20, 12, 26, 22, 6, 10, 19, 9, 18, 3, 13, 16, 21, 14, 17, 24, 15, 23, 7, 27, 11, 8, 4, 25, 28, 2, 5, 0] 2822
[1, 20, 4, 8, 5, 12, 23, 15, 7, 26, 22, 6, 10, 21, 13, 16, 17, 14, 3, 18, 24, 19, 9, 0, 27, 11, 2, 28, 25, 1] 2684
[2, 28, 25, 8, 5, 4, 20, 19, 1, 27, 0, 23, 12, 3, 13, 18, 14, 24, 10, 9, 16, 17, 21, 11, 7, 26, 15, 22, 6, 2] 3486
[3, 9, 14, 17, 21, 16, 13, 10, 26, 23, 15, 22, 24, 18, 12, 19, 1, 20, 4, 8, 25, 11, 5, 27, 0, 7, 2, 28, 6, 3] 3081
[4, 8, 9, 3, 14, 19, 17, 16, 10, 21, 13, 24, 22, 12, 28, 25, 2, 5, 11, 20, 1, 23, 15, 26, 7, 27, 0, 18, 6, 4] 3486
[5, 11, 8, 2, 28, 25, 4, 20, 17, 13, 21, 16, 3, 9, 19, 1, 26, 7, 15, 12, 10, 24, 18, 14, 23, 0, 22, 6, 27, 5] 3058
[6, 10, 21, 13, 14, 3, 9, 12, 20, 4, 5, 27, 7, 26, 23, 15, 18, 24, 19, 2, 25, 28, 1, 11, 8, 0, 22, 17, 16, 6] 3110
[7, 23, 5, 11, 4, 9, 19, 12, 3, 21, 13, 17, 16, 10, 24, 15, 1, 2, 28, 25, 8, 0, 27, 22, 18, 6, 14, 26, 20, 7] 3301
[8, 4, 1, 20, 0, 12, 26, 23, 7, 19, 9, 3, 14, 6, 10, 21, 16, 17, 13, 15, 18, 24, 22, 27, 5, 11, 2, 28, 25, 8] 2865
[9, 3, 14, 21, 16, 13, 17, 24, 0, 23, 7, 19, 1, 20, 27, 11, 8, 28, 25, 4, 12, 10, 6, 18, 26, 22, 15, 2, 5] 3342
[10, 24, 18, 14, 17, 13, 21, 16, 9, 3, 26, 7, 12, 11, 27, 19, 0, 23, 22, 6, 15, 1, 20, 5, 4, 8, 2, 28, 25, 10] 3327
[11, 27, 0, 20, 26, 7, 24, 22, 15, 6, 18, 9, 14, 10, 21, 16, 13, 17, 3, 28, 2, 5, 8, 4, 25, 1, 19, 12, 23, 11] 3091
[12, 20, 19, 9, 23, 26, 7, 0, 22, 15, 18, 24, 14, 6, 10, 17, 3, 21, 13, 16, 5, 11, 8, 25, 4, 28, 2, 1, 27, 12] 3208
[13, 17, 9, 14, 24, 21, 16, 6, 10, 12, 3, 19, 8, 20, 23, 5, 11, 2, 28, 25, 4, 1, 15, 18, 22, 26, 7, 0, 27, 13] 3688
[14, 9, 3, 19, 12, 27, 0, 23, 15, 26, 7, 22, 6, 2, 28, 25, 4, 11, 5, 20, 1, 8, 10, 24, 21, 13, 16, 17, 18, 14] 3125
[15, 9, 3, 14, 10, 6, 18, 24, 12, 7, 26, 22, 23, 0, 27, 5, 11, 1, 2, 28, 25, 4, 8, 20, 19, 17, 13, 21, 16, 15] 2746
[16, 21, 13, 15, 18, 24, 22, 6, 10, 19, 9, 12, 28, 2, 1, 8, 5, 4, 25, 0, 20, 26, 7, 23, 27, 11, 3, 27, 14, 3] 3245
[17, 16, 13, 24, 10, 3, 14, 26, 15, 23, 7, 5, 8, 4, 2, 28, 1, 20, 0, 27, 11, 19, 9, 12, 18, 22, 6, 21, 25, 17] 3458
[18, 14, 3, 17, 13, 12, 9, 19, 16, 21, 10, 24, 15, 26, 23, 0, 27, 7, 22, 6, 2, 28, 25, 4, 20, 1, 8, 5, 11, 18] 2973
[19, 28, 12, 15, 26, 7, 27, 0, 11, 8, 4, 5, 1, 20, 14, 24, 10, 17, 16, 13, 21, 18, 23, 25, 2, 6, 22, 3, 9, 19] 3476
