Practica 5. Optimización por Colonia de Hormigas

Algoritmos Bioinspirados

Diego Castillo Reyes
Marthon Leobardo Yañez Martinez
Aldo Escamilla Resendiz

6 de junio de 2024

Índice general

1. Introducción

El Algoritmo de Optimización por Colonia de Hormigas (ACO, por sus siglas en inglés) es una técnica metaheurística inspirada en el comportamiento de las hormigas en la naturaleza. Este algoritmo, propuesto inicialmente por Marco Dorigo en la década de 1990, simula la forma en que las hormigas encuentran caminos óptimos entre su colonia y una fuente de alimento.

En la naturaleza, las hormigas depositan feromonas en su camino al buscar comida, creando un rastro químico que guía a otras hormigas. Con el tiempo, los caminos con más feromonas se vuelven más atractivos, y las hormigas tienden a seguir estos senderos más prometedores, lo que eventualmente conduce a la convergencia en el camino más corto.

El ACO utiliza este principio para resolver problemas de optimización combinatoria. En este algoritmo, una colonia de hormigas artificiales construye soluciones al problema, y durante este proceso, cada hormiga deposita feromonas en el entorno, influyendo en las decisiones de las hormigas subsiguientes. Los caminos que resultan en mejores soluciones reciben más feromonas, aumentando la probabilidad de ser seleccionados en iteraciones futuras.

2. Problema de Asignación Cuadrática

```python
'''
Resolver el problema de asignacion
cuadratica usando ACO.
'''
import numpy as np
import itertools

def selecciona_siguiente_ciudad(actual, no_visitadas, feromona, distancia, alpha, beta, epsilon=1e-10):
    feromona_actual = feromona[actual][no_visitadas] ** alpha
    visibilidad = (1.0 / (distancia[actual][no_visitadas] + epsilon)) ** beta
    probabilidades = feromona_actual * visibilidad

    # Chequear y manejar valores infinitos o NaN en probabilidades
    if np.any(np.isinf(probabilidades)) or np.any(np.isnan(probabilidades)):
        probabilidades = np.nan_to_num(probabilidades, nan=epsilon, posinf=epsilon, neginf=epsilon)

    suma_probabilidades = sum(probabilidades)

    # Chequear si la suma de probabilidades es cero o infinito para evitar NaNs
    if suma_probabilidades == 0 or np.isinf(suma_probabilidades) or np.isnan(suma_probabilidades):
        probabilidades = np.ones_like(probabilidades) / len(probabilidades)
    else:
        probabilidades /= suma_probabilidades

    return np.random.choice(no_visitadas, 1, p=probabilidades)[0]

def construye_camino(inicio, n_ciudades, feromona, distancia, alpha, beta):
    camino = [inicio]
    no_visitadas = list(range(n_ciudades))
    no_visitadas.remove(inicio)
```

```
    actual = inicio
    while no_visitadas:
        siguiente = selecciona_siguiente_ciudad(actual, no_visitadas, feromona, distancia, alpha, beta)
        camino.append(siguiente)
        no_visitadas.remove(siguiente)
        actual = siguiente

    return camino

def computa_costo(camino, distancia, flujo):
    costo = 0
    for i in range(len(camino)):
