Documento de la práctica. Sistema de Distribución de Gasolina

Resumen ejecutivo:

El objetivo era diseñar y evaluar estrategias para asignar peticiones de repostaje (gasolineras) a camiones desde centros de distribución, buscando maximizar beneficio (ingresos por peticiones atendidas) y minimizar costes (kilómetros recorridos / tiempo). El código principal está en IA.Gasolina, con la clase de tablero GasolinaBoard que implementa el estado y los operadores, y Main que permite ejecutar búsquedas (Hill-Climbing, SA desde AIMA).

1. Implementación del estado

Clases relevantes:

- `IA.Gasolina.Estado` – Representa la agregación de camiones.
  Contiene una lista `List<Camion> camiones` con todos los camiones (cada camión con sus Viajes), con sus funciones get/set.
- `IA.Gasolina.Camion` – Modela un camión con coordenadas (coordX, coordY), depósito, lista de Viajes, distancia/horas acumuladas.
  El camión almacena una lista de viajes `List<Viajes> viajesCamion,` y contiene las funciones `getViajes, setViaje y addViaje` para hacer las modificaciones necesarias.
  Tambien mantiene métricas acumuladas: `DistanciaRecorrida, horasTrabajadas` y restricciones (`maxViajes, DistanciaMaxima, HorasJornada`).
- `IA.Gasolina.Viaje` – Tramo individual entre puntos (inicio/fin), con `diasPendientes, distanciaTotal, tiempoTotal, flag provisionalReturn`.
  Modela una petición concreta (o tramos entre gasolineras) y almacena `diasPendientes` para calcular beneficio.
- `IA.Gasolina.Viajes` – Grupo de tramos que forman un viaje completo (centro -> gasolineras -> centro).

Justificación:

- Cada camión guarda su ruta (lista de viajes). Esto facilita:
    - Aplicar operadores locales que modifican viajes o reasignan peticiones entre camiones.
    - Calcular costes y beneficios de forma incremental (suma de distancias/tiempos por viaje).
    - Validar restricciones por camión (número de viajes, distancia máxima, horas).
- Guardar diasPendientes en Viaje permite valorar la urgencia de cada petición y modular el beneficio.

2. Operadores implementados

Funciones principales:

Se encuentran en `IA.Gasolina.GasolinaBoard`:

- `reasignarViajes(idCamionOrigen, idCamionDestino, idViaje)`
    - Movimiento: mueve la primera petición (no retorno provisional) de un viaje en el camión origen al camión destino.
    - Efecto: puede reducir solapamientos y equilibrar cargas.
    - Justificación: operador pequeño (bajo factor de ramificación si se selecciona con criterio), útil para escapar de soluciones con camiones saturados.

- `intercambiaViajes(idCamionA, idCamionB, idViajeA, idViajeB)`
    - Movimiento: intercambia peticiones entre dos camiones (reconstrucción completa de viajes).
    - Justificación: swap entre camiones es una manera clásica de explorar redistribuciones sin romper demasiadas restricciones.

- `invertirOrdenViaje(idCamion, idViaje)`
    - Movimiento: invierte el orden de visita de dos gasolineras dentro de un viaje (cuando hay exactamente dos).
    - Justificación: operador local que arregla pequeñas ineficiencias de orden dentro de un mismo viaje (análogo a 2-opt simplificado).

- `swapPeticionesMismoViaje(idCamion, idViaje, idxA, idxB)`
    - Movimiento: cambia el orden de dos paradas (`idxA y idxB`) dentro de un mismo viaje.
    - Justificación: como anterior, reduce coste dentro del mismo camión sin afectar a otros camiones.

- `dividirViajeEnDos(idCamion, idViaje)`
    - Movimiento: descompone un viaje con dos gasolineras en dos viajes separados (si las restricciones lo permiten).

    - Justificación: permite reducir penalizaciones por tiempo/distancia si dividir en dos viajes es más eficiente (p. ej. por proximidad al centro), y ayuda a explorar cambios estructurales.

- `moverPeticionEntreViajes(idCamion, idViajeOrigen, idxPeticion, idViajeDestino)`
    - Movimiento: mover una petición entre viajes del mismo camión.
    - Justificación: operador fino para reorganizar rutas del mismo camión.

- `fusionarViajes(idCamion, idViajeA, idViajeB)`
    - Movimiento: une dos viajes de un mismo camión si no excede capacidad.
    - Justificación: busca compactar viajes y reducir distancias de retorno múltiples.
- `swapPeticionesNoAsiganada(idCamion, idViaje,idTramo, idPeticionNoAsignada)`
    - Movimiento:cambia una petición asignada por una que no lo esté.
    - Justificación: Pueden haberse descartado viajes por tema de distancia recorrida o tiempo que hagan combinaciones mejores
- `addPeticionNoAsiganada(idCamion, idPeticionNoAsignada)`
    - Movimiento: asigna una que no lo está al camión
    - Justificación: Pueden haberse descartado viajes por tema de distancia recorrida o tiempo que hagan combinaciones mejores y que con los cambios hagan que ahora si que quepan.

Funciones auxiliares:

`reconstruirCamionConPeticiones, reconstruirCamionSinPeticion, extraerPeticiones`

- Uso: reconstruyen las rutas del camión a partir de lista de peticiones (útil después de operaciones de intercambio).
- Justificación: simplifica consistencia post-operador y recalcula métricas del camión.

Notas de implementación:

- Se combinan operadores de distinto alcance: cambios locales (swap dentro mismo viaje), cambios entre viajes del mismo camión, cambios entre camiones y cambios estructurales (dividir/fusionar).
- Esta mezcla permite tanto búsqueda de mejora local como saltos estructurales necesarios para escapar de óptimos locales.
- Muchos operadores reconstruyen viajes “desde cero” tras intercambios. Esto simplifica la lógica y mantiene consistencia
- Es importante que cada operador revise las restricciones del camión tras el cambio (distancia máxima, horas), lo cual ya se tiene en comprobaciones en `Camion.addPeticion` y `Camion.puedeAñadirViaje`.

3. Estrategias para hallar la solución inicial

Funciones:

- `crearEstadoInicial1()` – Asignación por camión más cercano.
    - Por cada petición (por cada gasolinera y sus peticiones) se asigna al camión cuya coordenada actual minimiza la distancia euclídea a la gasolinera.
    - Justificación: heurística simple y razonable, minimiza distancia inicial y carga local, produce rutas compactas como punto de partida.
    - Ventaja: produce solucionehes con baja distancia inicial, facilita que operadores locales consigan mejoras pequeñas.
    - Desventaja: puede sobrecargar camiones cercanos y dejar otros infrautilizados (no balancea cargas).
- `crearEstadoInicial2()` – Round-robin.
    - Asigna peticiones de forma circular entre camiones (primera petición al camión 0, segunda al 1, ...).
    - Justificación: produce distribución balanceada de solicitudes entre camiones sin considerar distancia.
    - Ventaja: cobertura y balance inicial; útil cuando la limitación es capacidad de camiones o número de viajes.
    - Desventaja: rutas iniciales pueden ser de mayor distancia, así que la heurística inicial puede ser peor.

Justificación general:

Probar dos estrategias contrarias (localidad vs. balance) para medir sensibilidad de la búsqueda a la solución inicial. Habitualmente:

- nearest-truck -> mejor punto de partida para coste; HC puede explotarlo.
- round-robin -> mejor punto de partida para evitar sobrecarga; SA puede mejorar distancias.

4. Funciones heurísticas

Heurística implementada en `GasolinaBoard.heuristic()`

Variables que influyen:

- beneficioTotal = suma de `getBeneficio()` por camión
- distanciaTotal = suma de `getDistanciaRecorrida()` por camión.
- perdidaTotal = suma de `getPerdida()` por camión

Heurística = `perdidaTotal - (distanciaTotal*2) - beneficioTotal`

Justificación:

Queremos maximizar el beneficio neto (ganancias por atender peticiones menos costes operativos por desplazamiento) y minimizar la pérdida por dejar peticiones pendientes

Rol de cada término:

- beneficioTotal: recompensa directa por hacer rutas eficientes que atienden peticiones valiosas.
- distanciaTotal*2: coste operativo por mover camiones, penaliza rutas largas.
- perdidaTotal: penaliza no atender peticiones (especialmente las que pierden mucho valor por retraso), fuerza al algoritmo a preferir atender peticiones con mayor pérdida inmediata.

5. Experimentos

Experimentos 1 y 2: Teniendo un contexto de problema en el cual empezamos con 10 centros de distribución, con un camión en cada uno,
y 100 gasolineras y usando el algoritmo de hill climbing hemos hecho un estudio de resultados en función de la heurística para 10 semillas
diferentes (de la 1 a la 10, ambas incluidas) y estos han sido los resultados para las dos estrategias de creación de estados iniciales:

estrategia 1, Greedy Best First:

Semilla	Inicial	Final
1	76529	76831
2	79828	80722
3	76924	77591
4	77504	77736
5	78136	78600
6	75310	76030
7	79766	80503
8	70935	71769
9	66418	66837
10	79445	79597
Total	760795	766216

estrategia 2, Round-Robin:

Semilla	Inicial	Final
1	69364	72454
2	77074	80037
3	75553	78678
4	74769	77319
5	67726	70937
6	72705	75306
7	75674	77825
8	78897	81852
9	64641	68058
10	62429	65829
Total	718832	748295

El estudio no es demasiado grande al ser solamente de 10 semillas diferentes pero basándonos en lo que vemos en él hemos decidido usar la primera
estrategia, en la segunda hay una diferencia más notable entre la solución inicial y final pero incluso después de usar el algoritmo la misma

estrategia inicial de la primera saca mejores resultados de media, y por lo general aunque haya menos diferencia se encuentran mejores resultados
usando la primera estrategia (Greedy Best First), así que nos quedaremos con ella.

En cuanto a los operadores, se usan básicamente todos ellos de manera bastante homogénea, no hay ninguno que a primera vista destaque mucho más
en respecto a los otros, según sea la disposición de las gasolineras y centros se usarán más algunos que otros, pero no nos parece que sobre ninguno.

Experimento 3:

Probando diferentes combinaciones de parámetros hemos decidido esta combinación: número de iteraciones: 1000, número de actualizaciones de temperatura: 100, parámetro k: 20, parámetro lambda: 0.001. Estos són los resultados con las semillas 1 - 10:

Semilla	Inicial	Final
1	76529	76573
2	79828	80157
3	76924	77015
4	77504	77596
5	78136	78346
6	75310	75519
7	79766	80250
8	70935	71248
9	66418	66661
10	79445	79557
Total	760795	762922

Experimento 4:

Estos son los resultados usando Hill Climbing para la semilla 1:

Semilla	Inicial	Final	Tiempo(ms)
10	76529	76831	144
20	105803	106799	407
30	120002	121135	614
40	121215	122419	822
50	125338	126599	1269
60	125724	127143	1571
70	125946	127338	1570
80	126193	127602	1988
90	126240	128067	2716

100	126278	128090	3121

Y estos para Simulated Annealing para la semilla 1 también:

numCentros	Inicial	Final	Tiempo(ms)
10	76529	76689	3352
20	105803	106015	7632
30	120002	120431	12076
40	121215	121445	15856
50	125338	125523	20749
60	125724	126118	23069
70	125946	126054	26886
80	126193	126303	31802
90	126240	126252	35562
100	126278	126659	43677

Como se ve claramente en los resultados incluso aumentando el tamaño del problema ambos algoritmos siguen dando buenos resultados.

Experimento 5:

Estos son los resultados de correr el algoritmo de Hill Climbing teniendo sólo 5 centros con 2 camiones cada uno y 100 gasolineras, se ve claramente que el beneficio es mucho menor ya que ahora los camiones tienen que recorrer muchos más kilómetros para llegar a otras gasolineras y para repostar.

Semilla	Inicial	Final
1	26955	27463
2	27478	27792
3	27321	27617
4	27463	27613
5	27800	28388
6	27936	27983
7	27514	28019
8	28355	28559
9	27827	28164
10	27089	27208
Total	275738	278806

Experimento 6:

Estos son los resultados de usar Hill Climbing con 10 centros de distribución y 100 gasolineras mientras vamos doblando el coste por kilómetro recorrido:

coste por km	Inicial	Final
2	76529	76831
4	72999	73602
8	65939	67145
16	51818	54230
32	23577	28400
64	-32904	-23258

Cómo se puede apreciar en los resultados cuanto más aumenta el coste menos es el beneficio final que se puede obtener llegando incluso a resultados negativos.

Experimento 7:

Estos son los resultados de usar Hill Climbing con 10 centros de distribución y 100 gasolineras con 7 horas de jornada máxima usando semillas del 1 - 10:

Semilla	Incial	Final
1	76745	77045
2	77950	78865
3	75206	75717
4	77504	77736
5	78136	78600
6	74445	75137
7	76135	77066
8	70120	70721
9	65612	65994
10	79445	79597
Total:	751298	756478

Y estos con 9 horas de jornada máxima:

Semilla	Incial	Final
1	76529	76831
2	79828	80722
3	76924	77591
4	77504	77736
5	78136	78600
6	75310	76030
7	79766	80503
8	70935	71769
9	66418	66837
10	79445	79597

Total:	760795	766216

Aunque hay algún caso aislado que hace que con un límite menor de horas se consiga más beneficio la media nos indica que por lo general si aumentamos el límite de horas a trabajar a 9 conseguiremos un mejor beneficio total.