Búsqueda local

Arribas Pardo, Fèlix
felix.arribas@est.fib.upc.edu

Williams Corral, David
david.george.williams@est.fib.upc.edu

García Fuentes, Raul
raul.garcia.fuentes@est.fib.upc.edu

Índice

1. Descripción del problema

1.1. Problema

En éste problema queremos crear una lista de peticiones diaria para una empresa de abastecimiento de gasolineras, así éstas pueden seguir funcionando correctamente. Las peticiones son enviadas por las gasolineras.

En una jornada de ocho horas la empresa tiene que cumplir el máximo de peticiones generando el **beneficio máximo**, **pérdidas mínimas** y el **mínimo de kilómetros recorridos**, ya que esto supone una perdida.

Tenemos el **número de centros** n de distribución de combustible y su posición, y el conjunto de m **gasolineras** que han hecho una petición. Es decir, podemos encontrar la distancia que hay entre las gasolineras que han enviado solicitud y el centro de abastecimiento que las puede atender. Los n centros se tienen que conectar con las m gasolineras necesitadas.

Teniendo en cuenta que n y m son valores muy grandes y que $n < m$, vemos que hay infinitas combinaciones diferentes y es logísticamente imposible cumplir con todas las peticiones que se mandan. Hay que encontrar la **mejor solución**, la que ofrezca más beneficio a la empresa.

Una vez identificado el problema, para saber encontrar la solución tenemos que analizar las características a fondo.

1.1.1. Características

Todo el problema se desarrolla en un área cuadrada de 100×100 kilómetros. Tanto los centros de abastecimiento como las gasolineras ofrecen unas coordenadas para situarlas en este mapa.

La empresa dispone de varios centros de abastecimiento cada uno de éstos dispone de **un camión**. Los camiones se desplazan a 80km/h y las paradas para abastecer gasolineras o para cargar cisternas en el centro supondremos que duran 0h. Por lo tanto, sabiendo que las jornadas son de ocho horas y que el camión circula a 80km/h, en un día éste puede recorrer $80km/h \times 8h = 640km$.

Los camiones no solo están restringidos por los kilómetros que pueden recorrer si no también por el número de viajes: **5 viajes** como máximo. Y, obviamente, tiene un limite de cisternas que pueden transportar, que es 2.

Una cisterna abastece una sola petición, así que el máximo de peticiones que puede atender un camión en un día son $5\ viajes \times 2\ cisternas = 10\ peticiones$. Siempre y cuando no superen los 640 kilómetros recorridos.

A todo esto hay que añadir un factor más: Los días de retraso de cada petición. Como hemos explicado al principio el problema sucede en una día. Las peticiones que recibe la empresa pueden ser del mismo día o de días anteriores. Se empieza con una lista de peticiones, algunas de días anteriores y otras del mismo día. Por cada día que pasa sin abastecer una gasolinera se pierde dinero o se gana menos de lo previsto, siguiendo esta función matemática:

$$\%precio = (100 - 2^{dias})\,\%$$

$$Para\ dias = 0\ partimos\ del\ 102\,\%$$

En el caso en que la solicitud se haya hecho el mismo día el precio será el 102 %. El precio de una cisterna es de 1000 euros. En un principio será mejor abastecer primero las gasolineras que hicieron la petición hace días que no las que acaban de hacerla. Es importante tener en cuenta que se supone que las peticiones que no se cumplen durante la jornada que representa el problema se cumplirán el día siguiente.

Por último, la distancia entre centro y gasolinera también importa. No solo porque se pueda superar el límite de kilómetros recorridos por un camión en un día (640km) si no porque asumiremos un coste por kilómetro recorrido: 2 euros por kilómetro recorrido. Para poder saber la distancia que se recorre se usará la siguiente fórmula para encontrar los kilómetros entre el punto A i B:

$$d(A, B) =\mid A_x - B_x \mid + \mid A_y - B_y \mid$$

1.1.2. Tipo de problema

Podemos definir este problema como un problema de **búsqueda local**. Cada combinación diferente de peticiones repartidas entre los diferentes centros de abastecimiento tiene sus propias características y esta relacionado con las demás situaciones. En cada una de estas combinaciones se puede formular un cálculo que se puede como la calidad de esa solución.

A partir de las relaciones y las diferentes formas que se pueden combinar las peticiones con los centros, podemos aplicar **operadores** para cambiar de un **estado** a otro. Al aplicar una sola operación se puede volver a calcular la calidad de la solución y saber si ese cambio vale la pena o no.

