HOJA DE RESUMEN: TSP - Algoritmos Heurísticos

CONCEPTOS CLAVE

1. Definición Matemática del TSP

Dado grafo completo ponderado $G = (V, E, w)$, encontrar ciclo hamiltoniano H de costo mínimo:

$$\text{mín} \sum_{(i,j)\in H} w(i,j)$$

¡IMPORTANTE! TSP es **NP-completo**: $(n-1)!/2$ rutas. Para $n = 20$: 6×10^{16} combinaciones. Fuerza bruta es $O(n!)$ - ¡intratable!

2. Estructuras de Datos - Comparación

Matriz de Adyacencia (usar para TSP):

- Acceso a peso: $O(1)$
- Espacio: $O(n^2)$
- Ideal para grafos densos/completos

Lista de Adyacencia:

- Espacio: $O(|V| + |E|)$
- Mejor para grafos dispersos
- Verificar arista: $O(\text{grado})$

3. Algoritmo Voraz: Vecino Más Cercano

Estrategia: Desde ciudad actual, ir a la no visitada más cercana.
Complejidad: $O(n^2)$ — **Gap:** ∼25 % del óptimo

```
def vecino_mas_cercano(dist_matrix, inicio=0):
    n = len(dist_matrix)
    visitado = [False] * n
    tour = [inicio]
    visitado[inicio] = True
    ciudad_actual = inicio
    costo_total = 0

    for _ in range(n - 1):
        min_dist = float('inf')
        ciudad_mas_cercana = -1

        # Buscar ciudad no visitada mas cercana
        for j in range(n):
            if not visitado[j] and dist_matrix[ciudad_actual][j] < min_dist:
                min_dist = dist_matrix[ciudad_actual][j]
                ciudad_mas_cercana = j

        tour.append(ciudad_mas_cercana)
        visitado[ciudad_mas_cercana] = True
        costo_total += min_dist
        ciudad_actual = ciudad_mas_cercana

    # Cerrar ciclo
    costo_total += dist_matrix[ciudad_actual][inicio]
    tour.append(inicio)

    return tour, costo_total
```

Ventajas: Ultra-rápido, fácil implementar
Desventajas: Solución subóptima (20-30 % peor que óptimo)

4. Algoritmo Genético (GA)

Conceptos: Población (soluciones) — Cromosoma (tour) — Fitness (1/distancia) — Selección (torneo) — Cruce (OX) — Mutación (2-opt)

Complejidad: $O(g \cdot p \cdot n^2)$ donde g=generaciones, p=población

Parámetros típicos:

- Población: 100-200
- Generaciones: 1000-5000
- Prob. mutación: 0.01-0.05
- Torneo: tamaño 3-7

Order Crossover (OX) - Clave para examen:

```
def order_crossover(padre1, padre2):
    n = len(padre1)
    i = random.randint(0, n-2)
    j = random.randint(i+1, n)

    # Copiar segmento del padre1
    hijo = [-1] * n
    hijo[i:j] = padre1[i:j]

    # Completar con elementos del padre2 en orden
    pos = j
    for ciudad in padre2[j:] + padre2[:j]:
        if ciudad not in hijo:
            hijo[pos % n] = ciudad
            pos += 1

    return hijo
```

Mutación 2-opt:

```
def mutacion_2opt(tour, prob_mutacion=0.01):
    if random.random() < prob_mutacion:
        i = random.randint(1, len(tour)-3)
        j = random.randint(i+1, len(tour)-1)
        tour[i:j] = tour[i:j][::-1]  # Invertir segmento
    return tour
```

5. Recocido Simulado (SA)

Analogía: Enfriamiento de metal. Acepta soluciones peores para escapar óptimos locales.
Complejidad: $O(k \cdot n)$ donde k=iteraciones

Fórmula de Aceptación - ¡MEMORIZAR!

$$P(\Delta E, T) = \begin{cases} 1 & \text{si } \Delta E < 0 \text{ (mejor)} \\ e^{-\Delta E/T} & \text{si } \Delta E \geq 0 \text{ (peor)} \end{cases}$$

Enfriamiento Geométrico: $T_{k+1} = \alpha \cdot T_k$ donde $\alpha \in [0{,}9, 0{,}999]$

```
def recocido_simulado(dist_matrix, temp_inicial=1000, alpha=0.995):
    n = len(dist_matrix)
    tour_actual = list(range(n))
    random.shuffle(tour_actual)
    costo_actual = calcular_costo(tour_actual)

    mejor_tour = tour_actual[:]
    mejor_costo = costo_actual
    temperatura = temp_inicial

    while temperatura > 0.1:
        # Generar vecino (2-opt)
        i, j = sorted(random.sample(range(n), 2))
        tour_vecino = tour_actual[:]
        tour_vecino[i:j+1] = tour_vecino[i:j+1][::-1]
        costo_vecino = calcular_costo(tour_vecino)

        # Decidir aceptacion
        delta = costo_vecino - costo_actual
        if delta < 0 or random.random() < math.exp(-delta/temperatura):
            tour_actual = tour_vecino
            costo_actual = costo_vecino

            if costo_actual < mejor_costo:
                mejor_tour = tour_actual[:]
                mejor_costo = costo_actual

        # Enfriar
        temperatura *= alpha

    return mejor_tour, mejor_costo
```

6. Comparación de Algoritmos

Algoritmo	Complejidad	Gap	Características
Vecino Cercano	$O(n^2)$	∼25 %	Muy rápido, baja calidad
Genético	$O(g \cdot p \cdot n^2)$	5-15 %	Paralelizable, muchos parámetros
Recocido	$O(k \cdot n)$	3-13 %	Mejor balance calidad/-tiempo

7. Cuándo Usar Cada Algoritmo

ESCENARIOS PARA EXAMEN:

- **Tiempo real (¡1s):** Vecino Más Cercano
- **Alta calidad necesaria:** Recocido Simulado o GA

- **n ¡20 ciudades:** Branch & Bound (exacto)
- **n ¿500 ciudades:** SA paralelo
- **Necesita paralelización:** Algoritmo Genético
- **Balance calidad/tiempo:** Recocido Simulado

8. Optimizaciones Avanzadas

Mejora 2-opt: Búsqueda local. Invertir segmentos del tour hasta que no haya mejora. Complejidad: $O(n^3)$

Algoritmo Híbrido: GA (exploración global) → SA (refinamiento local) → 2-opt (búsqueda local)

9. Aplicación Real - Caso Costa Rica

Valle Central (8 ciudades):

- Reducción: 25.5 % (72.7 km ahorrados)
- Algoritmo: Recocido Simulado

Logística San José (50 puntos):

- Reducción distancia: 22 %
- Ahorro: $3,500/mes
- Tiempo planificación: 2h → 5min
- ROI: 2 meses

10. Fórmulas y Conceptos Fundamentales

REFERENCIA RÁPIDA:

Número de rutas posibles: $(n-1)!/2$

Fitness en GA: fitness $= \frac{1}{\text{distancia\_total}}$

Probabilidad SA: $P = e^{-\Delta/T}$ cuando $\Delta \geq 0$

Enfriamiento: $T_{k+1} = \alpha \cdot T_k$

Matriz distancias: $M[i][j] = w(i,j)$ si existe arista, ∞ si no

Complejidades:

- Fuerza Bruta: $O(n!)$ - intratable
- Vecino Cercano: $O(n^2)$ - rápido
- GA: $O(g \cdot p \cdot n^2)$ - medio
- SA: $O(k \cdot n)$ - medio
- 2-opt: $O(n^3)$ - búsqueda local

PUNTOS IMPORTANTES:

- TSP en grafo **completo**
- Vecino Cercano es **greedy**
- GA necesita población **diversa**
- SA acepta peores **probabilísticamente**
- OX preserva **orden** de padre2
- 2-opt **invierte** segmento