INTELIGENCIA ARTIFICIAL

PRÁCTICA 1

Búsqueda local

Pau Vallespí
Andrés Eduardo Bercowsky
Guillem González

ÍNDICE

1 Introducción

El objetivo de la práctica es estudiar los algoritmos de búsqueda local Hill Climbing y Simulated Annealing, utilizando la librería AIMA.
Para realizar este estudio, hemos tenido que entender en primer lugar cómo funciona la librería AIMA, además de familiarizarnos con el problema. Lo que hemos tenido que hacer con el problema ha sido: pensar en la óptima representación, los operadores, la generación de la solución inicial, la generación de estados sucesores y la creación de funciones heurísticas. Todo esto nos permite llegar a realizar experimentos con los que evaluar nuestra implementación del problema y así tomar decisiones.

2 Descripción del problema

En la práctica se plantea un problema de planificación de rutas de abastecimiento. Lo que queremos saber es cómo podemos distribuir un recurso entre diferentes localizaciones que necesitan mayor abastecimiento. Aparte se nos presentan las gasolineras, los centros de distribución y los camiones cisterna, de los que hablaremos más adelante. Para resolver el problema supondremos un área geográfica equivalente a una cuadrícula de 100 x 100 km2. El objetivo es encontrar la asignación de peticiones a camiones cisterna y cómo deben resolverse, de forma que obtengan el mayor beneficio y minimizar lo que pierdan por las peticiones no atendidas y también la distancia recorrida. Para calcular la distancia entre

2.1 Elementos del problema

Se diferencian tres elementos principales: los centros de distribución, las gasolineras y los camiones cisterna. Además, también el enunciado habla de peticiones.

2.1.1 Centro de distribución

Una compañía de distribución de combustible tiene n centros de distribución repartidos geográficamente dentro del área comentada anteriormente. En cada centro hay un camión cisterna que permite llevar el combustible a la gasolinera que deba llevarse. Para cada centro de distribución sabemos sus coordenadas (x, y).

2.1.2 Gasolinera

Cada gasolinera tiene varios depósitos donde almacena el combustible y que utiliza para abastecer a los clientes. Asumiremos que las gasolineras utilizan un depósito hasta que termina, y luego cambian al siguiente. Para cada gasolinera sabemos sus coordenadas (x, y) y también podemos saber las peticiones pendientes (de las que hablaremos más adelante).

2.1.3 Camión cisterna

Como hemos dicho, cada centro tiene un camión cisterna. Todas las mañanas a una cisterna le llega un conjunto de peticiones y las debe servir haciendo un número de viajes específicos (lo que deberemos determinar nosotros mediante los algoritmos). Una cisterna en un viaje lo que hace es, primero llenarse completamente, sirve a las gasolineras que tiene asignadas en ese viaje (como máximo 2) y vuelve al centro de distribución. Esto se repite hasta agotar las peticiones que ha recibido.

2.1.4 Peticiones

El enunciado nos dice que cada gasolinera genera sus peticiones al avisar a la compañía de distribución cuando uno de sus depósitos se queda vacío. De esta forma cada día se dispone de un listado de peticiones de gasolineras que han ido llegado y todavía no se han podido servir. Asumiremos que puede haber más de una petición para atender por gasolinera. Aparte, debemos tener en cuenta que dependiendo de los días que lleva pendiente una petición, el precio que se cobra va bajando siguiendo la fórmula:

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

2.2 Restricciones y criterios de la solución

- Las cisternas viajan a 80km/h y trabajan durante 8 horas, por tanto pueden recorrer como mucho 80*8 = 640 km por día.
- Una cisterna no puede realizar más de 5 viajes diarios.
- Debe maximizarse el beneficio que obtenemos teniendo en cuenta la pérdida que supone no atender una petición el día actual si suponemos que no se podrán atender al día siguiente.
- El valor de un depósito es 1000.
- Debemos minimizar también la distancia recorrida asumiendo que existe un coste fijo por km recorrido. Este coste es 2.

2.3 Justificación del uso de búsqueda local

En este problema en concreto, nos encontramos con que queremos llegar a una solución pero no nos importa saber el camino específico que debe tomarse para llegar a la solución. En la clase de problemas donde no importa saber cuál es el camino para llegar a la solución, utilizaremos la búsqueda local. Con este algoritmo, operamos utilizando un único nodo y nos movemos generalmente a sus vecinos. Aunque la búsqueda local no sea sistemática, tiene la ventaja de que utiliza muy poca memoria (normalmente constante) y que normalmente

encuentran buenas soluciones en un espacio de estados muy grande o infinito, para los que los algoritmos sistemáticos no encuentran.

Además, los algoritmos de búsqueda local son útiles para resolver problemas de optimización, cuyo objetivo es encontrar el mejor estado en relación a una función objetivo. Como sabemos, el problema de las gasolineras trata de un problema de optimización puesto que queremos maximizar las ganancias y minimizar la distancia recorrida.

3 Implementación del projecte

3.1 Representación del estado

En aquest problema, per fer la representació de l'estat hem emprat les següents estructures de dades:

- Un objeto Gasolineras gas, con la información de las gasolineras del problema y las peticiones pendientes de resolver.
- Un objeto CentrosDistribucion cien, con la información de los centros de distribución del problema.
- Un ArrayList de enteros kilómetros con el número de kilómetros que realiza cada uno de los camiones cisterna.
- Una estructura bidimensional ArrayList<ArrayList<pair<Integer,Integer> >> itinerarios donde se almacenan las gasolineras que visita un camión cisterna por viaje. Cada camión cisterna se representa por una posición del ArrayList más externo, y cada viaje de cada camión cisterna ocupa una posición del ArrayList más interno.
- Un pair<Integer,Integer> representa un viaje, de tal forma que los enteros representan un identificador de gasolinera. Los viajes pueden ser de tres tipos:
    1. **Dos peticiones de dos gasolineras:** El primer entero representa el identificador de la primera gasolinera del viaje, y el segundo entero representa el identificador de la segunda gasolinera.
    2. **Dos peticiones de una gasolinera:** Ambos enteros representan el identificador de la única gasolinera del viaje.
    3. **Una petición de una única gasolinera:** El primer entero representa el identificador de la única gasolinera del viaje, y el segundo tiene asignado un valor -1, indicando el fin del viaje y el regreso al centro de distribución.
- Una estructura bidimensional ArrayList<ArrayList<pair<Integer,Integer> >> peticiones donde se almacenan los días que llevan las peticiones de las gasolineras. Cada diario de la estructura bidimensional con índices ij corresponde al mismo viaje a itinerarios, de tal forma que itinerarios[i][j].first corresponde a la primera petición a realizar en ese viaje y peticiones[i][j].first corresponde a los días que trae esta petición. Lo mismo sucede por los segundos elementos del digerir a itinerarios y

peticiones, pero éstos constituyen la segunda petición a suplir en este viaje. Como itinerarios, encontramos los siguientes casos:

1. **Dos peticiones de dos gasolineras:** El primer entero representa los días que lleva activa la primera petición a suplir a lo largo del viaje, y el segundo entero representa los días que lleva activa la segunda petición.
2. **Dos peticiones de una gasolinera:** Los dos enteros representan los días que llevan dos peticiones distintas activas. Como son peticiones distintas, el número de días que llevan activas las peticiones pueden ser iguales o diferentes.
3. **Una petición de una única gasolinera:** El primer entero representa el número de días que lleva la petición activa de la primera petición a suplir, y el segundo tiene asignado un valor nulo, indicando el fin del viaje y el regreso al centro de distribución.

- Un entero beneficio que indica la cantidad de dinero que ganaríamos haciendo el itinerario del estado, teniendo en cuenta lo que ganamos por resolver las peticiones, así como las pérdidas debidas al precio por kilómetro.
- Unos enteros numViajes, numKilómetros, precioKm y valorDepósito que nos indican el número de viajes máximo que puede hacer un camión cisterna, el número máximo de kilómetros que puede hacer una cisterna, el precio por kilómetro que debemos pagar por suplir las demandas y el valor de un depósito, respectivamente.

3.2 Análisis de los operadores

Cómo utilizaremos técnicas de búsqueda local dentro de un espacio de soluciones, partiremos de una mala solución inicial, si hablamos en términos de optimalidad. Con esta solución inicial, podemos asegurar que tendrán unas peticiones asignadas, de forma que añadamos el máximo número de peticiones por los camiones cisterna de los que disponemos. Contando con esta suposición, también podemos asegurar que los intercambios entre dos gasolineras de dos viajes distintos será el operador predominante en nuestro conjunto de operadores, ya sea entre dos camiones cisterna o entre distintos viajes de un mismo camión cisterna.

Aparte de éste, también encontramos que, por optimización de la solución, camiones cisterna que antes no aceptaban recibir más peticiones, en un momento dado sí que aceptan más. De esta necesidad surgen dos nuevos operadores: mover una petición de un camión cisterna a otro y aceptar una nueva petición por un camión cisterna dado.