Por lo tanto, en este problema tenemos diferentes **estados**, de los cuales obtenemos un valor numérico aplicando una **función heurística** para saber su *calidad*, y al que podemos aplicar distintos cambios o **operadores** para cambiar a un estado nuevo, más cerca o más lejos del estado óptimo o **estado solución**. Todo esto partiendo de un **estado inicial** totalmente aleatorio o pensado previamente para que supuestamente esté mas cerca del estado solución.

1.2. Estado

Después estudiar las variables y las restricciones de la práctica, hemos llegado a la conclusión de que el conjunto de estados posibles se puede representar de la siguiente forma:

Una lista de camiones, cada uno con sus peticiones para ese día guardadas correctamente, es decir, cumpliendo todas las restricciones definidas anteriormente.

1.2.1. Elementos del estado

Como se puede ver, cada estado tendrá una lista de **camiones** (*Truck*). Este objeto representa los diferentes camiones de los **centros de distribución** que se generan aleatoriamente al principio del problema.

Cada camión esta representado en la clase `Truck.java`. Ésta tiene sus coordenadas de **origen**, es decir, del su centro de distribución y dos listas para guardar los **viajes** que harán ese día (`trips`) y el día siguiente (`extraTrips`). Cada vez que se quieren modificar la cantidad de viajes que se quieren hacer ese día en ese camión se controla que no sean más de cinco ni que superen los kilómetros máximos (como definen las **restricciones**).

Los viajes que hace cada camión (representados en la clase `Trip.java`), no son las **peticiones** hechas por las **gasolineras**, esto puede engañar. Al ver que cada camión podía llevar dos cisternas y con cada cisterna cumplir dos peticiones vimos claro que cada camión, en un viaje, puede completar **dos peticiones** (o *Request*). Esto son los viajes, combinaciones de dos peticiones de las gasolineras. Estas peticiones pueden estar en el mismo lugar o no, pueden ser de gasolineras difentes.

También tenemos que representar las peticiones, para eso tenemos la clase `Request.java`. Este tipo de objeto simplemente nos ofrece la coordenada donde se ha de cumplir esta petición y los días que lleva creada. El beneficio que puede ofrecer es una variable calculada, así nos ahorramos algo memoria.

Para calcular el beneficio total del estado empezamos en la petición (`Request`): Una función nos calcula las ganancias según el día que la petición lleva vigente. Una vez conseguimos este valor, sabiendo el origen de coordenadas inicial de donde sale el camión y las dos peticiones del viaje (`Trip`) podemos saber el beneficio total de cada viaje contando también las pérdidas por los kilómetros recorridos. Si sumamos todos los beneficios de todos los viajes de todos los camiones (`Truck`), conseguimos el beneficio total del estado (`State.class`).

Por último falta guardar en algún sitio todas esas peticiones que no se hacen pero que queremos tener en cuenta para otros posibles estados sucesores. Para almacenar estas peticiones hemos creado camiones fantasma (`GhostTruck.java`) con viajes fantasma (`GhostTrip.java`). Funcionan exactamente igual que sus hermanos no fantasma, pero en cuanto queremos saber su beneficio nos devuelven el peor posible, 0.

1.2.2. Espacio de búsqueda

Si usamos esta representación del estado vemos que nuestro espacio de búsqueda es inmenso. Todas las combinaciones diferentes de todas las diferentes peticiones con todos los camiones disponibles. Esto es un problema, porque teniendo solamente 10 camiones y 100 peticiones ya tenemos 100.000 combinaciones diferentes.

Por eso lo vamos a simplificar un poco con los operadores y el estado inicial nos ayudará a estar cerca de la solución des de un inicio.

1.3. Operadores

Para la representación del estado explicada anteriormente y teniendo en cuenta el espacio de búsquedas tan grande, hemos implementado un único operador, que nos permite intercambiar viajes (par de peticiones) entre dos camiones

diferentes:

```
swapTrip(truck1: Int, trip1: Int, truck2: Int, trip2: Int): Bool
```

Con el que intercambiamos un viaje numero `trip1` de un camión en la posición `truck1`, con otro viaje en la posición `trip2`, de un camión distinto con índice `truck2` en la lista de camiones del estado.

Este operador no nos permite abarcar todo el conjunto de estados solución porque no se pueden intercambiar peticiones entre viajes. Hay que crear la mejor combinación de peticiones posibles en el estado inicial.

