

Otimização por colônia de formigas

October 3, 2023

1 Otimização por Colônia de Formigas (ACO) para Resolução de Problemas de Roteamento

O algoritmo de Otimização por Colônia de Formigas (ACO, do inglês Ant Colony Optimization) é uma técnica bioinspirada que simula o comportamento de formigas na busca por caminhos entre uma fonte de alimento e o formigueiro. As formigas depositam uma substância chamada feromônio nos caminhos que percorrem, e esse feromônio serve como um sinalizador para outras formigas sobre a qualidade desse caminho. Com o tempo, o feromônio evapora, reduzindo sua intensidade, mas os caminhos mais curtos e, portanto, mais rapidamente percorridos por múltiplas formigas, acabam sendo reforçados.

Neste código, aplicamos o ACO para resolver um problema de roteamento em um grafo, buscando o caminho mais curto entre os vértices. Utilizamos um grafo pré-definido, e a intensidade do feromônio é atualizada em cada iteração com base nos caminhos encontrados.

```
[4]: import numpy as np
import networkx as nx
import matplotlib.pyplot as plt

# Grafo (distâncias)
graph = {
    'A': {'A': 0, 'B': 2, 'C': 10, 'D': 8, 'E': 3},
    'B': {'A': 1, 'B': 0, 'C': 2, 'D': 5, 'E': 7},
    'C': {'A': 9, 'B': 1, 'C': 0, 'D': 3, 'E': 6},
    'D': {'A': 10, 'B': 4, 'C': 3, 'D': 0, 'E': 2},
    'E': {'A': 2, 'B': 7, 'C': 5, 'D': 1, 'E': 0}
}

# Parâmetros
alpha = 1
beta = 1
rho = 0.5
initial_pheromone = 2

[5]: # Inicialização de feromônio
pheromones = {i: {j: initial_pheromone for j in graph[i]} for i in graph}

def plot_graph(path=None):
```

```

G = nx.Graph()
for i in graph:
    for j in graph[i]:
        G.add_edge(i, j, weight=pheromones[i][j]) # Mostrando o feromônio
↪em vez da distância

pos = nx.spring_layout(G)
edge_labels = {(i, j): '{:.2f}'.format(G[i][j]['weight']) for i, j in G.
↪edges()}
nx.draw(G, pos, with_labels=True, node_size=1500, node_color='skyblue')
nx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels)
if path:
    path_edges = [(path[i], path[i+1]) for i in range(len(path) - 1)]
    nx.draw_networkx_edges(G, pos, edgelist=path_edges, edge_color='r',
↪width=2)
plt.show()

def calculate_probability(k, available_nodes):
    total = sum([(pheromones[k][j]**alpha * (1.0 / graph[k][j])**beta) for j in
↪available_nodes])
    probs = [(pheromones[k][j]**alpha * (1.0 / graph[k][j])**beta) / total for
↪j in available_nodes]
    return probs

def aco_simulation(start):
    path = [start]
    available_nodes = list(graph.keys())
    available_nodes.remove(start)

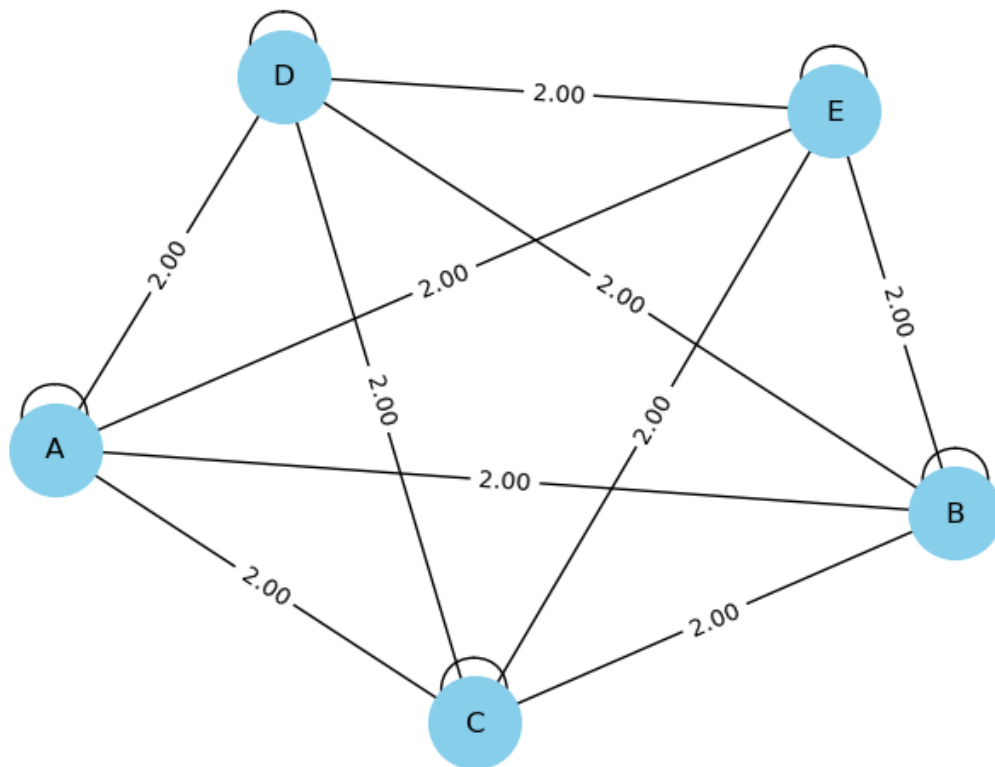
    current_node = start
    while available_nodes:
        next_node_probs = calculate_probability(current_node, available_nodes)
        next_node = np.random.choice(available_nodes, 1, p=next_node_probs)[0]
        path.append(next_node)
        available_nodes.remove(next_node)
        current_node = next_node

    return path

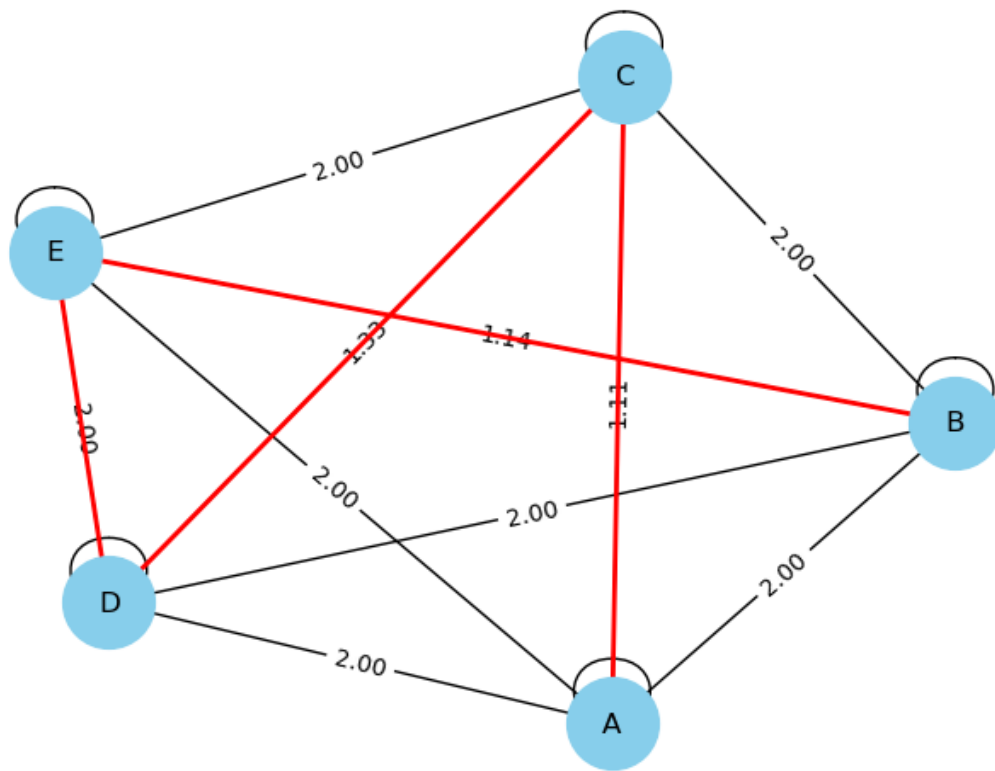
# Atualização de feromônio
def update_pheromones(path):
    for i in range(len(path)-1):
        k, j = path[i], path[i+1]
        pheromones[k][j] = (1 - rho) * pheromones[k][j] + (1 / graph[k][j])
        pheromones[j][k] = (1 - rho) * pheromones[j][k] + (1 / graph[j][k])