[20, 7, 23, 15, 0, 27, 11, 4, 25, 28, 2, 19, 9, 12, 1, 8, 5, 26, 3, 18, 24, 10, 14, 13, 17, 16, 21, 6, 22, 20] 3055
[21, 13, 17, 14, 18, 24, 12, 20, 11, 23, 15, 22, 6, 16, 10, 3, 19, 9, 27, 0, 25, 1, 5, 8, 4, 28, 2, 26, 7, 21] 3534
[22, 26, 0, 5, 27, 11, 8, 4, 20, 2, 28, 25, 1, 19, 9, 3, 17, 21, 13, 16, 14, 18, 15, 23, 7, 12, 10, 24] 2235
[23, 26, 1, 20, 12, 9, 10, 24, 22, 6, 3, 19, 14, 17, 13, 21, 16, 18, 15, 0, 27, 7, 11, 5, 8, 2, 28, 25, 4, 23] 3025
[24, 10, 21, 16, 13, 17, 19, 3, 9, 14, 12, 23, 0, 7, 26, 15, 20, 5, 11, 4, 8, 2, 28, 1, 25, 27, 22, 6, 18, 24] 2842
[25, 28, 5, 11, 27, 4, 8, 0, 1, 20, 19, 7, 15, 12, 9, 18, 6, 26, 23, 14, 3, 21, 17, 13, 16, 10, 24, 22, 2, 25] 3393
[26, 3, 6, 18, 15, 23, 5, 11, 4, 8, 27, 0, 1, 20, 28, 25, 19, 9, 12, 10, 14, 13, 16, 21, 17, 24, 22, 7, 2, 26] 3403
[27, 11, 4, 8, 28, 1, 5, 0, 23, 26, 7, 15, 18, 14, 13, 16, 9, 3, 21, 10, 24, 17, 6, 2, 19, 20, 12, 22, 25, 27] 3819
[28, 4, 20, 1, 0, 27, 11, 8, 2, 6, 17, 13, 14, 9, 15, 12, 23, 22, 26, 7, 24, 10, 21, 16, 18, 3, 19, 5, 25, 28] 3382

```

• Com 100 iterações:

```

Joaquim@MacBook-Air de João: Trabalho Final Heurística % python3 main.py
Tempo de execução: 1.545050859451294 segundos
[0, 9, 3, 14, 26, 7, 18, 24, 22, 15, 23, 12, 11, 8, 4, 5, 27, 20, 1, 28, 25, 2, 19, 21, 10, 6, 16, 13, 17, 0] 3361
[1, 27, 0, 12, 15, 26, 23, 7, 22, 24, 10, 21, 16, 13, 17, 14, 18, 9, 19, 3, 6, 2, 28, 25, 4, 8, 11, 5, 20, 1] 2717
[2, 28, 25, 11, 5, 4, 8, 27, 0, 20, 26, 7, 15, 24, 12, 9, 19, 3, 21, 14, 17, 16, 13, 10, 6, 22, 23, 18, 1, 2] 2868
[3, 9, 23, 7, 26, 15, 24, 10, 14, 17, 21, 13, 16, 18, 1, 0, 27, 11, 8, 4, 25, 28, 2, 19, 12, 20, 5, 6, 22, 3] 3045
[4, 8, 25, 28, 23, 5, 27, 26, 22, 12, 1, 19, 9, 3, 14, 21, 13, 17, 16, 10, 24, 18, 6, 15, 7, 20, 11, 2, 0, 4] 3250
[5, 4, 8, 11, 0, 23, 12, 9, 19, 1, 25, 28, 2, 6, 10, 14, 3, 18, 15, 17, 16, 21, 13, 24, 26, 7, 27, 22, 20, 5] 3113
[6, 15, 18, 24, 7, 26, 19, 9, 12, 3, 14, 17, 21, 13, 16, 10, 23, 0, 5, 27, 20, 1, 4, 8, 28, 2, 25, 11, 22, 6] 2995
[7, 27, 12, 17, 21, 13, 14, 24, 15, 18, 3, 9, 19, 1, 20, 8, 4, 28, 25, 2, 6, 22, 0, 23, 26, 10, 16, 11, 5, 7] 3496
[8, 4, 28, 25, 2, 20, 19, 14, 3, 21, 16, 13, 17, 18, 24, 15, 23, 7, 26, 22, 6, 10, 12, 9, 0, 1, 11, 27, 5, 8] 2842
[9, 24, 10, 6, 26, 23, 7, 27, 0, 20, 1, 19, 12, 4, 8, 5, 25, 28, 2, 11, 17, 13, 21, 16, 18, 14, 3, 15, 22, 9] 3245
