Planificador de Hospedaje en Hoteles

Sistema de Optimización de Itinerarios

16 de junio de 2025

1. Introducción

Este sistema utiliza metaheurísticas para optimizar itinerarios de hoteles considerando:

- Presupuesto total
- Calidad (estrellas) de hoteles
- Preferencia por minimizar cambios de hotel
- Restricciones de destino y disponibilidad

2. Función Objetivo (Fitness)

Definida en `fitness.py`, evalúa soluciones mediante:

$$\text{fitness} = \alpha \times \text{stars\_norm} + \beta \times (1 - \text{cost\_norm}) + \gamma \times (1 - \text{changes\_norm}) \tag{1}$$

Donde:

$$\begin{aligned} \text{stars\_norm} &= \frac{\sum \text{estrellas}}{\text{noches} \times \text{max\_stars}} \\ \text{cost\_norm} &= \text{mín}\left(\frac{\text{costo\_total}}{\text{presupuesto}}, 1\right) \\ \text{changes\_norm} &= \frac{\text{cambios\_hotel}}{\text{noches} - 1} \end{aligned}$$

3. Metaheurísticas Implementadas

3.1. Búsqueda en Profundidad (DFS)

Implementada en `graph_explorer.py` y `graph_node.py`.

- **Estrategia**: Búsqueda exhaustiva con backtracking
- **Representación**: Árbol donde cada nodo contiene:

```
class GraphNode:
    night: int
    hotel: Hotel
    budget_left: float
    stars_accum: int
    path: List[Tuple[int, Hotel]]
```

- **Flujo**:
  1. Inicializa con todos los hoteles válidos para la primera noche
  2. Expande nodos generando hijos válidos (presupuesto)
  3. Ordena opciones por: estrellas ↓ y relación estrellas/precio ↓
  4. Mantiene la mejor solución completa
- **Ventaja**: Óptimo garantizado
- **Limitación**: Complejidad exponencial $O(h^n)$

3.2. Optimización por Colonia de Hormigas (ACO)

Implementada en `aco_planner.py`.

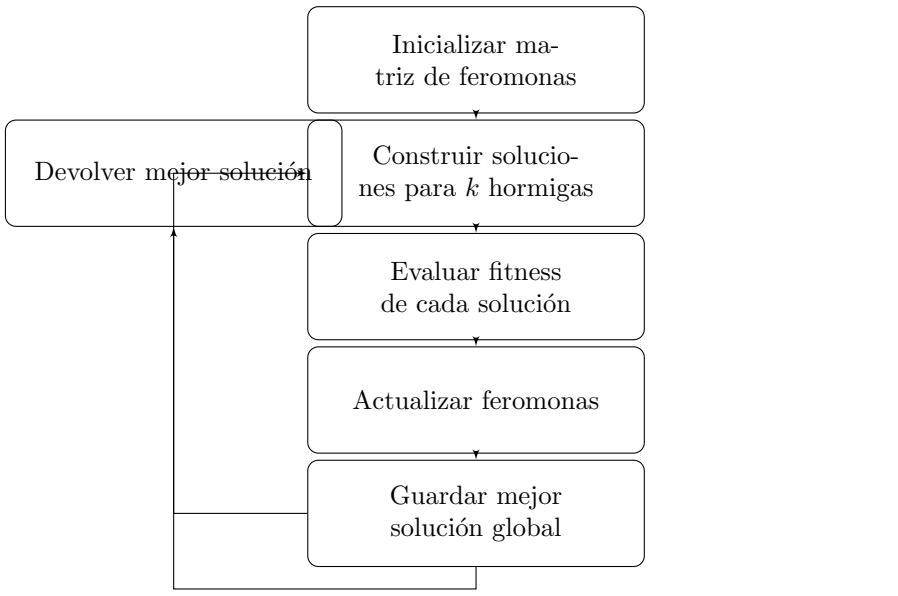

Figura 1: Diagrama de flujo del algoritmo ACO

Ecuación de probabilidad:

$$P(i) = \frac{\tau_i \times \eta_i}{\sum_j \tau_j \times \eta_j} \tag{2}$$

Donde:

- τ_i: Feromona en hotel i
- η_i: Heurística $\frac{\text{estrellas}_i}{\text{precio}_i}$
- Penalización por cambio: $\eta_i \times 0{,}8$ si $i \neq$ hotel anterior

Actualización de feromonas:

$$\tau_i^{(t+1)} = (1 - \rho) \times \tau_i^{(t)} + \Delta\tau_i$$
$$\Delta\tau_i = \sum \text{fitness(soluciones que usan } i)$$

3.3. Optimización por Enjambre de Partículas (PSO)

Implementada en `pso_planner.py`.

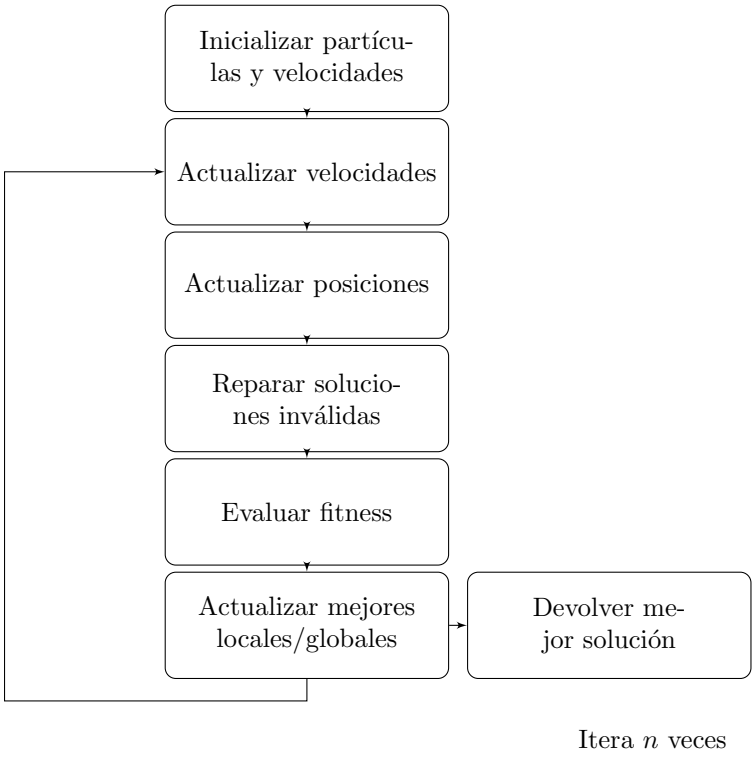

Figura 2: Diagrama de flujo del algoritmo PSO

Ecuaciones de actualización:

$$v_{id}^{(t+1)} = w \cdot v_{id}^{(t)} + c_1 r_1 (p_{id} - x_{id}^{(t)}) + c_2 r_2 (g_d - x_{id}^{(t)})$$

$$P(\text{cambio}) = \frac{1}{1 + e^{-v_{id}^{(t+1)}}}$$

Mecanismos clave:

- Representación: Vector de índices de hoteles
- Reparación: Reemplazar hoteles que exceden presupuesto
- Sigmoide: Transforma velocidad a probabilidad de cambio

4. Comparativo de Algoritmos

Característica	DFS	ACO	PSO
Optimalidad	Garantizada	Heurística	Heurística
Complejidad temporal	$O(h^n)$	$O(n \cdot k \cdot h)$	$O(n \cdot p \cdot d)$
Espacio de búsqueda	Pequeños	Grandes	Grandes
Paralelización	Difícil	Alta	Muy Alta
Noches recomendadas	< 7	< 15	> 10
Sensibilidad a parámetros	Baja	Media	Alta

Cuadro 1: Comparativo de metaheurísticas

5. Conclusiones

- **DFS**: Ideal para problemas pequeños con garantía de optimalidad
- **ACO**: Balance óptimo entre exploración y explotación
- **PSO**: Alto rendimiento en problemas grandes con paralelización
- **Función fitness**: Combina calidad, costo y estabilidad en métrica unificada