1.3.1. Factor de ramificación

Al hacer un cambio ya generamos un estado nuevo. Con n camiones y m peticiones, si agrupamos las peticiones de dos en dos por viajes y suponiendo que no se viola ninguna restricción podemos crear aproximadamente $n \times (m/2)^2$ sucesores diferentes en el peor caso.

1.4. Función heurística

La función heurística es la suma del beneficio total de cada camión con los viajes de ese estado. Recordemos:

$$\%precio = (100 - 2^{dias})\,\%$$

$$Para\ dias = 0\ partimos\ del\ 102\,\%$$

$$d(A, B) =\mid A_x - B_x \mid + \mid A_y - B_y \mid$$

$$2\ euros/km\ recorrido$$

Para calcular el beneficio total del estado empezamos en la petición: Con la fórmula del precio de una petición según los días conseguimos ganancias. Una vez tenemos este valor y con el origen de coordenadas inicial de donde sale el camión y las dos peticiones del viaje podemos saber el beneficio total de cada viaje contando también las pérdidas por los kilómetros recorridos. Si sumamos todos los beneficios de todos los viajes de todos los camiones conseguimos el beneficio total.

$$\sum_{i=1}^{n} Truck(i)$$

$$Truck(i) = \sum_{j=1}^{5} Trip(j)$$

$$Trip(i) = Request_1 + Request_2 - Loss_{km}$$

En el heurístico usado también tiene en cuenta el beneficio que se ganaría el con las peticiones sin atender si se cumplieran el próximo día. Todas las peticiones que no se atienden se resuelven al día siguiente, según el enunciado del problema. Para el experimento especial esto no se tiene en cuenta.

La función de `State.getTotalProfit(): Double` hace este calcula para una instancia de un estado.

1.4.1. Factores que intervienen

En el cálculo del beneficio intervienen la **distancia recorrida** por el camión, que se puede pasar de distancia a dinero sabiendo que $1km$ equivale a un gasto de 2, el **beneficio aportado** al abastecer una gasolinera y el número de **días** que lleva la **petición**.

Para el heurístico del experimento especial no se necesitaba tener en cuenta la suposición que las peticiones sin solventar serían atendidas el día siguiente, por lo que solo se maximizan los beneficios de las peticiones de ese mismo día.

1.5. Estado inicial

El estado inicial es un estado ya completo, con todos los camiones llenos. Tiene que ser de esta manera, si no necesitaríamos más operadores para poder añadir nuevas peticiones o viajes al espacio de búsqueda. Peticiones que son definidas inicialmente en el problema pero que en el estado inicial no están en ningún camión no fantasma.

Teniendo en cuenta que las peticiones no se pueden cambiar de viaje (recordamos que nuestro operador intercambia **viajes** entre camiones y un viaje es un **par de peticiones**) las parejas de peticiones de un viaje han de ser lo mejor posible.

Por ese motivo, al principio de todo cogemos las peticiones de las mismas gasolineras y las juntamos por parejas. Las peticiones sobrante las juntamos entre ellas con la más próxima. Así conseguimos la mínima perdida por kilómetro recorrido. Es cierto que puede ser que, al juntar por parejas las peticiones de esta forma, haya peticiones de hace muchos días, es decir, que aportan muy poco beneficio, pero teniendo en cuenta que también es importante el **beneficio del día siguiente**, aceptamos esta característica del estado inicial

2. Experimentos

2.1. Experimento 1

(A continuación se explican los experimentos 1 y 2)

Para generar una solución inicial según los criterios de calidad del apartado 3.1 (del enunciado de la práctica) lo que hacemos es lo siguiente

Primero, agrupamos por parejas (para formar viajes) el máximo número de peticiones que pertenecen a una misma gasolinera, luego repartimos estos viajes

entre los camiones de manera que no se incumpla ninguna restricción. Cuando ya tenemos los viajes asignados a camiones repartimos las peticiones que nos quedaron sin agrupar anteriormente. En caso de que no tengamos suficientes camiones para atender a todas las peticiones generamos camiones fantasma (*Ghost Trucks*) para simular que tenemos suficientes camiones.

Lo que representan estos *Ghost Trucks* son las peticiones que se atenderán al día siguiente. En el caso de que un viaje no aporte beneficio para ningún camión, éste será asignado a un *Ghost Trucks*.

Esta manera de generar la solución inicial nos permite evitarnos operadores como `add request` o `remove request`, o `add trip` o `remove trip`, que lo único que harían es aumentar exponencialmente la ramificación al generar estados sucesores.

