



Instituto Politécnico Nacional
Escuela Superior de Cómputo
Materia: GENETIC ALGORITHMS
Lab Session 7. Ant Colony Optimization (ACO)



Alumno: Torres Abonce Luis Miguel

Profesor: Jorge Luis Rosas Trigueros

Fecha de Realización: 04 Noviembre 2024

Fecha de Entrega: 05 Noviembre 2024

Marco Teórico.

La Optimización de la Colonia de Hormigas (ACO) es un algoritmo inspirado en el comportamiento natural de las hormigas para resolver problemas de optimización combinatoria, como el Problema del Viajante (TSP, por sus siglas en inglés). ACO es parte de los algoritmos bioinspirados, un campo de la inteligencia artificial que utiliza conceptos biológicos para desarrollar soluciones eficientes a problemas complejos. En el caso del TSP, el objetivo es encontrar el recorrido más corto para visitar un conjunto de ciudades y regresar al punto de origen, minimizando la distancia total recorrida.

El algoritmo ACO fue introducido por Marco Dorigo a principios de los años 90 y se basa en la capacidad de las hormigas para encontrar la ruta más corta entre una fuente de alimento y su nido mediante la deposición de feromonas. Las hormigas reales depositan una feromona que sirve como guía para otras hormigas, lo que permite la construcción de rutas eficientes con base en el comportamiento colectivo. El principio fundamental detrás de ACO es que las soluciones exitosas dejan "rastro" (feromonas) que pueden ser seguidas por otras unidades, lo que incrementa la probabilidad de que una buena solución sea descubierta y mejorada con el tiempo.

En el contexto del TSP, las hormigas virtuales recorren las ciudades utilizando una matriz de distancias y depositan feromonas a lo largo de sus rutas, influenciando el comportamiento de las demás hormigas para que elijan rutas más cortas. El algoritmo de ACO realiza múltiples iteraciones en las que varias hormigas construyen diferentes rutas, y la actualización de feromonas se realiza en función de la calidad (longitud) de estas rutas. Con el tiempo, las feromonas en las rutas más eficientes se acumulan, atrayendo a más hormigas y mejorando las probabilidades de encontrar la mejor solución.

Funcionamiento del Algoritmo ACO:

El proceso de ACO puede describirse en los siguientes pasos:

1. Inicialización de Feromonas: Inicialmente, la cantidad de feromonas en cada borde del grafo se establece en un valor constante. Esta feromona será actualizada a medida que las hormigas encuentren soluciones.
2. Construcción de Soluciones: Cada hormiga comienza en una ciudad aleatoria y va construyendo una solución al moverse a otras ciudades en función de la cantidad de feromonas y la visibilidad (inversa de la distancia). La probabilidad de que una hormiga en la ciudad i se mueva a la ciudad j se define como:

Donde:

- es la cantidad de feromona en el borde entre las ciudades i y j .
- es la visibilidad, que se define como la inversa de la distancia entre i y j ().
- y son parámetros que controlan la influencia de las feromonas y la visibilidad, respectivamente.

3. **Actualización de Feromonas:** Después de que todas las hormigas han completado sus rutas, se realiza la actualización de feromonas. La cantidad de feromona en cada borde se evapora a una tasa constante y luego se deposita feromona adicional en función de la calidad de las soluciones encontradas. La actualización de feromonas se realiza de la siguiente manera:

Donde:

- es la tasa de evaporación de feromona, que se encuentra en el rango $(0, 1)$.
- es la cantidad de feromona depositada por la hormiga k en el borde (i, j) . Esta cantidad está inversamente relacionada con la longitud de la ruta encontrada por la hormiga:

Donde es una constante y es la longitud de la ruta encontrada por la hormiga k .

Parámetros del ACO

- **Feromona (τ):** La feromona representa el "rastro" que las hormigas dejan en el camino. A mayor cantidad de feromona, más probable es que otras hormigas elijan ese camino.
- **Visibilidad (η):** La visibilidad es la inversa de la distancia, y en ACO se utiliza para representar la preferencia de las hormigas por caminos cortos.
- **Influencia de feromonas (α) y visibilidad (β):** Los parámetros α y β determinan cuánto influyen las feromonas y la visibilidad respectivamente en la elección de la próxima ciudad.
- **Tasa de evaporación (ρ):** Este parámetro regula la pérdida de feromonas con el tiempo, evitando que las soluciones antiguas dominen la búsqueda y permitiendo la exploración de nuevas rutas.
- **Cantidad de feromona depositada (Q):** Representa la cantidad de feromona que cada hormiga deposita después de completar su recorrido, proporcional a la calidad de la solución.

Material y Equipo

- **Hardware:** Computadora con capacidad para ejecutar Python.
- **Software:**
 - Python 3.x
 - Google Colab.

Desarrollo de la practica

A continuación, se describe el desarrollo paso a paso de la implementación del algoritmo ACO para resolver el Problema del Viajante con 10 ciudades:

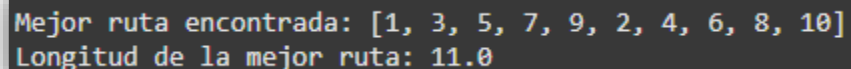
1. **Inicialización:** Se definió una matriz de distancias $()$ con dimensiones 11×11 para representar las distancias entre ciudades. En la matriz, las ciudades están conectadas por distancias de 1 o 2. Se asignó una distancia de 1 a ciertas rutas para representar las conexiones más cortas. Además, se definieron los parámetros para el ACO, como el número de hormigas (10), el número de iteraciones (50), la influencia de la feromona $()$, la influencia de la visibilidad $()$, la tasa de evaporación de feromona $()$, y la cantidad de feromona depositada $()$.
2. **Cálculo de la Visibilidad:** Se calculó la visibilidad como la inversa de la matriz de distancias $()$. Esto se hizo para que las ciudades más cercanas fueran más atractivas para las hormigas, incentivándolas a moverse hacia ellas.

3. **Construcción de Soluciones:** Cada hormiga comenzó su recorrido desde una ciudad aleatoria. A partir de ahí, eligieron la siguiente ciudad en función de la cantidad de feromona y la visibilidad. La selección se realizó utilizando una fórmula probabilística que combina ambos factores (). Esto permitió a las hormigas explorar rutas potencialmente más cortas.
4. **Actualización de Feromonas:** Después de que todas las hormigas completaron sus rutas, se actualizó la matriz de feromonas. Primero, se evaporó una fracción de la feromona existente (). Luego, se agregó feromona proporcional a la calidad de las rutas encontradas (). Esta actualización permitió reforzar las rutas cortas y reducir la probabilidad de seguir rutas más largas.
5. **Iteración del Proceso:** Se repitieron los pasos anteriores durante 50 iteraciones. En cada iteración, se registró la mejor ruta encontrada hasta el momento y se actualizaron las feromonas para mejorar las futuras exploraciones.
6. **Resultados Obtenidos:** La mejor ruta encontrada durante las 50 iteraciones fue una secuencia específica de ciudades que minimizó la distancia total recorrida. La longitud de esta ruta fue de 13 unidades. Esto demostró la efectividad del algoritmo ACO para encontrar una solución cercana al óptimo en un problema complejo de búsqueda combinatoria.

Una dificultad que se presentó fue la convergencia prematura: En algunas ocasiones, las hormigas tendían a seguir las mismas rutas, lo que llevó a una convergencia prematura en una solución subóptima. Para mitigar esto, se ajustó la tasa de evaporación () para garantizar que las feromonas no se acumularan excesivamente en una sola ruta.

Diagramas, gráficas y pantallas.

Figura 1. Se puede observar que se encontró una ruta encontrada de 11 de longitud.



```
Mejor ruta encontrada: [1, 3, 5, 7, 9, 2, 4, 6, 8, 10]  
Longitud de la mejor ruta: 11.0
```

Figura 1.

