UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS ITAPAJÉ
DISCIPLINA DE COMPUTAÇÃO EVOLUTIVA

Trabalho Computacional

Ana Lívia Sousa - 536158
Maverick Alekyne - 541062

Itapajé, Ceará, Setembro de 2024

Sumário

1 Introdução

O Problema do Caixeiro Viajante (PCV) é um problema de otimização combinatória onde o objetivo é encontrar o caminho mais curto que passa por um conjunto de cidades exatamente uma vez e retorna à cidade de origem. Este problema é NP-difícil e, portanto, não há uma solução eficiente conhecida para todos os casos.

O Algoritmo de Otimização por Colônia de Formigas (ACO) é uma técnica inspirada no comportamento das formigas na natureza para resolver o PCV. O ACO utiliza o conceito de feromônio para guiar as formigas na construção de soluções e melhorar continuamente a qualidade das soluções encontradas.

2 Princípios do ACO

O ACO é baseado na observação de que formigas reais depositam feromônio ao se moverem entre os nós de um grafo. Este feromônio influencia as decisões de outras formigas sobre quais caminhos seguir. O algoritmo ACO usa uma abordagem semelhante:

- **Inicialização:** As formigas são distribuídas aleatoriamente sobre as cidades e o feromônio é inicializado com um valor baixo.
- **Construção da Solução:** Cada formiga constrói uma solução (um ciclo) movendo-se de uma cidade para outra com probabilidade proporcional à quantidade de feromônio no caminho e à visibilidade (reciprocidade da distância).
- **Atualização do Feromônio:** Após todas as formigas terem completado seus ciclos, o feromônio é atualizado. O feromônio é evaporado ao longo do tempo e depositado nas arestas que foram usadas pelas formigas que encontraram boas soluções.
- **Evaporação e Intensificação:** A evaporação do feromônio ajuda a evitar a convergência prematura e a intensificação do feromônio nas melhores soluções guiam as formigas para melhores caminhos.

3 Aplicação ao PCV

Para resolver o PCV com o ACO, o algoritmo é aplicado da seguinte forma:

- Cada cidade é representada como um nó em um grafo completo, onde as arestas são ponderadas pela distância entre as cidades.

- As formigas começam em cidades aleatórias e constroem suas rotas baseadas na quantidade de feromônio e na visibilidade das cidades.
- O feromônio é atualizado após cada iteração, reforçando os caminhos que levaram a soluções de menor custo e evaporando o feromônio das arestas menos utilizadas.
- O processo é repetido por várias iterações até que uma solução satisfatória seja encontrada ou até que o critério de parada seja atingido.

4 Conclusão

O Algoritmo de Otimização por Colônia de Formigas oferece uma abordagem robusta e flexível para resolver o Problema do Caixeiro Viajante. Ele é especialmente eficaz para problemas grandes e complexos, onde métodos exatos se tornam impraticáveis. O uso de estratégias como a atualização de feromônio e a evaporação permite que o ACO encontre boas soluções para o PCV, aproveitando o comportamento coletivo das formigas para explorar e explorar eficientemente o espaço de soluções.

5 Implementação do codigo

6 Classe `AntColony`

A classe `AntColony` é a base da implementação e contém os seguintes componentes:

6.1 Construtor (`init`)

Inicializa a matriz de distâncias entre cidades, a matriz de feromônio e define os parâmetros do algoritmo. Os parâmetros são:

- `distances`: Matriz de distâncias entre as cidades.
- `n_ants`: Número de formigas por iteração.
- `n_best`: Número de melhores formigas que depositam feromônio.
- `n_iterations`: Número de iterações.
- `decay`: Taxa de evaporação do feromônio.

- `alpha`: Exponente do feromônio.
- `beta`: Exponente da distância.

6.2 Método `run`

Executa o ACO por um número definido de iterações. Em cada iteração, as formigas geram caminhos e o feromônio é espalhado com base nas melhores soluções encontradas. O melhor caminho e a distância são atualizados.

6.3 Método `spread_pheromone`

Atualiza a matriz de feromônio com base nas melhores rotas encontradas. As arestas usadas por essas rotas recebem um incremento no feromônio.

6.4 Método `gen_path_dist`

Calcula a distância total de um caminho, incluindo o retorno ao ponto inicial.

6.5 Método `gen_all_paths`

Gera todos os caminhos possíveis das formigas e calcula suas distâncias.

6.6 Método `gen_path`

Constrói um caminho a partir de um ponto inicial, movendo-se para outras cidades não visitadas, de acordo com a probabilidade ponderada pelo feromônio e pela visibilidade.

6.7 Método `pick_move`

Seleciona o próximo movimento com base na probabilidade, que é calculada usando o feromônio atual e a distância.

7 Código da Implementação

A seguir, o código Python para a classe `AntColony`:
[language=Python, caption=Implementação do ACO para o Problema do Caixeiro Viajante
backgroundcolor=lightgray]
import numpy as np

from numpy.random
import choice as np$_{c}hoice$
class AntColony: def init(self, distances, n$_{a}nts, n_{b}est, n_{i}terations, decay, alpha =$
$1, beta = 1)$:
self.distances = distances
self.pheromone = np.ones(self.distances.shape) /
len(distances)
self.all$_{i}nds = range(len(distances))$
self.n$_{a}nts = n_{a}nts$
self.n$_{b}est = n_{b}est$
self.n$_{i}terations = n_{i}terations$
self.decay = decay
self.alpha = alpha
self.beta = beta
def run(self):
all$_{t}ime_{s}hortest_{p}ath = None$
all$_{t}ime_{s}hortest_{d}istance = np.inf$
for i in range(self.n$_{i}terations)$:
all$_{p}aths = self.gen_{a}ll_{p}aths()$
self.spread$_{p}heromone(all_{p}aths, self.n_{b}est)$
shortest$_{p}ath = min(all_{p}aths, key = lambdax : x[1])$
print(f"Iteração i + 1: shortest$_{p}ath$")
if all$_{t}ime_{s}hortest_{p}athisNoneorshortest_{p}ath[1] <$
all$_{t}ime_{s}hortest_{d}istance$:
all$_{t}ime_{s}hortest_{p}ath = shortest_{p}ath[0]$
all$_{t}ime_{s}hortest_{d}istance = shortest_{p}ath[1]$
self.pheromone *= self.decay
return all$_{t}ime_{s}hortest_{p}ath, all_{t}ime_{s}hortest_{d}istance$
def spread$_{p}heromone(self, all_{p}aths, n_{b}est)$:
sorted$_{p}aths = sorted(all_{p}aths, key = lambdax : x[1])$
for path, dist in sorted$_{p}aths[: n_{b}est]$:
for i in range(len(path) - 1):
move = (path[i][1], path[i + 1][0])
if not np.isnan(self.distances[move]):
self.pheromone[move] += 1.0 / dist
move = (path[-1][1], path[0][0])
if not np.isnan(self.distances[move]):
self.pheromone[move] += 1.0 / dist
def gen$_{p}ath_{d}ist(self, path)$:
total$_{d}ist = 0$
for i in range(len(path) - 1):

$\text{total}_d ist+ = self.distances[path[i][1], path[i+1][0]]$
$\text{total}_d ist+ = self.distances[path[-1][1], path[0][0]]$
return $\text{total}_d ist$
def $\text{gen}_a ll_p aths(self)$:
$\text{all}_p aths = []$
for $_i nrange(self.n_a nts)$:
path = self.$\text{gen}_p ath(0)$
dist = self.$\text{gen}_p ath_d ist(path)$
$\text{all}_p aths.append((path, dist))$
return $\text{all}_p aths$
def $\text{gen}_p ath(self, start)$:
path = []
visited = set()
visited.add(start)
prev = start
for $_i nrange(len(self.distances) - 1)$:
move = self.$\text{pick}_m ove(self.pheromone[prev]$,
self.distances[prev], visited)
path.append((prev, move))
prev = move
visited.add(move)
path.append((prev, start))
return path
def $\text{pick}_m ove(self, pheromone, dist, visited)$:
pheromone = np.copy(pheromone)
pheromone[list(visited)] = 0
row = pheromone ** self.alpha * ((1.0 / dist) ** self.beta)
$\text{row}_s um = row.sum()$
if $\text{row}_s um == 0$:
return np.random.choice(self.$\text{all}_i nds$)
$\text{norm}_r ow = row/row_s um$
move = $\text{np}_c hoice(self.all_i nds, 1, p = norm_r ow)[0]$
return move
distances = np.array([
[np.inf, 2, 2, 5, 7],
[2, np.inf, 4, 8, 6],
[2, 4, np.inf, 1, 3],
[5, 8, 1, np.inf, 2],
[7, 6, 3, 2, np.inf]
])
Parâmetros do algoritmo

$n_ants = 500$
$n_best = 3$
$n_iterations = 1000$
decay = 0.5
alpha = 1
beta = 5
Criação da instância do AntColony
$ant_colony = AntColony(distances, n_ants, n_best, n_iterations,$
decay, alpha, beta)
Executando o algoritmo
$best_path, best_distance = ant_colony.run()$
Exibindo o melhor caminho e sua distância
print(f"Melhor caminho: $best_path$")
print(f"Distância do melhor caminho: $best_distance$")

8 Instruções para Executar o Código

Para executar o código Python fornecido, siga estas etapas:

1. **Instale as dependências**: Certifique-se de que o Python e o pacote `numpy` estão instalados. Você pode instalar o `numpy` usando o seguinte comando:

   ```
   pip install numpy
   ```

2. **Salve o código**: Copie o código Python fornecido e cole-o em um arquivo com a extensão `.py`, por exemplo, `aco.py`.

3. **Execute o código**: No terminal ou prompt de comando, navegue até o diretório onde o arquivo `aco.py` está salvo e execute o comando:

   ```
   python aco.py
   ```

4. **Verifique a saída**: O código imprimirá o melhor caminho encontrado e a distância do melhor caminho na saída padrão (terminal).

9 Conclusão

A implementação do ACO descrita aqui é projetada para encontrar o melhor caminho que minimiza a distância total percorrida ao visitar todas as cidades e retornar ao ponto de origem. A combinação de feromônio e visibilidade guia as formigas para soluções melhores ao longo das iterações.

Tabela de Resultados

Table 1: Resumo dos resultados obtidos com diferentes números de formigas

Número de Formigas	**Iteração Final**	**Melhor Caminho**
500	1000	[(0, 2), (2, 3), (3, 4), (4, 1), (1, 0)]
300	1000	[(0, 1), (1, 2), (2, 3), (3, 4), (4, 0)]
200	1000	[(0, 2), (2, 3), (3, 4), (4, 1), (1, 0)]
100	1000	[(0, 2), (2, 3), (3, 4), (4, 1), (1, 0)]

Observações

- Todos os caminhos encontrados repetem um padrão, o que pode sugerir que a matriz de distâncias pode estar influenciando o resultado.
- A diferença no número de formigas parece não ter impacto significativo nos caminhos encontrados ou nas distâncias, indicando que outros parâmetros podem precisar de ajuste.