Como conclusión, los operadores escogidos para resolver este problema quedan formalizados de la siguiente forma:

- **swapGasolineras (Integer Ci, Integer Cj, Integer Vino, Integer Vj, Integer Pi, Integer Pj):** Operador que intercambia dos peticiones de dos gasolineras de

dos camiones cisterna. Ci y Cj representan los camiones cisterna a los que se realiza el swap; Vino y Vj son los viajes que se harán intercambio; Pi y Pj indican cuál de las dos gasolineras de cada par de gasolineras de cada viaje realizará el swap.

- **addPeticion (Integer Ci, Integer Gj, Integer Pj):** Operador que mueve una petición pendiente a un itinerario de un camión cisterna. Ci representa al camión cisterna que aceptará la nueva petición; Gj representa a la gasolinera que da la petición; Pj representa la petición en cuestión. Esta petición ocupará la posición final del itinerario del camión cisterna dado.

- **swapPeticionNoAsignada(Integer Ci, Integer Vino, Integer Pi, Integer Gj, Integer Pj):** Operador que intercambia una petición Pi de un viaje Vino ya asignada a una cisterna Ci determinada, por una petición a asignar Pj de una gasolinera Gj. Cuando se realiza este intercambio, la petición asignada antes del swap vuelve a su gasolinera original como petición no asignada.

3.3 Representación de la solución inicial

Para generar la solución inicial, partimos de dos estrategias algorítmicas totalmente opuestas: un algoritmo aleatorio (random) y un algoritmo voraz (greedy). Decimos que son totalmente opuestos porque el algoritmo voraz escoge, a partir de criterios de proximidad, la mejor solución por parte del problema, intentando buscar la mejor solución para el problema general; mientras que el algoritmo aleatorio asigna a cada parte del problema una solución aleatoria, sin basarse en ningún criterio.

Vemos un poco más en profundidad qué operaciones y decisiones realizan ambos algoritmos:

- **Inicialización aleatoria del estado inicial:** Este algoritmo asigna, por cada camión cisterna de cada centro de distribución, peticiones aleatorias de gasolineras aleatorias hasta que el número de viajes por camión cisterna alcanza el máximo, hasta que la cantidad de kilómetros alcanza el máximo o , si se da el caso, hasta que no queden peticiones por suplir.
  Cada vez que se asigna una petición a una cisterna se contabilizan los nuevos kilómetros que realiza, el número de viajes en caso de que se realice en un viaje nuevo y el incremento (o decremento) de beneficio.
  El algoritmo devuelve el estado inicializado cuando toda cisterna tiene su itinerario al máximo, de modo que cada cisterna está cerca del máximo de viajes y peticiones que puede realizar, o cuando ha asignado todas las peticiones pendientes.

- **Inicialización voraz del estado inicial:** Este algoritmo utiliza una técnica voraz, concretamente selecciona las peticiones de las gasolineras más cercanas de un centro de distribución y las asigna al centro, de tal forma que inserta el mayor número posible de peticiones por camión cisterna .

Este algoritmo produce un resultado más cercano a una solución óptima que el algoritmo aleatorio, por tanto provocará que, a priori, hill climbing y simulated Annealing expanden menos nodos que el algoritmo aleatorio.

3.4 Generación de los estados sucesores

Para generar los estados sucesores se implementan dos métodos diferentes, uno por cada algoritmo: por hill climbing tenemos la función generadora de estados sucesores 1, y por simmulated annealing tenemos la función generadora de estados sucesores 2:

- **SucesorFunction1:** Este algoritmo parte de una lista vacía de estados y del estado actual. Empleando los operadores disponibles, aplicamos combinatoria para conseguir todos los estados posibles derivados de aplicar este cambio en el estado actual, de modo que vemos todos los escenarios posibles. Aplicamos esta lógica por todos los operadores. Por último, obtenemos una lista de estados sucesores, de donde hill climbing elegirá el estado sucesor que mejor heurístico tenga y mejor calidad de solución dé.

- **SucesorFunction2:** Este algoritmo parte de un estado inicial y un estado clon. A partir de aleatoriedad, se escogen el operador que se utilizará para obtener el sucesor y cuyos parámetros el operador obtendrá este estado sucesor. Posteriormente, se crea este nuevo estado aplicando el operador que se ha escogido aleatoriamente y finalmente se añade a la lista de sucesores.

3.5 Función heurística

La función heurística debe minimizar el número de kilómetros y viajes que los camiones cisterna realizan, así como maximizar los beneficios de suplir las peticiones de las gasolineras. Así pues, consideramos las siguientes premisas: sea $\boldsymbol{PC}$ el conjunto de peticiones completadas por los camiones cisterna, $\boldsymbol{PNC}$ el conjunto de peticiones no asignadas, $\boldsymbol{vD}$ el beneficio obtenido de llenar una cisterna, $\boldsymbol{km}$ el número de kilómetros totales que hacen los camiones cisterna, $\boldsymbol{pK}$ el precio por kilómetro, el heurístico se define de la siguiente forma:

$$h(x) = -\left(\left[\sum_{\forall i \in PC}[(100 - 2^{dias_i}) \cdot vD]\right] - pK \cdot km - \sum_{\forall i \in PNC}[(100 - 2^{dias_i}) \cdot vD]\right)$$

Por último, obtenemos un indicador que nos proporciona una forma numérica y sencilla de comparar la calidad de los estados sucesores, de tal modo que aquellos que minimicen los costes y maximicen los beneficios tendrán un heurístico menor, mientras que aquellos que tengan una calidad de solución más baja, es decir unos costes más elevados con beneficios más bajos, tendrán un heurístico mayor.

4 Experimentos

1. **Determinar qué conjunto de operadores da mejores resultados para una función heurística que optimice el criterio de calidad del problema (3.1) con un escenario en el que el número de centros de distribución es 10, hay un camión en cada centro y el número de gasolineras es 100. Deberéis usar el algoritmo de Hill Climbing. Escoged una de las estrategias de inicialización de entre las que proponéis. A partir de estos resultados deberéis fijar los operadores para el resto de experimentos.**

De entre todos los operadores, queremos ver el que nos aporta una solución mejor para unos valores determinados y el algoritmo de Hill Climbing. Para ello, bloquearemos todos los operadores menos el que estamos estudiando en la iteración i-ésima del experimento. De esta manera, vemos cuántos nodos expande el operador, cuánto avanza el algoritmo, el tiempo que ha consumido y la calidad de la solución encontrada.

Definimos formalmente el experimento de la siguiente manera:

Observación	¿Existen operadores que generen una solución final mejor que otros?
Planteamiento	A partir de una solución aleatoria, escogemos diversos estados iniciales y aplicamos un único operador por solución.
Hipótesis	Todos los operadores obtienen una solución con calidad igual (H0), o existen desigualdades en la calidad de solución que generan los operadores (H1).
Variable/s independiente/s	Solución inicial generada aleatoriamente.
Variable/s dependiente/s	Tiempo de ejecución, beneficio, número de nodos expandidos.
Variable/s controlada/s	Semilla del problema, número de centros, número de gasolineras, multiplicidad de los centros.
Método	• Partimos de una semilla cualquiera, la misma para toda ejecución del algoritmo (semilla 5479). Definimos el número de centros a 10, el número de gasolineras a 100 y un camión por centro. También definimos que el algoritmo a usar es Hill Climbing y la solución inicial es generada aleatoriamente. • Ejecutamos el algoritmo 10 veces por operador seleccionado, siendo un total de 30 veces. • Recogemos, para cada ejecución, los resultados de tiempo de ejecución, calidad de solución final y

	número de nodos expandidos. • Analizamos los resultados y comparamos.

