Facultad de Ingenierías
Departamento de Ingeniería en Sistemas

Estrategia 3 -Proyecto

Análisis y Diseño de Algoritmos

Alejandro González Flórez

Luis Felipe Giraldo

Jose David González Villa

Manizales, Caldas

2024

Estrategia 3 - Optimización mediante Ant Colony Optimization (ACO)

1. Introducción

En problemas de optimización combinatoria, como la **división de sistemas probabilísticos en particiones óptimas**, los métodos tradicionales como la fuerza bruta presentan serias limitaciones debido al crecimiento exponencial del espacio de búsqueda. En este contexto, el algoritmo **Ant Colony Optimization (ACO)** surge como una solución bioinspirada eficiente para explorar grandes espacios de soluciones de manera inteligente y equilibrada.

El objetivo de esta estrategia es encontrar **la partición que minimice la métrica Earth Mover's Distance (EMD)** entre el sistema original y el sistema particionado. La métrica EMD, basada en la teoría del transporte óptimo, permite evaluar la **disimilitud** entre dos distribuciones de probabilidad, siendo especialmente útil en este tipo de problemas.

---

2. Problema a Resolver

Dado un sistema definido por probabilidades de transición entre un conjunto de estados actuales (cs) y futuros (ns), se busca dividir el sistema en **particiones disjuntas** que preserven la distribución original de manera óptima.

La **función objetivo** es minimizar la siguiente distancia:

$$EMD = wasserstein\ distance(P_{original}, P_{particionado})$$

Donde:

- $P_{Original}$: Distribución de probabilidad del sistema original.
- $P_{Particional}$: Distribución generada por las particiones.

---

3. Descripción del Algoritmo Ant Colony Optimization (ACO)

3.1 Principios del ACO

El **ACO** está inspirado en el comportamiento colectivo de las hormigas en la naturaleza. Estas hormigas depositan **feromonas** en los caminos recorridos, lo que ayuda a guiar a otras hormigas hacia rutas más prometedoras. Las decisiones de las hormigas están influenciadas por dos factores clave:

1. **Feromonas (τ)**: Representan la memoria colectiva de las soluciones encontradas.
2. **Heurística (η)**: Representa la calidad de una solución basada en un criterio local (en nuestro caso, la métrica **EMD**).

$$P_i = \frac{\tau_i^{\alpha} . \eta_i^{\beta}}{\Sigma_i \tau_j^{\alpha} . \eta_j^{\beta}}$$

Donde:

- α: Controla la influencia de las feromonas.
- β: Controla la influencia de la heurística (calidad de la solución).

3.2 Características Principales del ACO

1. **Exploración vs Explotación**:
   El ACO equilibra la búsqueda de nuevas soluciones (**exploración**) y la mejora de las soluciones actuales (**explotación**) mediante los parámetros α y β.

2. **Actualización de Feromonas**:
   Se realiza en dos etapas:

   - **Evaporación global**: Reduce las feromonas en todas las soluciones:

   $$\tau_i = (1 - \rho) . \tau_i$$

   - **Deposición local**: Aumenta las feromonas proporcionalmente a la calidad de la solución:

   $$\tau_i = \tau_i + \frac{1}{1+EMD}$$

4. Implementación del Código

A continuación, se describe detalladamente el código implementado.

4.1 Inicialización

Cálculo de la tabla de probabilidades original: Se utiliza la función `obtener_tabla_probabilidades` para calcular la distribución original del sistema, a partir de las transiciones entre los estados actuales (cs) y futuros (ns).

```
original_table = obtener_tabla_probabilidades(
    repr_current_to_array(self.cs, self.cs_value),
    repr_next_to_array(self.ns),
    self.probabilities,
    self.states
)
self.original_system = np.array(original_table).flatten()
```

1. **Explicación**:
   La tabla de probabilidades es aplanada en un vector para facilitar su comparación mediante la métrica **EMD**.

4.2 Generación de Particiones Válidas

La función `generate_valid_partitions` genera **todas las combinaciones válidas** de particiones disjuntas a partir de los estados (cs y ns):

```
left_ns = list(self.ns[:i])
right_ns = list(ns_set - set(left_ns))

left_cs = list(self.cs[:i])
right_cs = list(cs_set - set(left_cs))
```

Ejemplo Teórico:
Supongamos $ns = [N1, N2, N3]$ y $cs = [C1, C2, C3]$. Una partición válida sería:

$$Left = \{N1, C1\}, Right = \{N2, N3, C2, C3\}$$

4.3 Ejecución del ACO

La función `run_aco` ejecuta el algoritmo en varias iteraciones con **un número definido de hormigas** (num_ants):

Selección de soluciones:
Cada hormiga selecciona particiones basándose en la fórmula de probabilidad:

```
probabilities = [pheromones[str(p)] ** alpha for p in valid_partitions]
probabilities /= np.sum(probabilities)
selected_partition = random.choices(
    list(valid_partitions), weights=probabilities, k=1
)[0]
```

Evaluación mediante EMD:
Se calcula la métrica **EMD** para evaluar la calidad de cada partición:

$$EMD = wassertein\_distance(original, partitioned)$$

Actualización de Feromonas:
Evaporación global:

$$pheromones[partition\_str] *= (1 - evaporation)$$

Incremento local basado en la calidad:

$$pheromones[str(partition)] += 1.0/(1 + emd)$$

4.4 Selección de la Mejor Solución

Al finalizar todas las iteraciones, se selecciona la **partición con menor EMD** como la solución óptima.

5. Resultados y Análisis

1. **Mejor partición encontrada:**
   Se presenta en formato matemático LaTeX:

$$\left(\frac{NS_{left}}{CS_{Right}}\right) \times \left(\frac{Cs_{left}}{NS_{Right}}\right)$$

2. **Valor mínimo de EMD:**

$$EMD\ Mínimo\ =\ <valor>$$

3. **Tiempo de ejecución:**
   $Tiempo\ Total:\ <valor\ en\ segundos>$

6. Consideraciones de Eficiencia

- **Complejidad temporal:**
  La complejidad del algoritmo ACO depende del número de hormigas (num_ants), iteraciones (iterations) y del tamaño del espacio de soluciones.

$$O(num\_ants \cdot iterations \cdot |particiones|)$$

- **Optimización de datos:**
  Se implementó un almacenamiento en memoria para evitar reprocesar soluciones ya evaluadas.
- **Reducción del espacio de búsqueda:**
  La función heurística basada en **EMD** permitió priorizar soluciones de alta calidad, acelerando la convergencia.

7. Justificación del Uso de ACO

- **Eficiencia en espacios grandes:** El ACO evita el crecimiento exponencial de combinaciones al explorar las soluciones de manera inteligente.
- **Capacidad adaptativa:** El algoritmo balancea la exploración de nuevas soluciones y la explotación de las mejores soluciones encontradas.
- **Evaluación robusta:** La métrica **EMD** proporciona una medida precisa y fundamentada de la calidad de las soluciones.