```

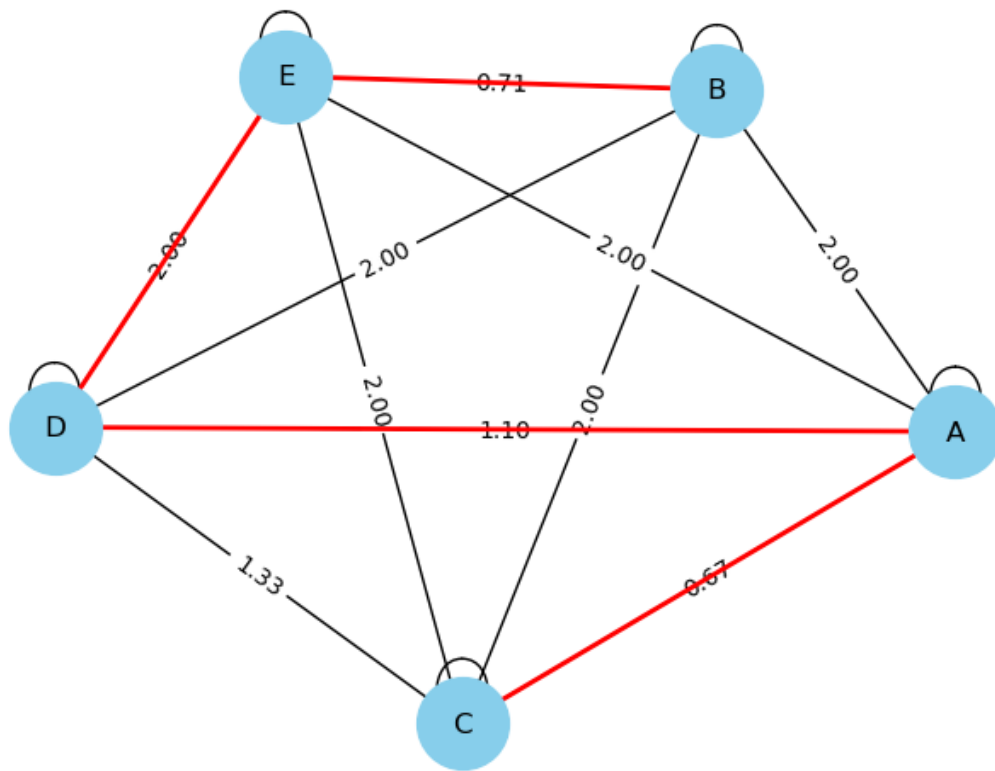
```
[6]: # Executando a simulação e plotando
plot_graph()
for start_node in ['B', 'C', 'D', 'E']:
    path = aco_simulation(start_node)
    print(f"Simulação partindo de {start_node}: Rota = {path}")
    update_pheromones(path)
    plot_graph(path)
```