Tras la ejecución del experimento, obtenemos los siguientes resultados para las variables analizadas:

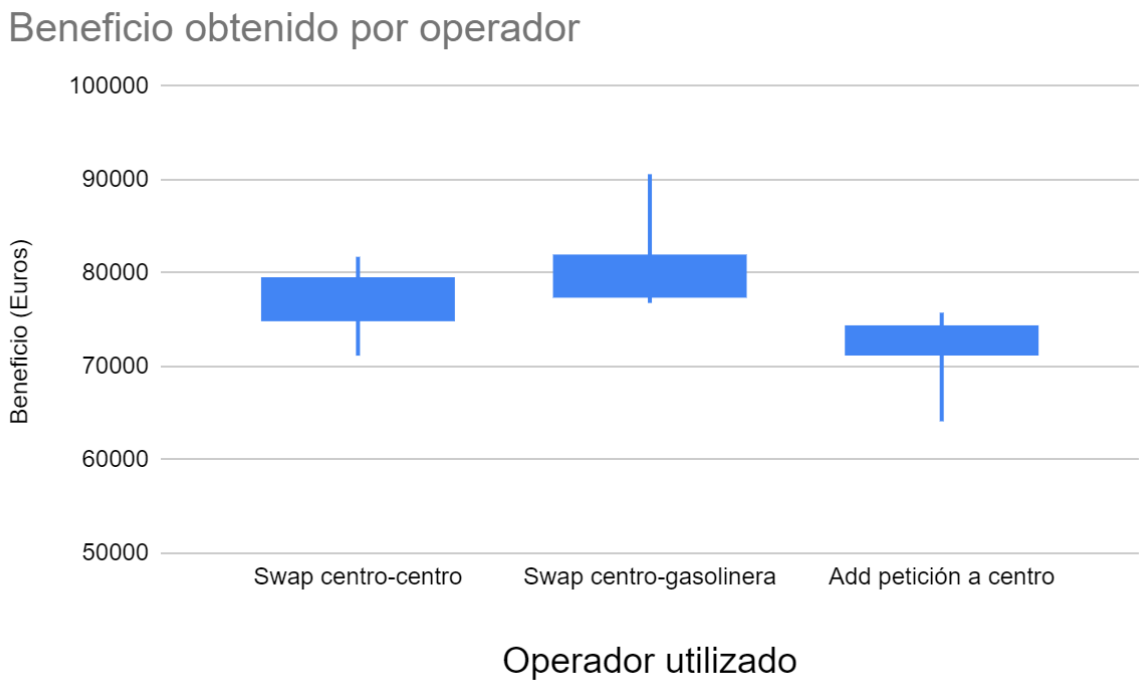
Operador swap centro-centro			
réplica	tiempo ejecución (ms)	beneficio	nodos expandidos
1	51	78.156	3
2	73	80.196	5
3	18	71.160	3
4	22	77.572	3
5	38	76.292	7
6	27	74.644	4
7	21	81.736	3
8	16	76.824	3
9	19	79.732	4
10	13	73.412	3
Valores medios:	**29,9**	**76.972,4**	**3,8**

Operador swap centro-gasolinera			
réplica	tiempo ejecución (ms)	beneficio	nodos expandidos
1	40	80.292	6
2	46	76.800	3
3	41	80.516	5
4	23	82.188	5
5	11	79.772	3

6	18	76.828	6
7	21	78.760	4
8	20	82.996	5
9	17	77.184	4
10	13	90.588	4
Valores medios:	**25,0**	**79.592,4**	**4,5**

Operador add petición a centro			
réplica	tiempo ejecución (ms)	beneficio	nodos expandidos
1	1	73.260	1
2	1	74.096	1
3	2	70.928	1
4	2	64.128	1
5	0	74.216	1
6	1	75.756	1
7	1	69.780	1
8	1	73.016	1
9	0	75.396	1
10	1	72.752	1
Valores medios:	**1,0**	**72.332,8**	**1**

Para ilustrar los resultados de forma visualmente esclarecedora, veamos los resultados en gráficos y comparemos los valores esperados de la muestra, así como su desviación muestral:



Finalmente, podemos concluir que, como el swap entre gasolinera y centro obtiene mejores resultados para nuestras variables, afirmamos que este operador aporta una mejor solución a nuestro problema. Por lo tanto, rechazamos la hipótesis nula y aceptamos la alternativa ⇒ Los operadores nos llevan a soluciones finales con diferente beneficio.

Hemos de destacar también que, si observamos los resultados experimentales, el operador add no genera prácticamente ningún estado sucesor. Esto es debido a que la solución inicial se genera de forma random e intenta llenar todo el itinerario de la cisterna, de manera que no pueda aceptar nuevas peticiones. Si este suceso ocurre (en nuestro caso, ocurre la mayoría de las veces) obtenemos que el operador add no puede añadir ninguna petición nueva, por lo que no genera estados sucesores. Aún así, su influencia en la solución final suele ser muy minoritaria.

2. Determinar qué estrategia de generación de la solución inicial da mejores resultados para la función heurística usada en el apartado anterior, con el escenario del apartado anterior y usando el algoritmo de Hill Climbing. A partir de estos resultados deberéis fijar también la estrategia de generación de la solución inicial para el resto de experimentos.

De entre todos los algoritmos de generación de estado inicial, queremos ver el que nos aporta una solución final mejor para Hill Climbing y la función heurística del experimento número 1. Para ello, haremos un número determinado de réplicas por algoritmo de generación de solución inicial. De esta manera, vemos cuánto avanza el algoritmo, el tiempo que ha consumido y la calidad de la solución encontrada.

Definimos formalmente el experimento de la siguiente manera:

Observación	¿Existen algoritmos generadores de solución inicial que generen una solución inicial mejor que otros?
Planteamiento	A partir de un número de soluciones iniciales por algoritmo generador de solución inicial, de diferentes semillas, ejecutamos Hill Climbing.
Hipótesis	Todos los algoritmos generadores de solución inicial obtienen una solución igual de buena (H0), o existen desigualdades en la calidad de solución que se obtiene con los diferentes algoritmos (H1).
Variable/s independiente/s	Tipo de generadora de estado inicial, semilla del problema.
Variable/s dependiente/s	Tiempo de ejecución, beneficio, número de nodos expandidos.
Variable/s controlada/s	Número de centros, número de gasolineras, multiplicidad de los centros.
Método	● Partimos de una semilla aleatoria (semilla 4781). Definimos el número de centros a 10, el número de gasolineras a 100 y un camión por centro. También definimos que el algoritmo a usar es Hill Climbing. ● Ejecutamos el algoritmo 10 veces por solución inicial, siendo un total de 20 veces. Para la solución voraz, damos una semilla aleatoria para cada réplica. ● Recogemos, para cada réplica, los resultados de tiempo de ejecución, calidad de solución final y número de nodos expandidos. ● Analizamos los resultados y comparamos.

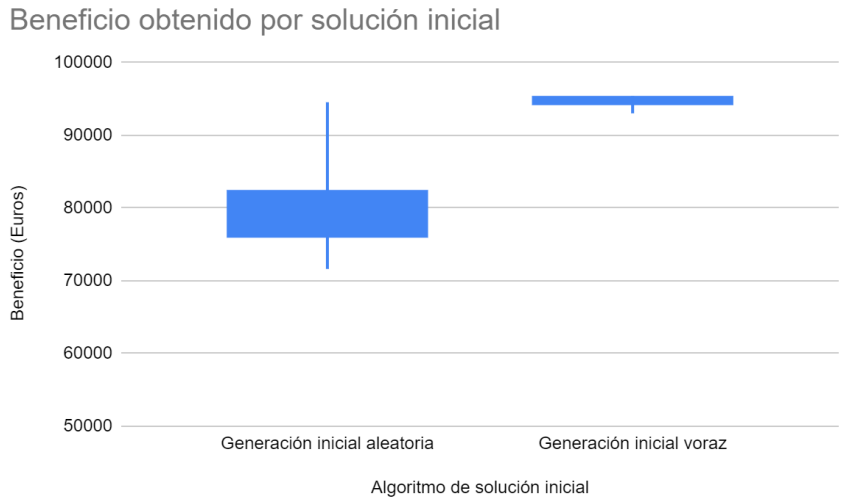
Tras la ejecución del experimento, obtenemos los siguientes resultados para las variables analizadas:

Solución inicial aleatoria			
réplica	tiempo ejecución (ms)	beneficio	nodos expandidos
1	38	94.548	5
2	29	79.220	4
3	31	83.000	5
4	26	83.192	3
5	23	72.756	5
6	17	77.416	5
7	29	75.780	6
8	16	77.160	4
9	21	71.628	3
10	15	80.148	3
Valores medios:	**24,5**	**79.484,8**	**4,3**

Solución inicial voraz			
réplica	tiempo ejecución (ms)	beneficio	nodos expandidos
1	60	94.524	1
2	93	94.320	2
3	89	95.400	1
4	117	95.276	2
5	106	94.526	2
6	118	94.452	2
7	42	94.280	1

8	97	95.248	2
9	66	93.028	2
10	48	95.144	1
Valores medios:	**83,6**	**94.619,8**	**1,6**

Para ilustrar los resultados de forma visualmente esclarecedora, veamos los resultados en gráficos y comparemos los valores esperados de la muestra, así como su desviación muestral:



En el gráfico observamos que, para la solución aleatoria, los datos se esparcen más en el gráfico, y tienen una desviación mucho mayor que la solución voraz. Esto se debe al propio criterio que siguen los algoritmos (la estrategia voraz siempre intentará buscar la configuración óptima del problema, por lo tanto observaremos que los datos siempre se agrupan, de la manera más estrecha posible, en un valor determinado para unos parámetros del problema determinados).

Finalmente, podemos concluir que, como el generador de solución inicial greedy obtiene mejores resultados para nuestras variables, afirmamos que la solución voraz aporta una mejor solución a nuestro problema, por lo tanto refutamos la hipótesis nula y aceptamos la hipótesis alternativa.

Sin embargo, para el resto de experimentos utilizaremos la solución random, pues es la que expande más nodos y nos permite experimentar más con los operadores y el heurístico.

3. **Determinar los parámetros que dan mejor resultado para Simulated Annealing con el mismo escenario, usando la misma función heurística, los operadores y la estrategia de generación de la solución inicial escogidos en los experimentos anteriores.**

Simulated Annealing consta de cuatro parámetros: el número de iteraciones máximas, el número de iteraciones por temperatura, el parámetro k y el parámetro lambda. Podemos alterar los valores de estos parámetros para, a través de experimentación, encontrar aquellos que son más apropiados para el modelo planteado, y que generan una solución final de mayor beneficio.

Para ello, definimos unos parámetros fijos para cada vez que experimentamos con un parámetro variable (por ejemplo, si estudiamos el comportamiento de lambda dejamos el resto de parámetros con valores fijos). De esta manera, definimos el experimento de la siguiente forma:

Para cada parámetro de Simulated Annealing, definimos un rango de posibles valores con los que experimentar. Realizamos 20 réplicas por parámetro, controlando la semilla, el número de centros, el número de gasolineras y la multiplicidad de los centros. Para cada una de esas réplicas, el parámetro variable tiene un valor determinado dentro de ese rango establecido. Analizamos cuánto avanza el algoritmo, el tiempo que ha consumido y la calidad de la solución encontrada y establecemos un criterio para definir para qué valores los parámetros muestran un carácter óptimo para el problema.

Definimos formalmente el experimento de la siguiente manera:

Observación	¿Existen valores de los parámetros de Simulated Annealing que proporcionen resultados mejores?
Planteamiento	A partir de un rango de valores para cada parámetro de Simulated Annealing, ejecutamos el algoritmo.
Hipótesis	Todo valor dentro del rango definido por parámetro muestra el mismo comportamiento para Simulated Annealing (H0), o existen desigualdades en el comportamiento de Simulated Annealing dependiendo del valor del parámetro (H1).
Variable/s independiente/s	Número de centros, número de gasolineras, parámetros de Simulated Annealing
Variable/s dependiente/s	Tiempo de ejecución, beneficio
Variable/s controlada/s	Número de centros, número de gasolineras, multiplicidad de los centros, semilla del problema, valores de los parámetros

	fijos de Simulated Annealing.
Método	• Partimos de una semilla aleatoria. Definimos el número de centros a 10, el número de gasolineras a 100 y un camión por centro. • También definimos que el algoritmo a usar es Simulated Annealing y un parámetro como variable, con un valor dentro del rango, y el resto valores fijos. Seguimos esa lógica para todo parámetro. • Ejecutamos el algoritmo 20 veces por parámetro variable, siendo un total de 80 veces. • Recogemos, para cada réplica, los resultados de tiempo de ejecución, calidad de solución final y número de nodos expandidos. • Analizamos los resultados y comparamos.

Tras la ejecución del experimento, obtenemos los siguientes resultados para las variables analizadas:
Número de iteraciones máximas: Variable
Número de iteraciones por temperatura: 10
Parámetro k: 125
Parámetro lambda: 0.0001

Parámetro variable: Iteraciones máximas			
réplica	tiempo ejecución (ms)	beneficio	iteraciones máximas
1	1	68.504	2^1
2	2	74.684	2^2
3	1	65.892	2^3
4	1	71.816	2^4
5	2	69.504	2^5
6	1	67.596	2^6
7	5	71.572	2^7
8	7	70.800	2^8
9	10	70.472	2^9

10	23	81.372	2^{10}
11	17	77.732	2^{11}
12	31	88.036	2^{12}
13	64	88.448	2^{13}
14	98	88.164	2^{14}
15	165	88.676	2^{15}
16	189	88.608	2^{16}
17	371	88.372	2^{17}
18	442	88.792	2^{18}
19	473	88.652	2^{19}
20	628	89.100	2^{20}
Valores medios:	**126.55**	**79.339,6**	$\sim 2^{17}$

Número de iteraciones máximas: 150000
Número de iteraciones por temperatura: Variable
Parámetro k: 125
Parámetro lambda: 0.0001

Parámetro variable: Iteraciones máximas por temperatura			
réplica	tiempo ejecución (ms)	beneficio	iteraciones por temperatura
1	147	88.780	2^{0}
2	163	88.732	2^{1}
3	154	89.056	2^{2}
4	163	88.968	2^{3}
5	150	88.732	2^{4}
6	137	88.884	2^{5}

7	127	88.536	2^6
8	128	88.656	2^7
9	130	88.588	2^8
10	134	88.892	2^9
11	134	88.808	2^{10}
12	138	88.888	2^{11}
13	127	89.040	2^{12}
14	135	88.796	2^{13}
15	138	88.900	2^{14}
16	151	89.116	2^{15}
17	171	89.448	2^{16}
18	145	89.224	2^{17}
19	136	88.892	2^{18}
20	131	89.120	2^{19}
Valores medios:	141,95	88.902,6	$\sim 2^{16}$

Número de iteraciones máximas: 150000
Número de iteraciones por temperatura: 10
Parámetro k: Variable
Parámetro lambda: 0.0001

Parámetro variable: K			
réplica	tiempo ejecución (ms)	beneficio	k
1	658	88.876	2^0
2	863	88.748	2^1
3	769	89.380	2^2

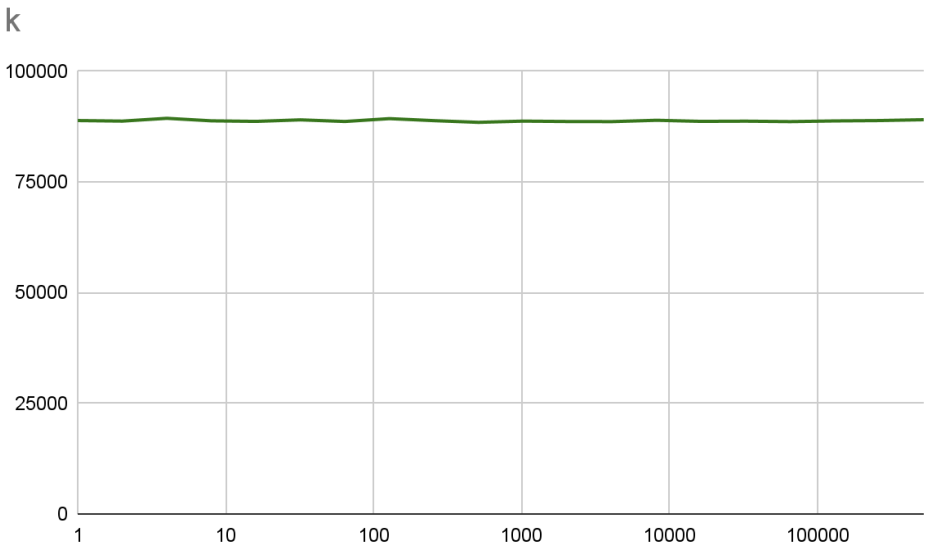
4	546	88.812	2^3
5	378	88.684	2^4
6	379	89.032	2^5
7	372	88.664	2^6
8	200	89.308	2^7
9	351	88.852	2^8
10	457	88.468	2^9
11	391	88.748	2^{10}
12	564	88.656	2^{11}
13	328	88.632	2^{12}
14	290	88.936	2^{13}
15	253	88.688	2^{14}
16	217	88.736	2^{15}
17	235	88.632	2^{16}
18	209	88.788	2^{17}
19	316	88.872	2^{18}
20	386	89.064	2^{19}
Valores medios:	408,1	88.828,8	$\sim 2^{16}$

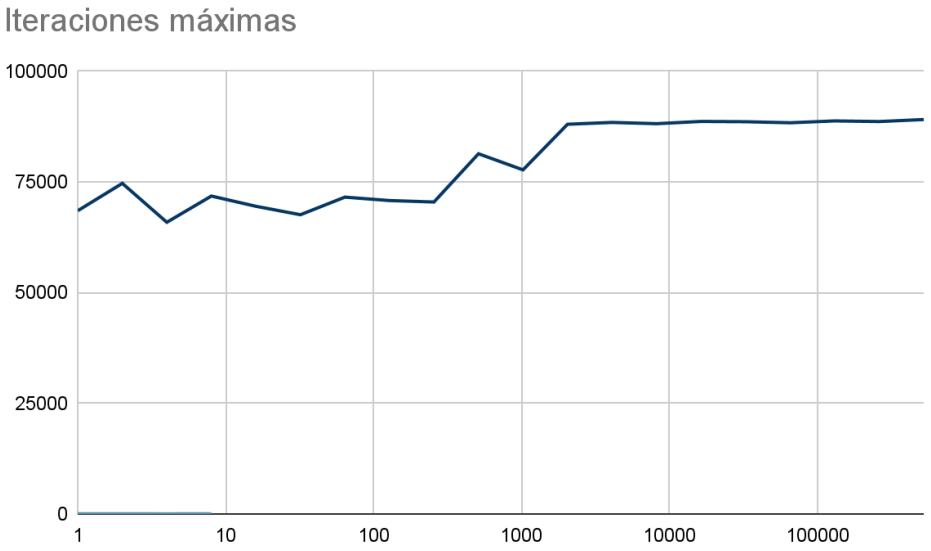
Número de iteraciones máximas: 150000
Número de iteraciones por temperatura: 10
Parámetro k: 125
Parámetro lambda: Variable

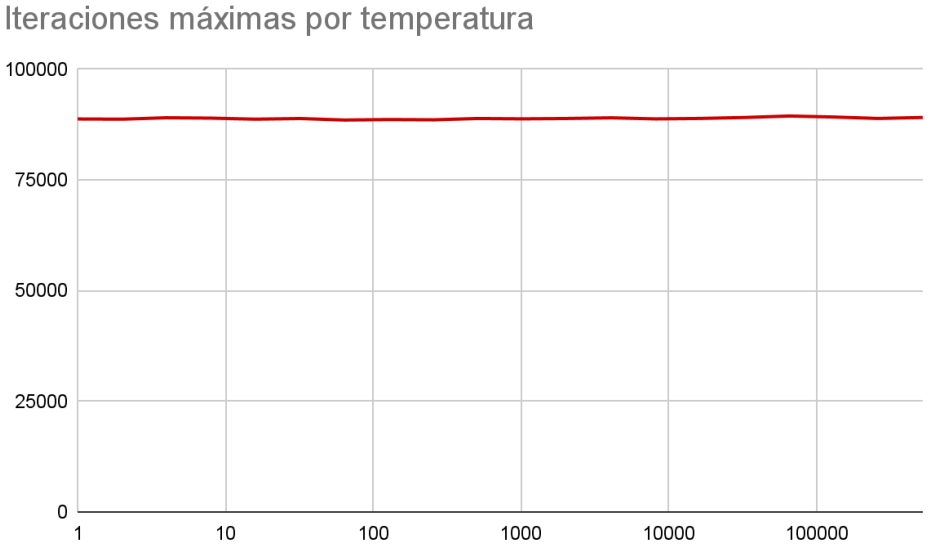
Parámetro variable: Lambda			
réplica	tiempo ejecución (ms)	beneficio	lambda

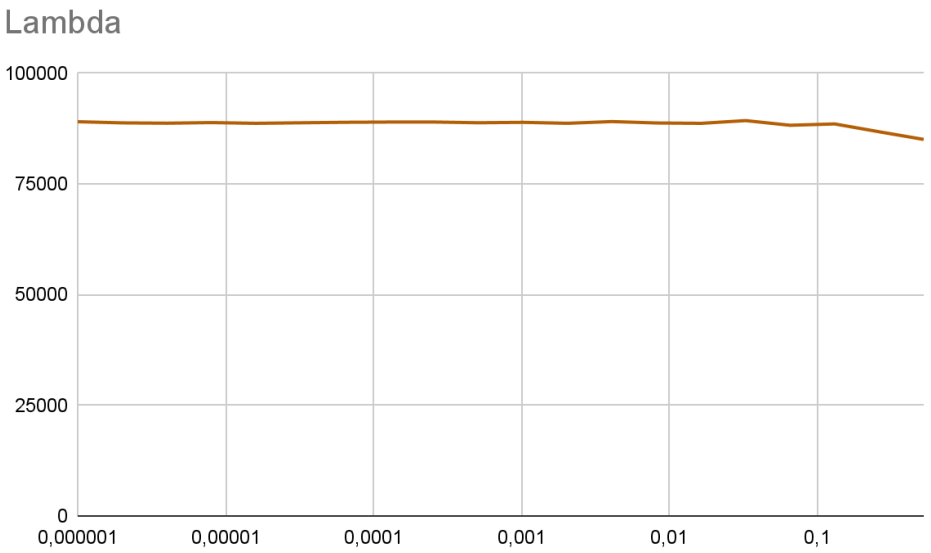
1	143	89.060	10^{-6}
2	153	88.804	$2*10^{-6}$
3	147	88.728	$4*10^{-6}$
4	145	88.868	$8*10^{-6}$
5	146	88.700	$1.6*10^{-5}$
6	134	88.820	$3.2*10^{-5}$
7	127	88.932	$6.4*10^{-5}$
8	129	88.988	$1.28*10^{-4}$
9	127	88.992	$2.56*10^{-4}$
10	137	88.828	$5.12*10^{-4}$
11	134	88.920	$1.024*10^{-3}$
12	124	88.708	$2.048*10^{-3}$
13	125	89.096	$4.096*10^{-3}$
14	82	88.780	$8.192*10^{-3}$
15	41	88.696	0.016328
16	19	89.312	0.032768
17	9	88.260	0.065536
18	4	88.560	0.131072
19	3	86.788	0.262144
20	2	85.060	0.524288
Valores medios:	96,55	88.545	~0.05

Para ilustrar los resultados de forma visualmente esclarecedora, veamos los resultados en gráficos. En estos gráficos situamos en el eje x las variables independientes, y en el eje y las variables dependientes, tal como se muestra:


k
100000
75000
50000
25000
0
1
10
100
1000
10000
100000


Iteraciones máximas
100000
75000
50000
25000
0
1
10
100
1000
10000
100000


Iteraciones máximas por temperatura
100000
75000
50000
25000
0
1
10
100
1000
10000
100000



Finalmente, podemos concluir que, como algunos valores como las iteraciones máximas proporcionan mejores resultados para la solución final, tenemos que rechazar la hipótesis nula y aceptamos la alternativa, encontrando la siguiente lógica para los parámetros:

- Para el número de iteraciones máximas, tenemos que éste valor es directamente proporcional al beneficio e inversamente proporcional al tiempo de ejecución. Nuestra decisión es quedarnos con el valor 150.000
- Para el número de iteraciones máximas por temperatura, tenemos que tanto el tiempo de ejecución como el beneficio obtenido se mantienen casi constantes a lo largo del experimento. Por ello, es indiferente que valor tomar dentro de este rango y hemos decidido elegir 10.
- Para k, tenemos que ésta no hace que el beneficio varíe demasiado, así que decidimos quedarnos con k = 125 (la que habíamos planteado originalmente).
- Para lambda, tenemos que para valores muy pequeños (a partir de 10^{-6}), los resultados son favorables. En cambio, a partir de un valor de 0.1, el resultado va empeorando. Por ello, decidimos elegir un valor pequeño como es 0.0001.

4. Dado el escenario de los apartados anteriores, estudiad cómo evoluciona el tiempo de ejecución para hallar la solución para valores crecientes de los parámetros del problema siguiendo la proporción 10:100 para centros y gasolineras. Comenzad con 10 centros de distribución e incrementad el número de 10 en 10 hasta que se vea la tendencia. Usad el algoritmo de Hill Climbing y Simulated Annealing y la misma función heurística que antes. Comprobad que los parámetros para Simulated Annealing que habéis hallado en el apartado anterior siguen dando buenos resultados al aumentar el tamaño del problema.

En este experimento creamos dos réplicas del mismo, uno para Hill Climbing y otro para Simulated Annealing. Utilizaremos el mismo heurístico, operadores y función generadora del estado inicial (aleatoria). Como dice el enunciado, utilizaremos los valores que configuran Simulated Annealing más adecuados para el problema, los encontrados en el experimento anterior. Iremos aumentando el tamaño del problema, siempre con la misma semilla generadora, para ver cuál es la tendencia de crecimiento del tiempo de ejecución del problema.

Definimos formalmente el experimento de la siguiente manera:

Observación	¿Varía el tiempo de ejecución cuando aumentamos el número de centros y de gasolineras?
Planteamiento	A partir de una proporción 10:100 de centros y gasolineras, vamos aumentando el tamaño del problema para observar qué cambios se presentan.
Hipótesis	Aumentar el número de centros y de gasolineras no varía el tiempo de ejecución del algoritmo (H0), o realmente modificar esos valores hace que el tiempo de ejecución varíe (H1).
Variable/s independiente/s	Número de centros, número de gasolineras, tipo de algoritmo (Hill Climbing o Simulated Annealing). En caso de S.A. también sus parámetros.
Variable/s dependiente/s	Tiempo de ejecución, beneficio, número de nodos expandidos.
Variable/s controlada/s	Multiplicidad de los centros, semilla del problema, valores de los parámetros de Simulated Annealing
Método	• Partimos de una semilla aleatoria, multiplicidad fija, solución inicial aleatoria y parámetros de Simulated Annealing fijos (aquellos que proporcionan una solución mejor, encontrados en el experimento anterior). • Para cada tipo de algoritmo, lo ejecutamos 10 veces con diferente número de centros y gasolineras, de tal

	manera que va aumentando el número de centros en 10 unidades por iteración, mientras que el número de gasolineras aumenta en 100 por iteración. ● Recogemos, para cada réplica, los resultados de tiempo de ejecución, calidad de solución final y número de nodos expandidos. ● Analizamos los resultados y comparamos.

Tras la ejecución del experimento, obtenemos los siguientes resultados para las variables analizadas:

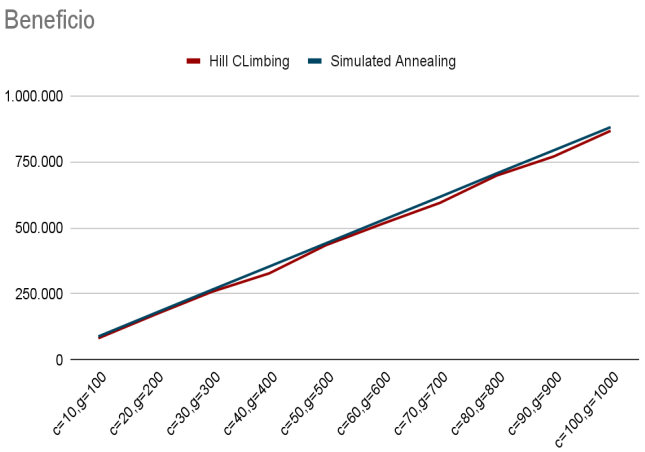
Sea c el número de centros del problema, y g el número de gasolineras:

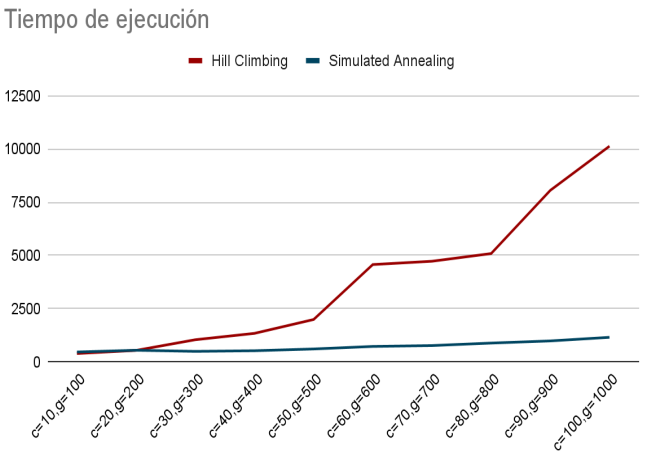
Hill Climbing			
réplica	tiempo ejecución (ms)	beneficio	nodos expandidos
c=10,g=100	395	81.892	7
c=20,g=200	542	171.712	5
c=30,g=300	1043	258.268	5
c=40,g=400	1342	327.436	4
c=50,g=500	1993	434.556	4
c=60,g=600	4574	516.144	7
c=70,g=700	4729	594.556	4
c=80,g=800	5090	698.272	5
c=90,g=900	8059	770.288	7
c=100,g=1000	10.133	867.560	7
Valores medios:	**3790**	**472.068**	**5,5**

Simulated Annealing			
réplica	tiempo ejecución (ms)	beneficio	nodos expandidos
c=10,g=100	468	88.536	150.001
c=20,g=200	548	177.408	150.001

c=30,g=300	497	265.776	150.001
c=40,g=400	526	353.288	150.001
c=50,g=500	609	441.536	150.001
c=60,g=600	728	529.620	150.001
c=70,g=700	769	617.212	150.001
c=80,g=800	886	706.376	150.001
c=90,g=900	987	793.456	150.001
c=100,g=1000	1156	881.292	150.001
Valores medios:	**717,4**	**485.450**	150.001

En este caso haremos dos gráficos, en el primero compararemos el tiempo de ejecución con el número de centros/gasolineras, y en el segundo compararemos el beneficio con el número de centros/gasolineras, tal como se muestra:





Como se puede observar fácilmente a partir de los gráficos, utilizando el algoritmo de *Simulated Annealing* obtenemos un beneficio prácticamente idéntico al que nos da al utilizar el *Hill Climbing* pero sin comprometer el tiempo de ejecución. Utilizando *Hill Climbing*, el tiempo de ejecución al aumentar el número de gasolineras y centros crece rápidamente mientras que al usar el segundo algoritmo, vemos como prácticamente este valor no se ve alterado.

Finalmente, podemos concluir que, como el tiempo de ejecución varía en función del número de centros y del número de gasolineras, de tal forma que crece conforme estos dos parámetros crecen también (dependencia lineal), tenemos que rechazar la hipótesis nula y aceptar la alternativa.

5. Hemos asumido que tener centros de distribución no tiene ningún coste y que tenemos un camión por centro. Usad el algoritmo de Hill Climbing para experimentar qué pasaría si mantenemos el número de camiones, pero reducimos el número de centros a la mitad, es decir un escenario de 5 centros, 10 camiones y 100 gasolineras. ¿Cómo afecta este cambio al beneficio y a los kilómetros recorridos?

Hemos comprobado que el número de centros, multiplicado por el número de cisternas por centro, nos aporta, a priori, un mayor beneficio, pues se pueden aceptar un mayor número de peticiones. Este beneficio puede convertirse en pérdidas, puesto que al beneficio de rellenar un depósito de una gasolinera hemos de sustraer el coste en gasolina que supone ir a la gasolinera y volver de ella.

En este experimento, definimos el escenario para el primer experimento, pero reducimos el número de centros a la mitad, respetando el número de gasolineras, la multiplicidad de los centros, la semilla generadora y los parámetros de Simulated Annealing.

Definimos formalmente el experimento de la siguiente manera:

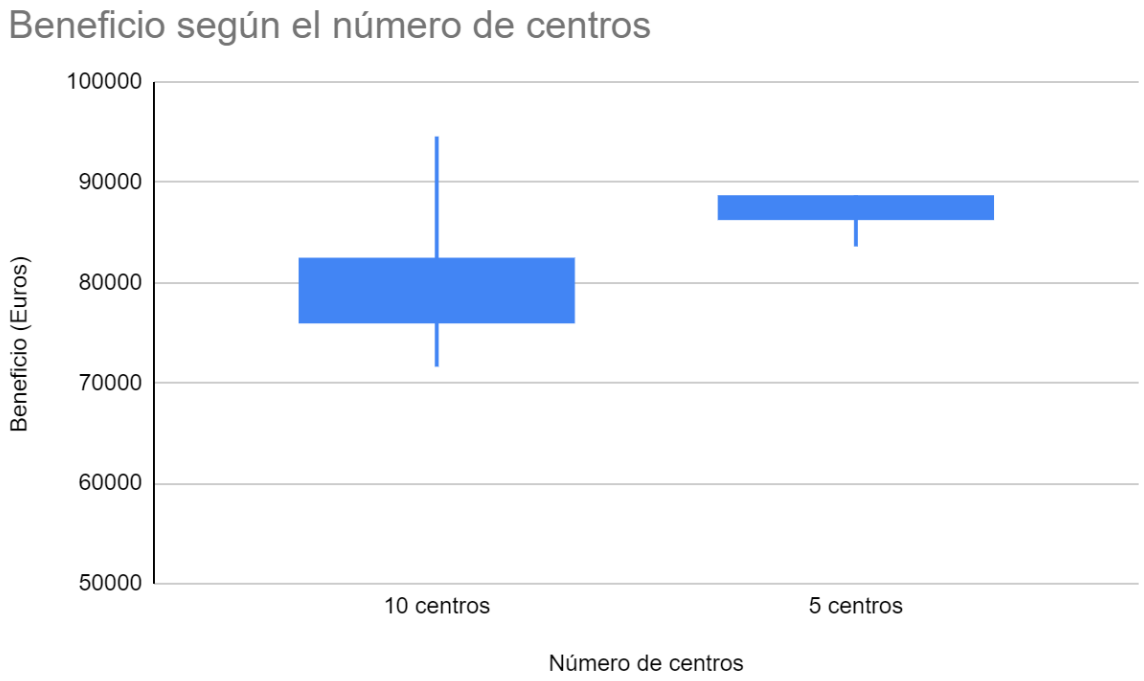
Observación	¿Varía el beneficio al dividir el número de centros, que poseíamos inicialmente, a la mitad?
Planteamiento	A partir de 5 centros de distribución, 100 gasolineras y los mismos parámetros que en el experimento inicial, queremos ver si hay variación en el resultado con los resultados del experimento 1.
Hipótesis	Modificar el número de centros para los valores del experimento 1 no varía en el beneficio obtenido (H0), o existen desigualdades en el beneficio obtenido reduciendo el número de centros a la mitad (H1).
Variable/s independiente/s	Número de centros.
Variable/s dependiente/s	Tiempo de ejecución. Adicionalmente, y aunque el experimento no lo requiera, la calidad de la solución y el número de nodos expandidos.
Variable/s controlada/s	Número de gasolineras, multiplicidad de los centros, semilla del problema, algoritmo de Hill Climbing.
Método	• Partimos de una semilla aleatoria. Definimos el número de centros a 5, la mitad que en el experimento 1, y el número de gasolineras a 100. También definimos que el algoritmo a usar es Hill Climbing. La multiplicidad es de 1 y la semilla es un número entero

	aleatorio fijo para toda réplica del experimento (semilla 9631). La solución inicial se genera de forma aleatoria. ● Ejecutamos el algoritmo 10 veces. ● Recogemos, para cada réplica, los resultados de tiempo de ejecución. Como adición, recogemos también la calidad de solución final y número de nodos expandidos. ● Analizamos los resultados y comparamos.

Tras la ejecución del experimento, obtenemos los siguientes resultados para las variables analizadas:

Experimento 1 con número de centros = 5			
réplica	tiempo ejecución (ms)	beneficio	nodos expandidos
1	204	86172	7
2	94	86476	6
3	138	88272	8
4	132	88548	8
5	136	86396	7
6	124	88676	8
7	126	88584	7
8	57	83600	5
9	119	86888	6
10	129	88380	7
Valores medios:	**125,9**	**87199,2**	**6,9**

Para ilustrar los resultados de forma visualmente esclarecedora, veamos los resultados en gráficos y comparemos los valores esperados de la muestra, así como su desviación muestral:



Finalmente, podemos concluir que el beneficio de la solución final sí varía al reducir el número de centros a la mitad, aumentando el número de cisternas por centro. Vemos que es más eficiente, y aporta mayor beneficio, tener 2 cisternas por centro con 5 centros que tener 10 centros con 1 cisterna por centro, pues la esperanza del beneficio crece y los valores no se encuentran tan dispersos en el gráfico, con lo que podremos estimar un beneficio aproximado con mayor precisión.

Así pues, podemos rechazar la hipótesis nula y aceptar la hipótesis alternativa.

6. Hemos asumido un coste fijo de 2 por kilómetro recorrido. Usad el algoritmo de Hill Climbing para estudiar cómo afecta al número de peticiones servidas el aumentar el coste por kilómetro usando el escenario del primer apartado. Id doblando el coste hasta que podáis ver la tendencia. ¿Tiene algún efecto en la proporción de peticiones servidas según el tiempo que llevan pendientes?

En este experimento, utilizaremos los parámetros del problema del experimento 1 (10 centros, 100 gasolineras, semilla aleatoria fija para toda réplica y multiplicidad 1 para todo centro de distribución). A partir de aumentar el valor del coste por kilómetro recorrido, observaremos cuál es la tendencia del beneficio para determinados valores del coste por kilómetro.

También observaremos qué peticiones se realizan, e intentaremos obtener un criterio común para generalizar las condiciones bajo las cuales se seleccionan qué peticiones para qué valores de coste por kilómetro recorrido.

Definimos formalmente el experimento de la siguiente manera:

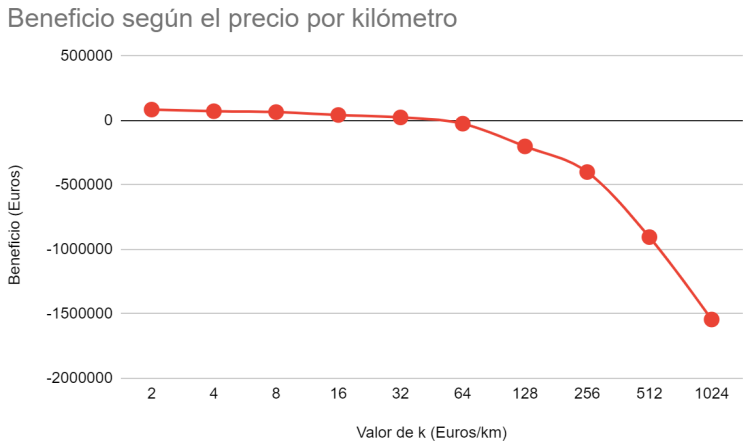
Observación	¿Varía el beneficio, y el tipo de peticiones asignadas, al aumentar el coste por kilómetro de las cisternas?
Planteamiento	A partir de los mismos parámetros que en el experimento inicial, modificando el precio por kilómetro de las cisternas, queremos ver si hay variación en el beneficio de la solución final, así como en el tipo de peticiones que se recogen.
Hipótesis	Modificar el precio por kilómetro no modifica ni el beneficio ni el tipo de peticiones asignadas (H0), o existe diferencia en el criterio de selección de peticiones y en el beneficio de la solución final en función del coste por kilómetro (H1).
Variable/s independiente/s	Precio por kilómetro recorrido por una cisterna.
Variable/s dependiente/s	Tiempo de ejecución, calidad de la solución , tipo de peticiones asignadas.
Variable/s controlada/s	Número de centros, número de gasolineras, multiplicidad de los centros, semilla del problema, algoritmo de Hill Climbing.
Método	• Definimos el número de centros a 10 y el número de gasolineras a 100. También definimos que el algoritmo a usar es Hill Climbing. La multiplicidad es de 1 y la semilla es un número entero aleatorio fijo para toda réplica del experimento. La solución inicial se genera de forma aleatoria. • Definimos un coste por kilómetro k_i, que sigue la recurrencia $k_i = 2*k_{i-1}$, siendo $k_0 = 2$. • Ejecutamos el algoritmo n = 10 veces, donde i pertenece al intervalo [0,9]. • Recogemos, para cada iteración del experimento, el beneficio de la solución final, y analizamos cuántos días tienen las peticiones de cada solución final (como simplificación, cogeremos el valor máximo de los días que llevan pendientes las peticiones asignadas al final del algoritmo). • Analizamos los resultados y comparamos.

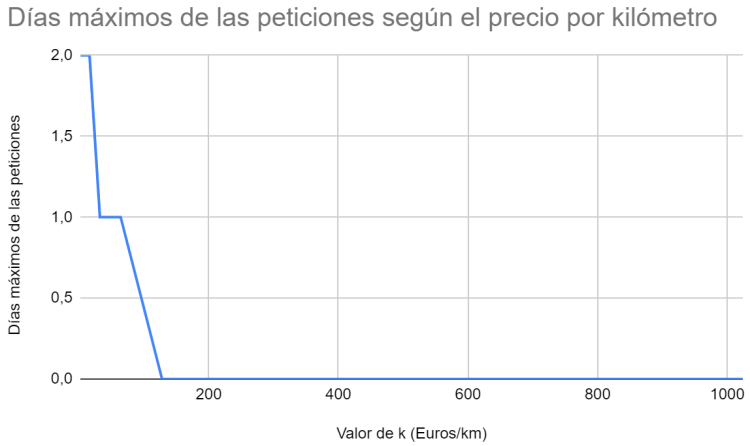
Tras la ejecución del experimento, obtenemos los siguientes resultados para las variables analizadas:

Coste por kilómetro

iteración	beneficio	días máximos de las peticiones
k = 2	85.024	2
k = 4	72.236	2
k = 8	65.800	2
k = 16	43.012	2
k = 32	24.200	1
k = 64	-24.304	1
k = 128	-200.532	0
k = 256	-399.552	0
k = 512	-904.212	0
k = 1024	-1.543.532	0
Valores medios:	**-278.186**	**1,0**

Para ilustrar los resultados de forma visualmente esclarecedora, veamos los resultados en gráficos y comparemos los valores esperados de la muestra, así como su desviación muestral:





Finalmente, podemos concluir que el coste por kilómetro de las cisternas sí varía el beneficio final de las soluciones, así como los días que llevan las peticiones sin asignar. Esto se debe a que ir a realizar una petición puede ser más costoso en pérdidas por viaje que en el beneficio de suplir la demanda. Observamos también que los días máximos de las peticiones decrecen rápidamente en cuanto crece el precio por kilómetro, así como también decrece el beneficio. Podemos observar que, para estos parámetros del problema en concreto, existe un valor de k entre 32 y 64 en que el beneficio de suplir la demanda menos las pérdidas de los viajes tienen un beneficio de 0 euros, y por tanto supone el límite entre beneficio y pérdida.

Podemos refutar la hipótesis nula y aceptar la hipótesis alternativa.

7. Manteniendo el límite de 5 viajes diarios por cisterna, comprobad qué efecto tendría el aumentar y disminuir en una hora el horario de trabajo de las cisternas con el escenario del primer apartado, es decir, aumentando/disminuyendo los kilómetros que podemos recorrer en un día. Usad el algoritmo de Hill Climbing para comprobar el efecto sobre el beneficio.

A partir de los parámetros iniciales (10 centros, 100 gasolineras, semilla aleatoria fija para todo el experimento, coste por kilómetro de 2 y Hill Climbing) vamos a alterar el valor de los kilómetros máximos permitidos por cisterna para ver cómo varía el beneficio, y como están estrictamente relacionados, el número de peticiones asignadas también.

Definimos formalmente el experimento de la siguiente manera:

Observación	¿Varía el beneficio si modificamos el número de kilómetros máximo que pueden hacer las cisternas?
Planteamiento	A partir de los mismos parámetros que en el experimento inicial, modificamos el número de kilómetros máximo que se pueden hacer por cada iteración del experimento, y observamos si existe una dependencia entre el beneficio y los

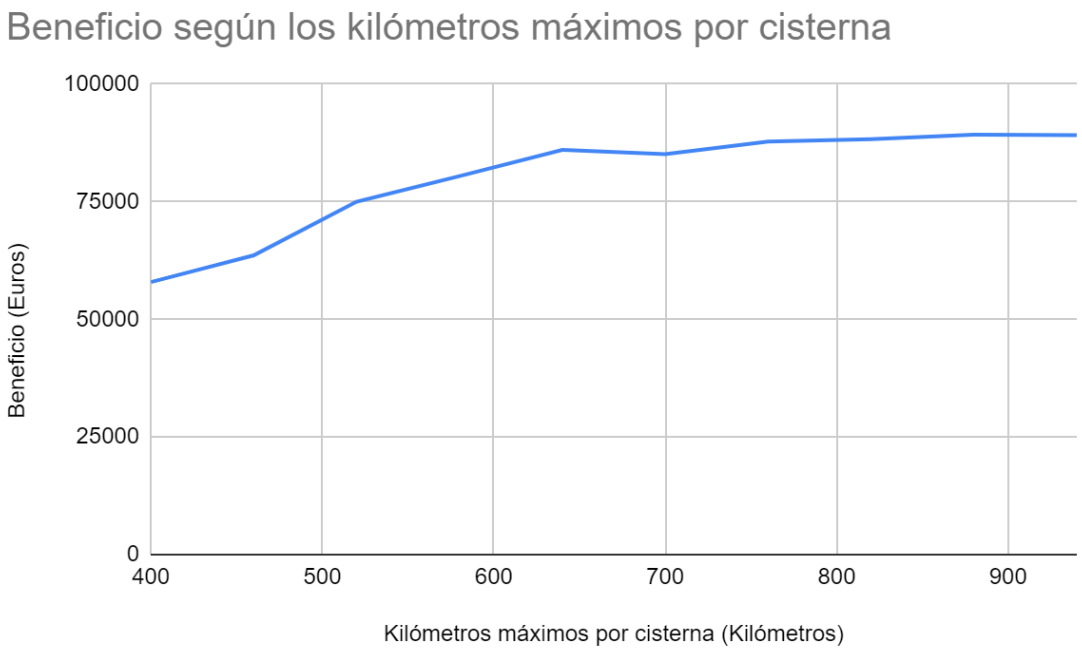
	kilómetros máximos por cisterna.
Hipótesis	Modificar los kilómetros máximos por cisterna no modifica el beneficio (H0), o existe diferencia en el beneficio de la solución final en función de los kilómetros máximos que se pueden hacer (H1).
Variable/s independiente/s	Kilómetros máximos que se pueden realizar.
Variable/s dependiente/s	Tiempo de ejecución, calidad de la solución , tipo de peticiones asignadas.
Variable/s controlada/s	Número de centros, número de gasolineras, multiplicidad de los centros, precio por kilómetro, semilla del problema, algoritmo de Hill Climbing.
Método	• Partimos de una semilla aleatoria. Definimos el número de centros a 10 y el número de gasolineras a 100. También definimos que el algoritmo a usar es Hill Climbing. La multiplicidad es de 1 y la semilla es un número entero aleatorio fijo para toda réplica del experimento (semilla 4705). La solución inicial se genera de forma aleatoria. • Ejecutamos 10 réplicas del algoritmo con unos kilómetros máximos por cisterna determinados, de tal manera que empezamos con menor kilometraje, hasta alcanzar un kilometraje alto por cisterna. • Ejecutamos el algoritmo y recogemos, para cada iteración del algoritmo, el beneficio de la solución final. • Analizamos y comparamos los resultados.

Sea maxK el número máximo de kilómetros que puede hacer un camión cisterna en un día. Tras la ejecución del experimento, obtenemos los siguientes resultados para las variables analizadas:

Kilometraje por cisterna variable		
iteración	beneficio	tiempo de ejecución
maxK = 400km	57.936	42
maxK = 460km	63.608	38
maxK = 520km	75.012	44
maxK = 580km	80.444	33

maxK = 640km	86.004	54
maxK = 700km	85.116	29
maxK = 760km	87.776	38
maxK = 820km	88.292	23
maxK = 880km	89.260	34
maxK = 940km	89.136	48
Valores medios:	**82.564**	**38,3**

Para ilustrar los resultados de forma visualmente esclarecedora, veamos los resultados en gráficos y comparemos los valores esperados de la muestra, así como su desviación muestral:



Finalmente, podemos concluir que el kilometraje máximo de cada cisterna sí influye en el beneficio de la solución, de tal manera que cuando este límite crece, el beneficio crece también. Por contraparte, también observamos que, si el límite alcanza un cierto valor elevado, puede llegar a estancarse el beneficio y dejar de crecer, conforme crece el límite de kilómetros. Esto es debido a que a partir de un cierto valor para esta variable, todas las peticiones, para un problema en concreto, serán satisfechas por algún camión cisterna, dejando vacío el conjunto de peticiones sin asignar, por lo que no podremos asignar ninguna petición más y, por tanto, el beneficio no crecerá en consecuencia.
Así pues, podemos rechazar la hipótesis nula y aceptar la hipótesis alternativa.

5 Conclusiones

Como resumen de la práctica de búsqueda local, hemos partido de un problema en el que teníamos que encontrar una manera de buscar una solución cercana a la óptima, aplicando técnicas de búsqueda en un espacio de soluciones (Hill Climbing y Simulated Annealing), donde se va mejorando la calidad de esta.

A través de la descripción del problema, hemos creado un modelado de este para ser representado a través de estructuras en Java. Hemos creado operadores para la transformación de los estados, así como una forma de calcular la calidad de la solución, es decir el beneficio que nos aportaba hacer cierta planificación de las peticiones para un problema con unos datos concretos. Hemos creado dos algoritmos generadores de solución inicial: Un algoritmo aleatorio y uno voraz, así como dos funciones generadoras de sucesores, una para cada algoritmo de búsqueda local.

Con el modelado, los operadores, la función de calidad y las demás operaciones que implementa Hill Climbing y Simulated Annealing, hemos construído una forma de resolver nuestro problema de optimización combinatoria.

El siguiente paso ha sido experimentar. A través de diferentes experimentos, hemos encontrado que: El operador que genera mayor beneficio es el swap entre centros y gasolineras; el generador de solución inicial que mejor beneficio aporta es el algoritmo voraz, aunque reduce el número de nodos expandidos a prácticamente 1 o 2; Los parámetros de Simulated Annealing influyen en la eficiencia del algoritmo de forma considerable, por lo que es necesario experimentar con ellos para encontrar aquellos que funcionan mejor con el problema; y que, finalmente, el número de centros, multiplicidad de los centros y el número de gasolineras, así como el precio por kilómetro y los kilómetros máximos permitidos por cisterna, alteran los resultados del algoritmo, modificando así el beneficio del itinerario propuesto por este.

Por lo tanto, vemos que la experimentación es crucial para determinar una mejor solución del problema, y que por muy cerca que esté de la solución óptima, por características de la búsqueda local, casi nunca se podrá llegar al óptimo, y si se llega es porque han coincidido una buena solución inicial, con un buen heurístico y una buena secuencia de nodos expandidos que nos han llevado a esa solución óptima.

6 Trabajo de innovación

6.1 Descripción

AI Dungeon es un videojuego de aventura gratis basado en texto, que usa inteligencia artificial para generar contenido ilimitado. También permite a los jugadores crear y compartir sus propios escenarios.

6.2 Reparto

- **Pau:** Descripción del producto o servicio
- **Pau:** Descripción de las técnicas de IA que se han utilizado
- **Andrés:** Descripción de cómo han sido utilizadas las técnicas
- **Guillem:** Explicación de porqué es un producto/servicio innovador y la naturaleza de la innovación (innovación en la técnica/método de IA, uso innovador de técnicas ya existentes)
- **Andrés:** Impacto del producto en la empresa (beneficios, riesgos, posición en el mercado)
- **Guillem:** Impacto del producto en el usuario o en la sociedad (beneficios y riesgos)

6.3 Referencias

[1] "Ai Dungeon." AI Dungeon, Latitude,5 Dec. 2019, Visitada: 29.09.2021 https://play.aidungeon.io/main/home.

[2] "WIKI Ai dungeon." Wikipedia, Wikimedia Foundation, 30 Dec. 2020, Visitada: 29.09.2021

https://es.wikipedia.org/wiki/AI_Dungeon.

[3] "Cómo Ai Dungeon Convierte La Ia En Un Juego." Gaming Online, Gaming Online, Visitada: 4.10.2021

https://gamingonline.es/blog/como-ai-dungeon-2-convierte-la-ia-en-un-juego/

6.4 Dificultades

De momento no hemos tenido grandes dificultades, ya que no hemos avanzado hasta el punto que nos gustaría, pero aún así vemos que una dificultad puede ser la falta de tiempo para el trabajo, además de que no hay tanta información como esperábamos encontrar a priori, aunque seguramente cuando hagamos una búsqueda exhaustiva sobre el tema podremos encontrar todo lo necesario.