Una vez hemos hecho esto ya tendríamos una posible solución a nuestro problema, aunque no podríamos asegurar que es la mejor solución.

Para mejorar esta solución pensamos en implementar un único operador (`swapTrip(...)`) que nos permite intercambiar viajes entre camiones (también entre *Ghost Trucks*) para encontrar una asignación de viajes distinta que nos aporte más beneficio.

2.2. Experimento 2

(Ver: 2.1. Experimento 1)

2.3. Experimento 3

Para encontrar una solución óptima con Simulated Annealing, a diferencia de Hill Climbing, no le obligamos directamente a buscar una mejora en el beneficio al hacer el swap de viajes entre camiones, de esta manera dejamos que el algoritmo explore estados sucesores mejores y peores que el inicial. Esto, lógicamente, repercute en el tiempo de ejecución pero conseguimos obtener más beneficio que con el algoritmo Hill Climbing para un mismo escenario

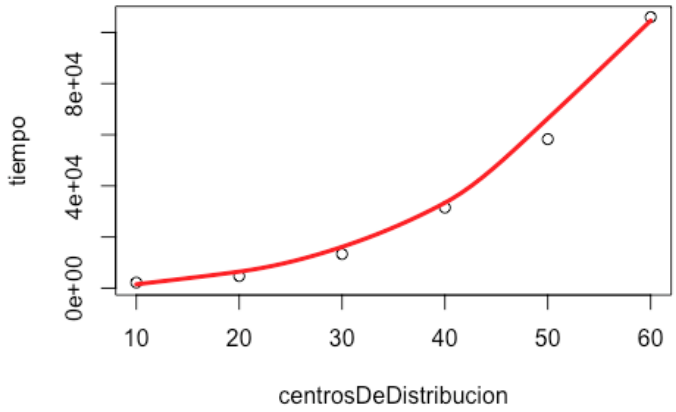
2.4. Experimento 4

Después de haber realizado varias pruebas para ambos algoritmos hemos podido comprobar que Simulated Annealing sigue dando buenos resultados pese a aumentar el número de centros. Sin embargo el tiempo de ejecución para Simulated Annealing aumenta exponencialmente al añadir centros, por ello hemos realizado las pruebas con una cantidad de centros menor a la recomendada por el enunciado.

En las siguiente tabla y gráfica representamos el tiempo de ejecución de Hill Climbling según el número de centros con los que operaban:

Centros de distribución	Tiempo de ejecución (ms)
10	2193
20	4725
30	13344
40	31497
50	58276
60	106000

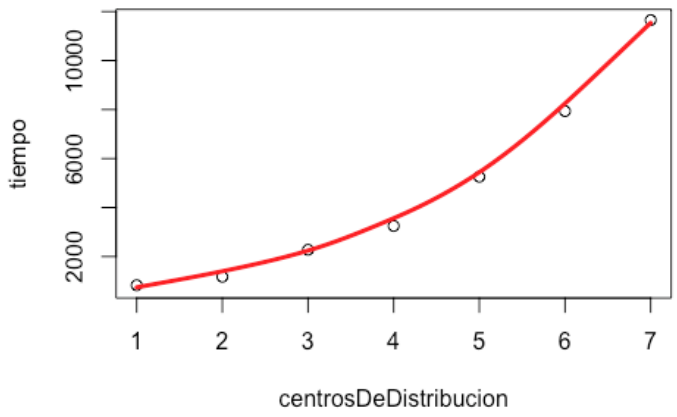
Cuadro 1: Experimento 4.1



En las siguiente tabla y gráfica representamos el tiempo de ejecución de Simulated Annealing según el número de centros con los que operaban:

Centros de distribución	Tiempo de ejecución (ms)
1	830
2	1175
3	2278
4	3249
5	5257
6	7937
7	11652

Cuadro 2: Experimento 4.2



2.5. Experimento 5

Para comprobar cuál es el efecto que tiene dividir en dos la cantidad de centros de distribución pero doblando la cantidad de camiones por centro (en otras palabras 2) hemos primero probado nuestra situación inicial.

Como hipótesis tenemos que no influirá demasiado en el beneficio, ya que la cantidad de camiones será la misma por tanto en número de peticiones será de esperar que sea similar. Posiblemente será un beneficio ligeramente menor, ya que la distancia a recorrer de algunos de los camiones será mayor.

Con 10 centros de distribución, 1 camión por centro, 100 gasolineras y la semilla 1234. Esto nos proporciona el beneficio **95672.00**.