[10, 14, 3, 7, 23, 26, 15, 0, 27, 5, 11, 12, 18, 13, 17, 16, 24, 22, 6, 2, 25, 28, 4, 8, 20, 1, 19, 9, 21, 10] 3177
[11, 5, 27, 7, 26, 9, 19, 1, 20, 0, 12, 4, 8, 25, 28, 2, 6, 22, 23, 15, 24, 18, 3, 10, 13, 17, 16, 21, 14, 11] 3040
[12, 19, 9, 13, 21, 16, 10, 17, 3, 14, 18, 24, 15, 23, 26, 22, 6, 2, 25, 28, 8, 4, 20, 1, 7, 27, 0, 5, 11, 12] 2963
[13, 17, 16, 10, 24, 15, 18, 14, 3, 9, 12, 1, 20, 0, 27, 7, 23, 26, 22, 6, 2, 28, 25, 4, 5, 11, 8, 19, 21, 13] 2757
[14, 18, 23, 8, 4, 20, 1, 25, 2, 28, 9, 19, 16, 3, 10, 21, 13, 17, 15, 27, 5, 11, 7, 26, 12, 24, 22, 6, 0, 14] 3500
[15, 18, 3, 14, 12, 23, 0, 27, 5, 25, 28, 4, 8, 19, 9, 26, 7, 22, 6, 2, 1, 20, 11, 16, 10, 21, 13, 17, 24, 15] 3416
[16, 13, 17, 14, 3, 10, 21, 18, 24, 0, 27, 7, 22, 23, 26, 15, 12, 9, 19, 1, 28, 25, 8, 4, 5, 11, 20, 2, 6, 16] 3039
[17, 9, 19, 23, 26, 22, 15, 6, 10, 14, 18, 3, 16, 13, 21, 24, 12, 1, 20, 4, 8, 5, 27, 0, 7, 11, 25, 28, 2, 17] 3104
[18, 21, 13, 17, 14, 15, 23, 1, 4, 25, 28, 12, 3, 6, 10, 26, 7, 11, 5, 27, 0, 24, 9, 19, 20, 8, 2, 22, 16, 18] 3930
[19, 9, 15, 24, 22, 23, 12, 20, 1, 28, 2, 4, 5, 11, 8, 25, 27, 7, 26, 3, 17, 13, 16, 21, 10, 6, 14, 18, 0, 19] 2980
[20, 25, 5, 11, 8, 28, 2, 1, 0, 22, 12, 14, 18, 24, 15, 26, 23, 3, 9, 19, 21, 13, 17, 16, 10, 6, 7, 27, 4, 20] 3041
[21, 13, 16, 17, 3, 14, 18, 15, 23, 26, 22, 6, 10, 12, 1, 20, 5, 11, 8, 25, 4, 27, 0, 9, 19, 2, 28, 24, 7, 21] 3163
[22, 23, 15, 26, 7, 11, 5, 27, 0, 8, 25, 28, 19, 9, 3, 17, 14, 18, 24, 10, 16, 13, 21, 4, 1, 20, 12, 2, 6, 22] 3299
[23, 22, 26, 15, 14, 18, 0, 7, 27, 20, 1, 11, 5, 8, 25, 28, 2, 19, 9, 3, 12, 24, 10, 21, 13, 16, 17, 6, 4, 23] 3138
[24, 15, 26, 14, 10, 3, 18, 23, 27, 11, 5, 4, 0, 22, 7, 12, 9, 19, 1, 20, 25, 8, 2, 28, 13, 17, 16, 21, 6] 3010
[25, 1, 19, 14, 3, 9, 12, 23, 26, 7, 22, 6, 18, 15, 24, 10, 13, 16, 21, 17, 20, 8, 4, 5, 0, 27, 11, 2, 28, 25] 2704
[26, 27, 11, 20, 1, 2, 28, 25, 4, 8, 12, 14, 3, 9, 10, 23, 22, 6, 18, 19, 0, 7, 5, 24, 17, 13, 21, 16, 15, 26] 3438
[27, 25, 28, 2, 1, 20, 0, 23, 26, 6, 10, 24, 15, 22, 7, 12, 3, 17, 13, 21, 16, 14, 9, 19, 18, 11, 5, 8, 4, 27] 2873
[28, 25, 4, 8, 11, 5, 27, 15, 18, 14, 6, 16, 21, 10, 24, 13, 17, 20, 0, 12, 9, 19, 1, 2, 3, 7, 26, 22, 23, 28] 3373

```

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.ifl.uni-heidelberg.de/software/TSPLIB95/>.

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