        for j in range(len(camino)):
            costo += flujo[i][j] * distancia[camino[i]][camino[j]]
    return costo

def actualiza_feromona(feromona, caminos, costos, evaporacion, q):
    feromona *= (1.0 - evaporacion)
    for camino, costo in zip(caminos, costos):
        for i in range(len(camino) - 1):
            feromona[camino[i]][camino[i+1]] += q / costo
        feromona[camino[-1]][camino[0]] += q / costo

def aco(distancia, flujo, n_hormigas, n_iteraciones, evaporacion, alpha=1, beta=5, q=1):
    n_ciudades = distancia.shape[0]
    feromona = np.ones(distancia.shape) / n_ciudades

    mejor_camino = None
    mejor_costo = float('inf')

    for _ in range(n_iteraciones):
        caminos = [construye_camino(np.random.choice(n_ciudades), n_ciudades, feromona, distancia, alpha, beta) for _ in range(n_hormigas)]
        costos = [computa_costo(camino, distancia, flujo) for camino in caminos]
        actualiza_feromona(feromona, caminos, costos, evaporacion, q)
        min_costo = min(costos)
        if min_costo < mejor_costo:
            mejor_costo = min_costo
            mejor_camino = caminos[costos.index(min_costo)]

    return mejor_camino, mejor_costo

def leer_matrices(file_path):
    with open(file_path, 'r') as file:
        lines = file.readlines()

    matrices = []
    current_matrix = []
    for line in lines:
        stripped_line = line.strip()
        if stripped_line:
            current_matrix.append(list(map(int, stripped_line.split())))
        else:
            if current_matrix:
                matrices.append(np.array(current_matrix))
                current_matrix = []
    if current_matrix:
        matrices.append(np.array(current_matrix))

    return matrices

def ANOVA(distancia, flujo, n_iteraciones, evaporacion, alphas, betas, n_hormigas):
    resultados = []
    combinaciones = list(itertools.product(n_hormigas, alphas, betas))

    for n_h, alpha, beta in combinaciones:
        mejor_camino, mejor_costo = aco(distancia, flujo, n_h, n_iteraciones, evaporacion, alpha, beta)
        resultados.append((n_h, alpha, beta, mejor_camino, mejor_costo))
```

```
99
100     return resultados
101
102 # Usar las matrices leídas
103 # ! Cambiar el path al archivo
104 # file_path = 'Algoritmos Bioinspirados/Practica 5/matricesProblemaQAP/matricesProblemaQAP/tai15.
    dat'
105 file_path = 'matricesProblemaQAP/matricesProblemaQAP/tai30.dat'
106 matrices = leer_matrices(file_path)
107 distancia = matrices[1]
108 flujo = matrices[2]
109
110 # Definir los parámetros
111 n_hormigas = [10, 25, 50, 100]
112 alphas = [50, 100, 200]
113 betas = [2, 10]
114 n_iteraciones = 100
115 evaporacion = 0.5
116
117 # Ejecutar experimentos
118 resultados = ANOVA(distancia, flujo, n_iteraciones, evaporacion, alphas, betas, n_hormigas)
119
120 # Imprimir resultados
121 for i, (n_h, alpha, beta, mejor_camino, costo) in enumerate(resultados):
122     print(f"{i + 1}. Número de hormigas: {n_h}, alpha: {alpha}, beta: {beta}, Mejor camino: {
    mejor_camino}, Costo: {costo}")
```

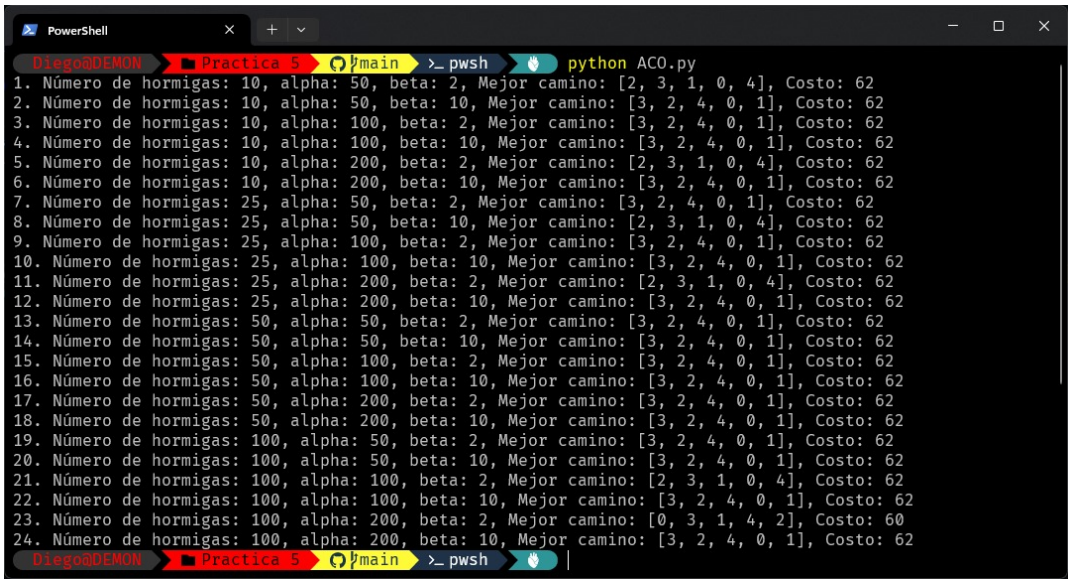

Figura 1: Pruebas de regresión

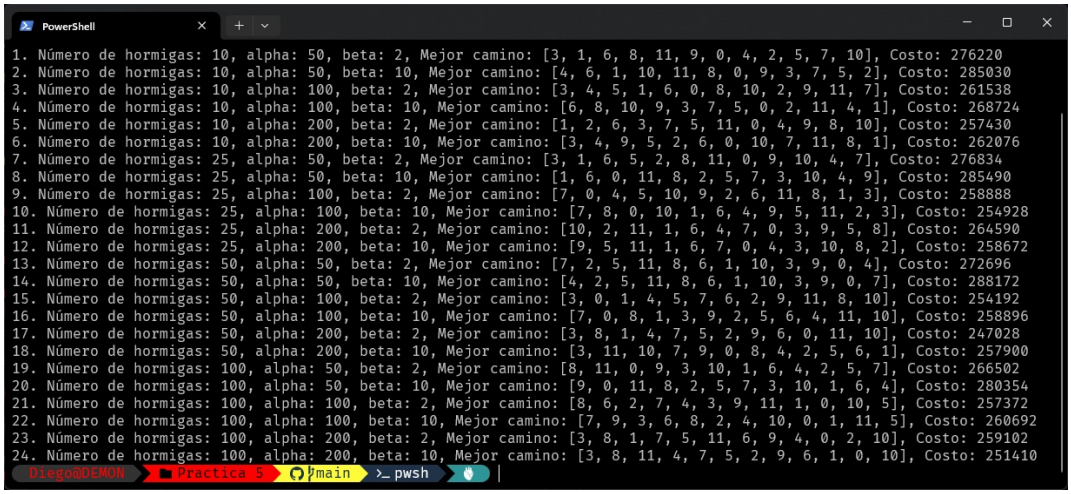

Figura 2: Pruebas de regresión

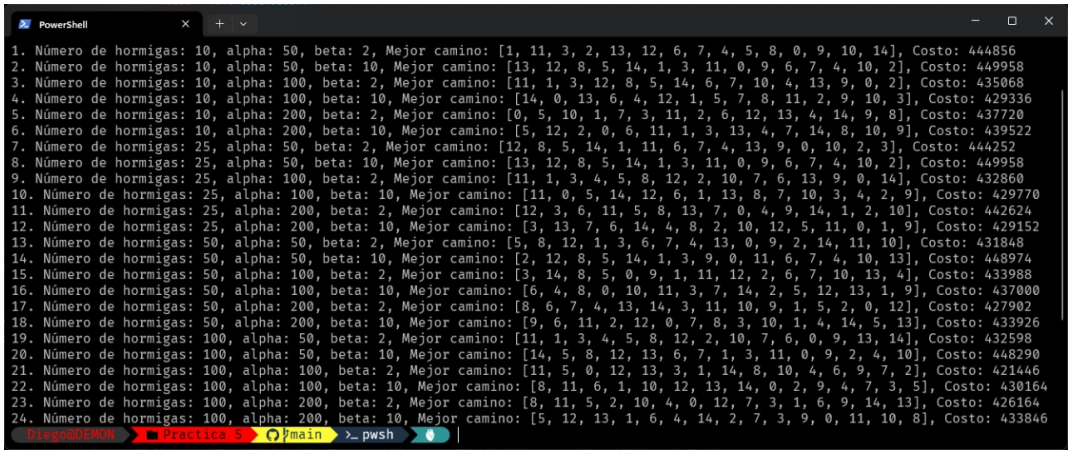

Figura 3: Pruebas de regresión

Figura 4: Pruebas de regresión

3. Lectura

Problema de Optimización Combinatoria Resuelto por ACO

El Algoritmo de Optimización por Colonia de Hormigas (ACO) resuelve problemas de optimización combinatoria convirtiéndolos en problemas de encontrar el camino más corto en un grafo ponderado. Formalmente, esto se describe como:

- **Grafo Ponderado** $G = (V, E)$: donde V es el conjunto de vértices y E es el conjunto de aristas.
- **Función de Costo** $c : E \to R^+$: Asigna un costo positivo a cada arista.
- **Soluciones** S: Conjuntos de caminos en el grafo.
- **Función Objetivo** $f : S \to R$: Evaluación del costo de cada solución.

Las hormigas artificiales construyen soluciones explorando el grafo, guiadas por niveles de feromonas que reflejan la calidad de soluciones previas.

Cuadro Comparativo de Versiones de ACO

Versión ACO	Descripción
Ant System (AS)	Primera versión básica de ACO. Las hormigas depositan feromonas proporcionalmente a la calidad de la solución encontrada.
Ant Colony System (ACS)	Introduce la actualización local de feromonas y un mecanismo de exploración más agresivo para evitar la convergencia prematura.
Max-Min Ant System (MMAS)	Establece límites máximo y mínimo a los niveles de feromonas, ayudando a prevenir la convergencia en soluciones subóptimas.
Rank-Based Ant System (RAS)	Asigna diferentes cantidades de feromonas según el ranking de las soluciones, favoreciendo las mejores soluciones encontradas.
Elitist Ant System (EAS)	Similar a AS, pero con un sesgo adicional hacia las mejores soluciones almacenadas en una memoria global.
Hybrid Ant System (HAS)	Combina ACO con otros métodos de optimización como el recocido simulado o algoritmos genéticos para mejorar la exploración y explotación del espacio de soluciones.

Cuadro 1: Comparación de versiones de ACO

El último congreso registrado es el .ªNTS 2022 - International Conference on Swarm Intelligence". Los tópicos abordados incluyeron:

- Algoritmos bioinspirados
- Optimización multiobjetivo
- Aplicaciones industriales de ACO
- Modelado y análisis teórico de ACO

Las pláticas cubrieron avances recientes en teoría de feromonas, aplicaciones prácticas y nuevas variantes híbridas del algoritmo.

Aplicaciones Relevantes

Tipo de Aplicación	Descripción
Mono-objetivo	Resolución del problema del viajero, optimización de rutas en redes de telecomunicaciones, y diseño de circuitos electrónicos.
Multiobjetivo	Optimización de problemas de programación de la producción, planificación de horarios en universidades, y diseño de redes robustas.

Cuadro 2: Aplicaciones Relevantes de ACO

4. Conclusiones

Escamilla Resendiz Aldo: El algoritmo de optimización por colonia de hormigas es una técnica metaheurística que ha demostrado ser efectiva en la resolución de problemas de optimización combinatoria. A través de la simulación del comportamiento de las hormigas en la naturaleza, el ACO es capaz de encontrar soluciones de alta calidad en un tiempo razonable. Las diferentes versiones de ACO ofrecen una variedad de enfoques para abordar problemas específicos, y su flexibilidad permite adaptarse a una amplia gama de aplicaciones. En el contexto de la configuración de parámetros, ANOVA ayuda a identificar cuáles parámetros tienen un efecto significativo en el rendimiento del sistema o proceso bajo estudio. Esto se realiza dividiendo la variabilidad total de los datos en componentes atribuibles a diferentes fuentes de variación.

Yañez Martinez Marthon Leobardo: El ACO es una técnica de optimización poderosa que ha demostrado ser efectiva en una variedad de aplicaciones. Al simular el comportamiento de las hormigas en la naturaleza, el ACO es capaz de encontrar soluciones de alta calidad para problemas de optimización combinatoria. Las diferentes versiones de ACO ofrecen una variedad de enfoques para abordar problemas específicos, y su flexibilidad permite adaptarse a una amplia gama de aplicaciones.

Castillo Reyes Diego: El algoritmo de optimización por colonia de hormigas es una técnica metaheurística que ha demostrado ser efectiva en la resolución de problemas de optimización combinatoria. A través de la simulación del comportamiento de las hormigas en la naturaleza, el ACO es capaz de encontrar soluciones de alta calidad en un tiempo razonable. Las diferentes versiones de ACO ofrecen una variedad de enfoques para abordar problemas específicos, y su flexibilidad permite adaptarse a una amplia gama de aplicaciones.