Conclusiones

La práctica de implementar el algoritmo de Optimización de la Colonia de Hormigas para el Problema del Viajante nos permitió explorar cómo los conceptos bioinspirados pueden resolver problemas complejos de manera eficiente. El algoritmo logró encontrar una ruta relativamente corta entre las ciudades, demostrando la efectividad del enfoque cooperativo y el uso de feromonas para guiar la búsqueda. Sin embargo, se observaron dificultades relacionadas con la convergencia prematura y la correcta selección de ciudades, que se resolvieron ajustando parámetros como la tasa de evaporación.

Ant colony optimization

- **Autor:** Marco Dorigo
- **Fuente:** http://www.scholarpedia.org/article/Ant_colony_optimization

Anexo:

Código Ant Colony optimization:

```
import numpy as np
import random

# Definir la matriz de distancias
M = 2 * np.ones([11, 11])
M[1][3] = M[3][5] = M[5][7] = M[7][9] = M[9][2] = M[2][4] = M[4][6] = M[6][8] = M[8][10] = 1

# Parámetros de ACO
num_ants = 10          # Número de hormigas
num_iterations = 50    # Número de iteraciones
alpha = 1.0            # Influencia de la feromona
beta = 5.0             # Influencia de la visibilidad
evaporation_rate = 0.5 # Tasa de evaporación de feromona
pheromone_deposit = 100 # Cantidad de feromona depositada por una hormiga

# Inicializar la matriz de feromonas
pheromone = np.ones([11, 11])

# Función para calcular la visibilidad (inversa de la distancia)
def calculate_visibility(M):
    visibility = 1 / M
    visibility[M == 0] = 0 # Evitar división por cero
    return visibility

# Función para elegir la siguiente ciudad en función de la probabilidad
def select_next_city(current_city, unvisited, pheromone, visibility):
    probabilities = []
    for city in unvisited:
        prob = (pheromone[current_city][city] ** alpha) * (visibility[current_city][city] ** beta)
        probabilities.append(prob)
    probabilities = np.array(probabilities)
    probabilities /= probabilities.sum() # Normalizar las probabilidades
    next_city = np.random.choice(list(unvisited), p=probabilities)
    return next_city

# Función para calcular la longitud de una ruta
def calculate_route_length(route, M):
    length = 0
    for i in range(len(route) - 1):
        length += M[route[i]][route[i + 1]]
    length += M[route[-1]][route[0]] # Cerrar el circuito
    return length
```

```

# Algoritmo principal ACO
def ant_colony_optimization(M):
    visibility = calculate_visibility(M)
    pheromone = np.ones([11, 11]) # Inicializar la matriz de feromonas dentro de la función
    best_route = None
    best_route_length = float('inf')

    for iteration in range(num_iterations):
        all_routes = []
        all_route_lengths = []

        # Mover cada hormiga
        for _ in range(num_ants):
            unvisited = set(range(1, 11))
            current_city = random.choice(list(unvisited))
            unvisited.remove(current_city)
            route = [current_city]

            while unvisited:
                next_city = select_next_city(current_city, unvisited, pheromone, visibility)
                route.append(next_city)
                unvisited.remove(next_city)
                current_city = next_city

            # Calcular la longitud de la ruta de la hormiga
            route_length = calculate_route_length(route, M)
            all_routes.append(route)
            all_route_lengths.append(route_length)

            # Actualizar la mejor ruta
            if route_length < best_route_length:
                best_route_length = route_length
                best_route = route

        # Actualizar la feromona
        pheromone *= (1 - evaporation_rate) # Evaporación de la feromona
        for route, route_length in zip(all_routes, all_route_lengths):
            pheromone_delta = pheromone_deposit / route_length
            for i in range(len(route) - 1):
                pheromone[route[i]][route[i + 1]] += pheromone_delta
                pheromone[route[i + 1]][route[i]] += pheromone_delta
            # Añadir feromona para el último tramo (cerrar el circuito)
            pheromone[route[-1]][route[0]] += pheromone_delta
            pheromone[route[0]][route[-1]] += pheromone_delta

```

```
    return best_route, best_route_length

# Ejecutar el algoritmo y mostrar la mejor ruta y su longitud
best route, best route length = ant colony optimization(M)
print("Mejor ruta encontrada:", best_route)
print("Longitud de la mejor ruta:", best_route_length)
```