Ahora cambiado los parámetros del problema a 5 centros, 10 camiones y 100 gasolineras, con la misma semilla tenemos un beneficio de **93512.00**.

Se nos hace lógica esta respuesta, ya que la cantidad de peticiones que se puedan resolver será prácticamente la misma (en este caso exactamente la misma), porque la cantidad de camiones es la misma. Ahora bien, hay en el segundo caso se obtiene menos beneficio en el primero, y esto es debido a que el terreno acaparado por 5 centros de distribución es menor al terreno que acaparan 10, por tanto habrá camiones que tendrán que hacer viajes más largos que si hubiese un centro de distribución más cercano a esas gasolineras.

2.6. Experimento 6

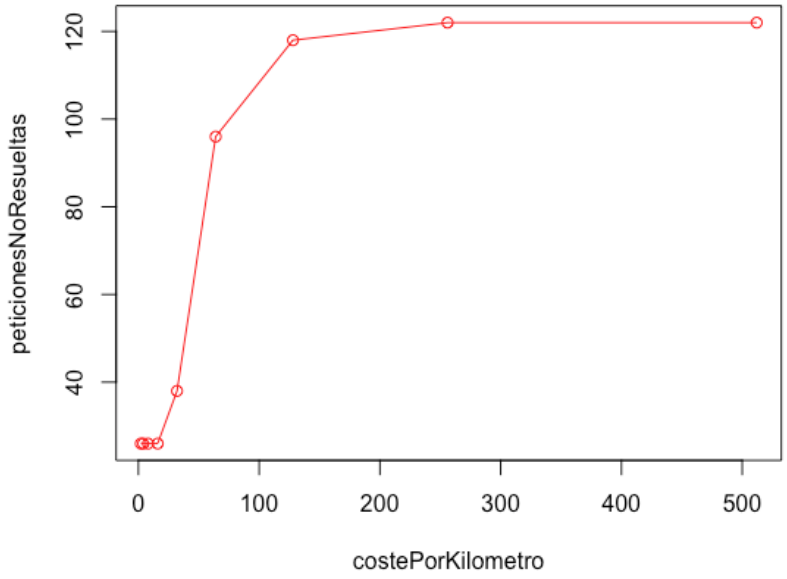
En este experimento queremos ver la relación que hay entre el coste por kilómetro y la cantidad de peticiones servidas.

Como hipótesis creemos que lo obvio es que a medida que aumente el coste del kilómetro aumentará el número de peticiones no servidas.

Hemos usado la situación del experimento especial con 10 centros de distribución, 100 gasolineras y la semilla 1234. Hemos usado Hill-Climbing y nuestro heurístico 2.

Coste por kilómetro	Beneficio generado	Peticiones no resueltas
2	95672	26
4	91236	26
8	82996	26
16	66652	26
32	36784	38
64	10424	96
128	5440	118
256	4020	122
512	4020	122

Cuadro 3: Experimento 6



2.7. Experimento 7

En este experimento queremos ver los efectos que tiene aumentar en una hora la cantidad de horas permitidas por camión, o lo que es lo mismo, aumentar de 640 a 720 los kilómetros permitidos.

No creemos que esto suponga un cambio en el beneficio muy grande, debería o quedarse igual o mejorar un poco. El único caso en el que realmente aumentaria el beneficio es si algun o algunos camiones se dejasen viajes sin usar por falta de kilometraje.

Como en los apartados anteriores hemos usado los parámetros del experimento especial. Que nos proporcionan un beneficio de 93512.0.

Cambiando el número máximo de kilómetros de 640 a 720 y ejecutando el mismo algoritmo con los mismos parámetros recibimos un beneficio de 95668.0

que es sólo ligeramente más que con 640. Eso se nos hacía evidente que la diferencia no iba a ser mucho ya que con 640 ya daba para que todos los camiones usaran sus 5 viajes para solventar 10 peticiones. El aumentar la cantidad de kilómetros solo significa que algún camión deber haber tenido como opción una petición que antes no le era posible acceder y que proporciona más beneficio que la anterior.

Esta hora extra solo proporciona un cambio realmente notable si las gasolineras estuviesen a mucha distancia de los centros de distribución y hubiesen camiones que se quedaran sin usar sus 5 viajes por falta de kilometraje.

3. Código

Se puede conseguir el código en el repositorio público:
`https://github.com/felixarpa/IA-Practica-1-Gasolina`