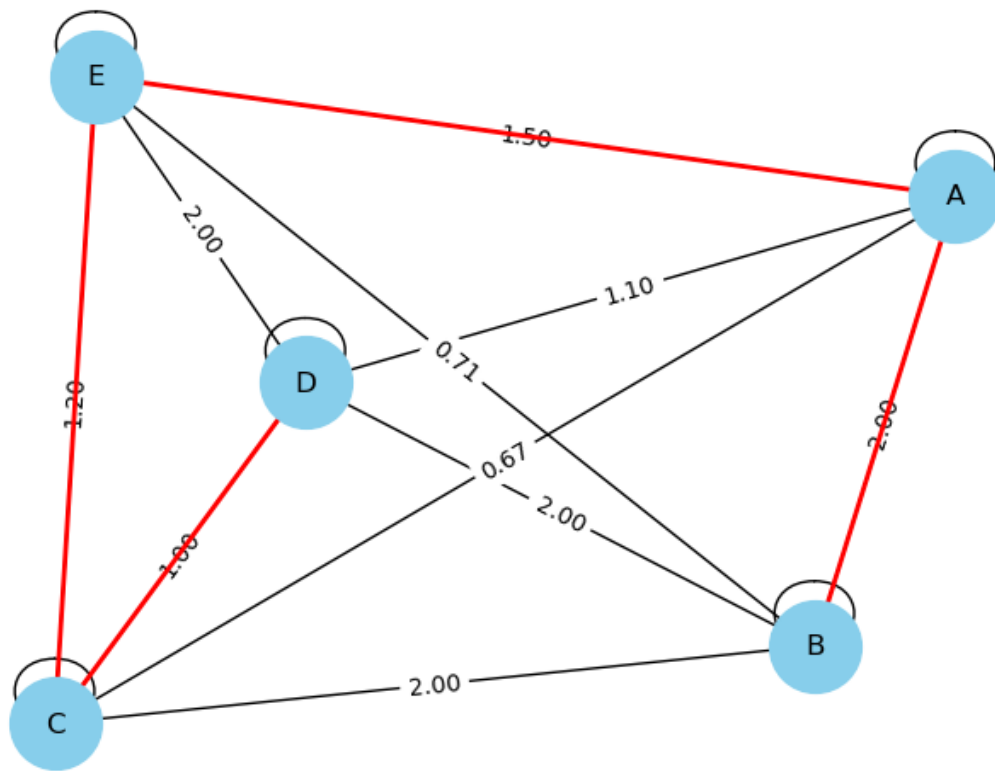
Simulação partindo de B: Rota = ['B', 'E', 'D', 'C', 'A']



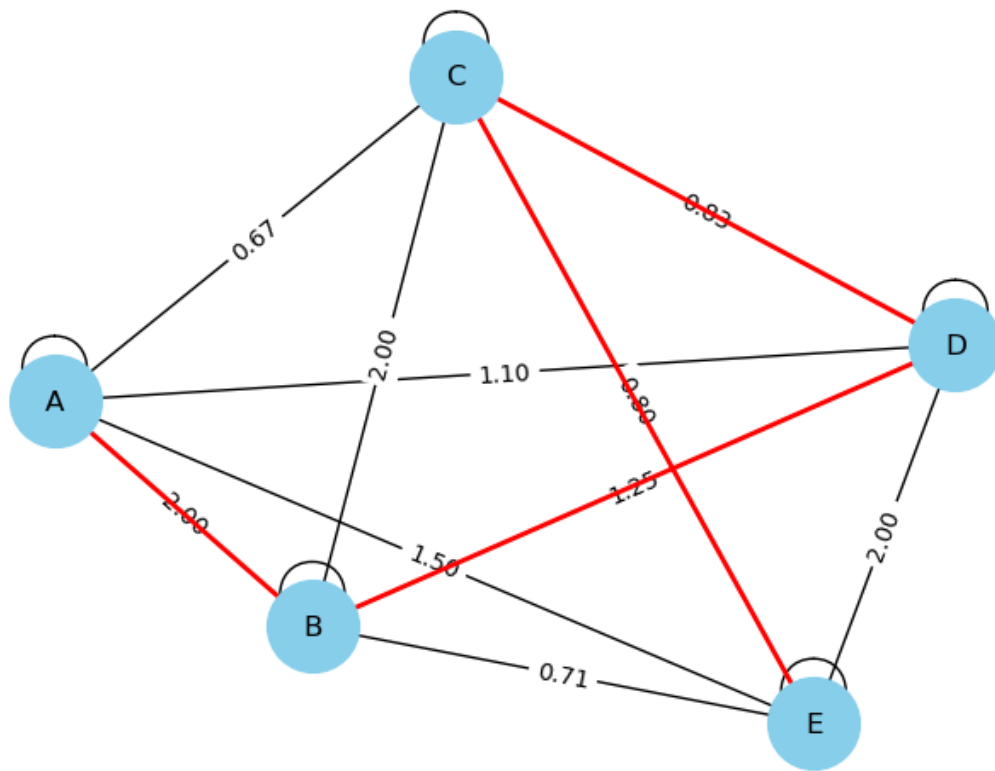
Simulação partindo de C: Rota = ['C', 'A', 'D', 'E', 'B']



Simulação partindo de D: Rota = ['D', 'C', 'E', 'A', 'B']



Simulação partindo de E: Rota = ['E', 'C', 'D', 'B', 'A']



[]: