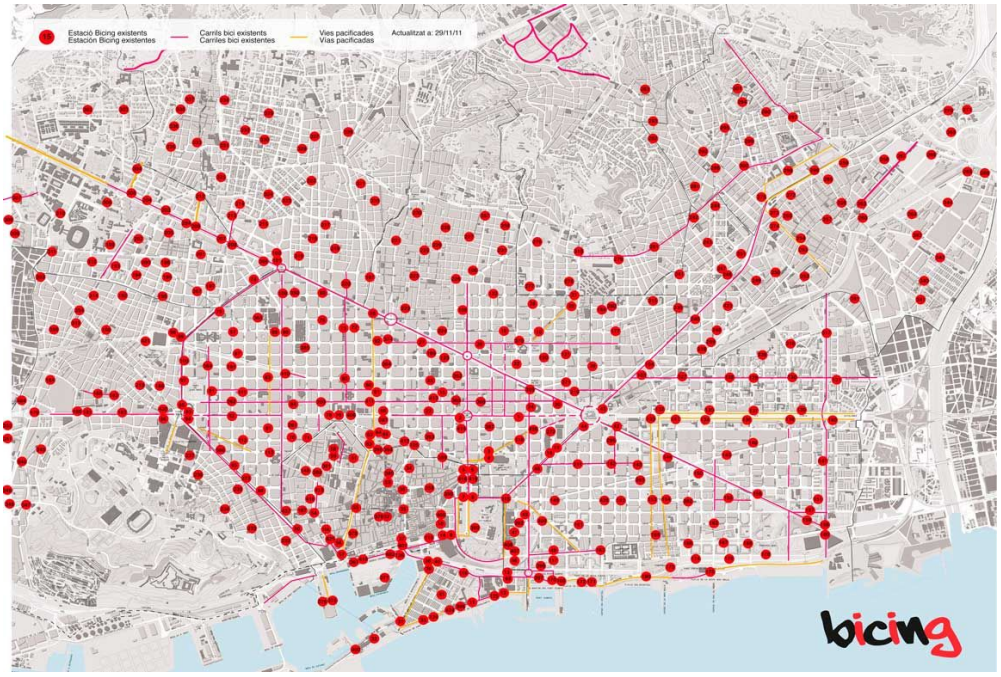
Inteligencia Artificial

Práctica 1: Búsqueda Local Bicing



Isaac Muñoz, Ignasi Sant, Pablo Fonoll

Índice

1.Identificación del problema

1.1.Descripción del problema

El problema consiste en optimizar la distribución de bicicletas del Bicing en las diferentes estaciones de manera que esta cumpla con los criterios de demanda de los datos que disponemos, es decir, que los usuarios puedan encontrar bicicletas en las estaciones cuando lo necesiten.

Bicing nos proporciona diferentes datos de las estaciones como: la previsión de bicis no usadas en una estación y hora, previsión del número de bicis por estación en la hora siguiente a la actual (sin contar traslados) y la previsión de la demanda en una estación en la hora siguiente a la actual. Las estaciones se encuentran en los cruces de las calles y tienen una capacidad ilimitada. La demanda en las estaciones puede ser equilibrada (parecida en todas las estaciones) o en hora punta (algunas tienen más demanda que otras).

Nosotros nos hemos de encargar del traslado de bicicletas con la ayuda de F furgonetas, con capacidad de 30 bicis, que pueden recoger una vez por hora bicis en una estación origen y dejarlas en una o dos estaciones destino. De estas furgonetas no puede haber dos con misma estación origen. Tampoco es necesario usar las F furgonetas en la solución.

Bicing nos paga 1 euro para cada bici que transportamos que haga que una estación se aproxime a su demanda en la siguiente hora, vemos por lo tanto que tendremos que tener en cuenta la previsión de bicis que habrá en la siguiente hora ya que si nos alejamos de la demanda no nos pagarán ya que la bici se ha transportado inutilmente. También nos cobraràn 1 euro si con el transporte hacemos que una estación se quede por debajo de la demanda prevista. A parte el coste del transporte en euros por kilómetro recorrido es ((nb + 9) div 10) donde nb es el número de bicicletas. La distancia entre dos estaciones se calcula $d(i,j) = |i_x - j_x| + |i_y - j_y|$.

El origen de las furgonetas lo podemos decidir teniendo en cuenta la optimización de la solución. Tendremos que definir también el/los destino/s de las furgonetas y cuantas bicis dejan en cada destino.

En la solución se busca el màximo el beneficio por traslados de bicis y el mínimo coste del transporte.

1.2.Características del problema

Se trata de un problema de optimización donde intervienen una gran cantidad de elementos, por lo tanto tendremos que buscar una buena representación del estado para que no ocupemos toda la memoria. A partir de aquí hemos de identificar los elementos clave para solucionar el problema. Fácilmente vemos que lo que condicionarà claramente nuestro sistema de transporte serà la demanda y previsión de bicicletas de una estación en la hora siguiente juntamente con el número de bicis a transportar y la distancia de transporte. Por lo tanto, tendremos que coger bicis de las estaciones donde exceden a la demanda de la hora actual y tendremos que colocarlas en las estaciones donde faltarán bicis para cumplir la demanda.

1.3.Elementos del estado

El estado representa un mapa de la ciudad con sus estaciones colocadas aleatoriamente por los distintos puntos posibles. Cada estación tendrá un par de coordenadas XY y un número de bicicletas cuyo límite no está definido.

Disponemos de algunas variables para ayudarnos a guiarnos por el algoritmo:

- int solucio[] - vector que indica el número de bicicletas que faltaran para cubrir la demanda en cada estación teniendo en cuenta la previsión de demanda que habrá en la hora siguiente. Si no hay demanda el valor será 0. Esta variable se mantiene siempre actualizada, permitiendo así el cálculo del beneficio, reflejada en la variable benefici.
- int benefici: beneficio total no teniendo en cuenta costes. Esta variable se calcula con la diferencia del sumatorio de los valores de solucio[] antes de mover las furgonetas y después de moverlas.
- int CostTotal: en esta variable se guardan los costes de transporte de las furgonetas (en total).
- Furgoneta flota[]: vector de objetos Furgoneta, estas constan de una estación origen y destino.
- Estaciones est: ArrayList de estaciones.

1.4.¿Por qué búsqueda local?

Al ver cómo está planteada la pràctica y las características que han de poseer las soluciones nos damos cuenta de que no hay posibilidad de hallar una solución óptima debido a que el tamaño del espacio de búsqueda es demasiado grande , y en este caso nos conformamos con una solución que podamos considerar suficientemente buena. Por este motivo, creemos que el uso de algoritmos de búsqueda local es ideal para resolver el problema.

2.Estado del problema y representación

2.1.Representación del problema

La representación del problema es la siguiente:
Como punto de partida tenemos un tablero con un número determinado de estaciones en las que se asignará un número determinado de bicicletas y un número determinado de furgonetas. La posición de las furgonetas y estaciones se asignará de manera arbitraria según el estado inicial que generemos.
A partir de esta situación inicial, nuestro objetivo será alcanzar el mayor beneficio posible, es decir, distribuir las bicicletas de manera que la diferencia entre lo ganado por satisfacer la demanda prevista y el coste que comporta sea la mayor posible.
Para alcanzar este objetivo, disponemos de herramientas para manipular los elementos del problema. Estas consistirán en las redistribuciones, mediante los recorridos de las furgonetas, que hacemos de las bicis. Para distribuir de manera eficiente, serán relevantes elementos como la demanda prevista y la localización de las estaciones y furgonetas. Una buena redistribución nos generará una mejor solución.
Esta solución podrá ser calificada como tal si se cumplen la siguientes restricciones: las furgonetas no pueden visitar más de dos estaciones, las furgonetas cargan bicicletas solo en la estación de origen (distinta entre todas las furgonetas) y el beneficio que obtenemos tiene que ser positivo. Además, para evaluar la calidad de una solución nos podremos fijar en dos criterios: maximización de lo que obtenemos por los traslados de las bicicletas y minimización de los costes de transporte de las bicicletas.

2.2.Análisis del tamaño del espacio de búsqueda

El espacio de búsqueda en nuestro problema corresponderá al espacio de soluciones que estará formado por todas las soluciones posibles. Este conjunto de soluciones posibles engloba todas las combinaciones posibles de distribuciones de bicis excepto las que no cumplen las siguientes restricciones: las furgonetas no pueden visitar más de dos estaciones, las furgonetas cargan bicicletas solo en la estación de origen (que es distinta para todas) y el beneficio que obtenemos tiene que ser positivo.
Vemos entonces que el espacio de búsqueda será inmenso y, por lo tanto, nos decantamos por utilizar algoritmos de búsqueda local en detrimento de los algoritmos de búsqueda no informada y búsqueda heurística.

3.Representación y análisis de los operadores

Los operadores usados son:

1-setPos(pos):

Condición de aplicabilidad: Tenemos que asegurarnos que la estación en la posición "pos" no ha sido asignada como la estación origen de ninguna furgoneta.

Efecto: Asigna la estación que hay en la posición "pos" como estación origen de la furgoneta sobre la que se aplica el método.

2-setDest(dest):

Condición de aplicabilidad: Tenemos que asegurarnos que la estación en la posición "pos" no ha sido visitada por una furgoneta todavía.

Efecto: Asigna la estación que hay en la posición "pos" como estación destino de la furgoneta sobre la que se aplica el método.

3-NoMou():

Condición de aplicabilidad: Ninguna

Efecto: La furgoneta sobre la que se aplica este método no hará ningún recorrido.

Hemos escogido únicamente estos tres operadores porque con ellos somos capaces de recorrer todo el espacio de soluciones.

4.Análisis de la Función Heurística

Como función heurística que guiará el problema hacia una solución suficientemente buena dentro del espacio de soluciones, tanto si aplicamos Simulated Annealing como Hill Climbing, hemos elegido:

1-BicingHeuristicFunction: Esta función calculará y tomará com heurístico el beneficio para cada estado solución. Este beneficio lo retornaremos cambiado de signo ya que la función que crea sucesores interpreta los valores altos como peores. Por tanto, si tenemos un beneficio de 100 y uno de 50 y lo retornamos como -100 y -50 respectivamente, se considerará el estado con valor de retorno -100 como mejor. Hemos escogido esta función como la heurística ya que la búsqueda de un mayor beneficio nos hará ir explorando soluciones cada vez más óptimas en términos de ganancias económicas y será mucho más rápido encontrar una solución suficientemente buena que si aplicamos la búsqueda de manera ciega.

2-BicingHeuristicFunction2: Esta función calculará y tomará como heurístico los ingresos para cada solución. El valor del ingreso obtenido lo retornaremos cambiado de signo como hemos hecho en la primera función. La gran diferencia entre las dos funciones radica en que en esta segunda función quizás exploremos antes soluciones con beneficios mínimos solo porque el ingreso ha sido muy superior que el de los estados que obtienen un buen beneficio.

5.Elección y generación del estado inicial

Hemos usado dos métodos diferentes para generar un estado inicial:

El primero de forma secuencial, en que movemos todas las furgonetas, donde la primera furgoneta tendrá la estación 0 como estación origen y la estación 1 como estación final, la segunda furgoneta tendrá la estación 2 como estación origen y la estación 3 como estación final y así sucesivamente. Si todas las estaciones son origen o destino de alguna furgoneta y hay furgonetas que no tienen asignadas ninguna estación, estas no intervienen en este estado.

El segundo genera el estado de forma aleatoria, es decir asigna aleatoriamente el origen y el destino de las furgonetas teniendo en cuenta las restricciones que se han explicado en los apartados anteriores.

Hemos escogido la generación de estado aleatoria ya que nos daba mejores resultados.

También pensamos en hacer una solución inicial Greedy, pero creemos que con una solución inicial tan buena, la mejoría de los resultados de la solución final respecto la solución inicial no se apreciaría tanto y nos costaría más sacar conclusiones. Pese a esto, creemos que con Greedy hubiésemos obtenido mejores resultados.

6.Experimentos

6.1.Determinar qué conjunto de operadores da mejores resultados para una función heurística que optimice el primer criterio (el transporte es gratis) con un escenario en el que el número estaciones es 25, el número total de bicicletas es 1250, el número de furgonetas 5 y la demanda es equilibrada. 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.

Observación: Puede haber conjuntos de operadores mejores que otros.

Planteamiento: Escogemos diferentes conjuntos de operadores y observamos sus resultados.

Hipótesis: Todos los conjuntos de operadores dan el mismo resultado (H0) o los hay que producen resultados diferentes.

Método: Usamos la generación aleatoria para la solución inicial. A partir de allí ejecutamos el algoritmo con los distintos operadores y comparamos resultados. Podemos ver claramente que con los operadores SetPos(), SetDest() y NoMou() obtenemos unos resultados mejores.

6.2.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.

Observación: Puede haber soluciones iniciales que están más cerca de las soluciones finales que otras y que hagan trabajar menos al Hill Climbing.

Planteamiento: Escogemos diferentes soluciones iniciales y observamos sus resultados

Hipótesis: Las diferentes soluciones iniciales dan resultados distintos (H0) o las hay que producen el mismo.

Método: Generamos dos soluciones iniciales, una aleatoria y otra secuencial (explicadas anteriormente). Ejecutamos la práctica partiendo de una solución o la otra y comparamos resultados.

Solución inicial aleatoria-> resultado = 53, tiempo = 20.

Solución inicial secuencial-> resultado = 16 , tiempo = 18.

Claramente podemos apreciar que la solución inicial aleatoria nos proporciona un resultado mucho mejor por la diferencia de tiempo que comporta respecto a la otra solución inicial. Por lo tanto a partir de ahora usaremos la aleatoriedad para generar la solución inicial.

6.3.Determinar los parámetros que dan mejor resultado para el Simulated Annealing con el mismo escenario, usando la misma función heurística y los operadores y la

estrategia de generación de la solución inicial escogidos en los experimentos anteriores.

Observación: La calidad de la solución puede variar según qué valores cojan las variables iteraciones, stiter, K y lambda.

Planteamiento: Asignamos diferentes valores a los parámetros (interacciones, stiter, K y lambda) y observamos sus resultados.

Hipótesis: Todas las soluciones obtenidas serán diferentes para los diferentes valores asignados a los parámetros (H0) o las soluciones obtenidas serán iguales.

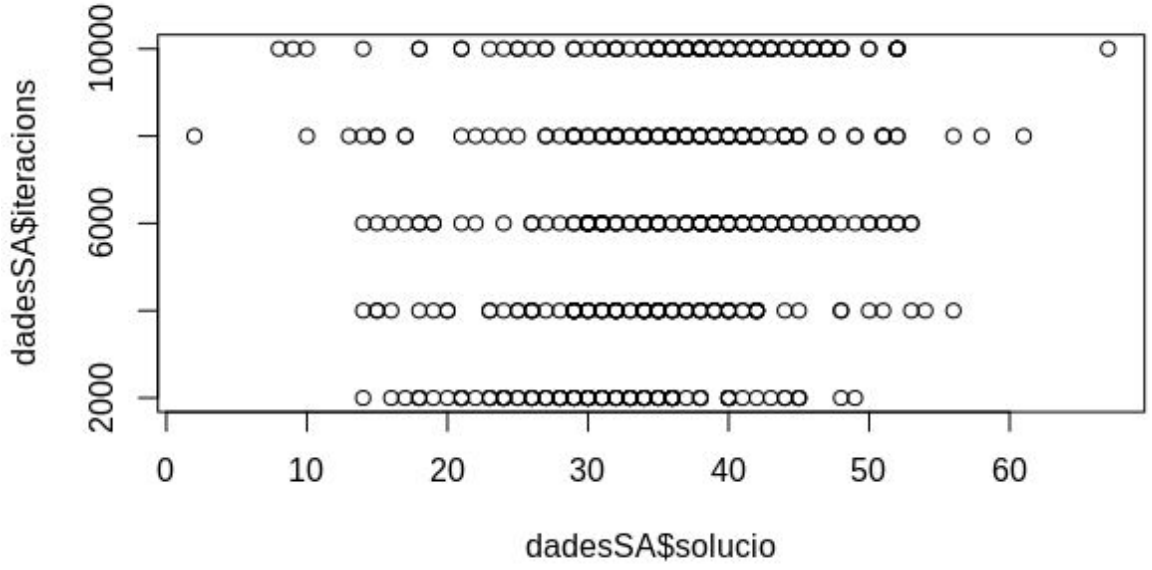
Método: Ejecutamos la práctica usando Simmulated Annealing, con el estado inicial aleatorio, con los operadores setPos(pos) y setDest(dest) y con la función heurística 1 que tiene en cuenta los beneficios. A partir de aquí, llamo varias veces la función SimmulatedAnnealingSearch() y le paso los parámetros con diferentes valores cada vez. El objetivo es que ejecute la función con todas las combinaciones posibles de parámetros para ver si podemos detectar algún patrón y sacar conclusiones. Los valores que pueden tener las variables son los siguientes:

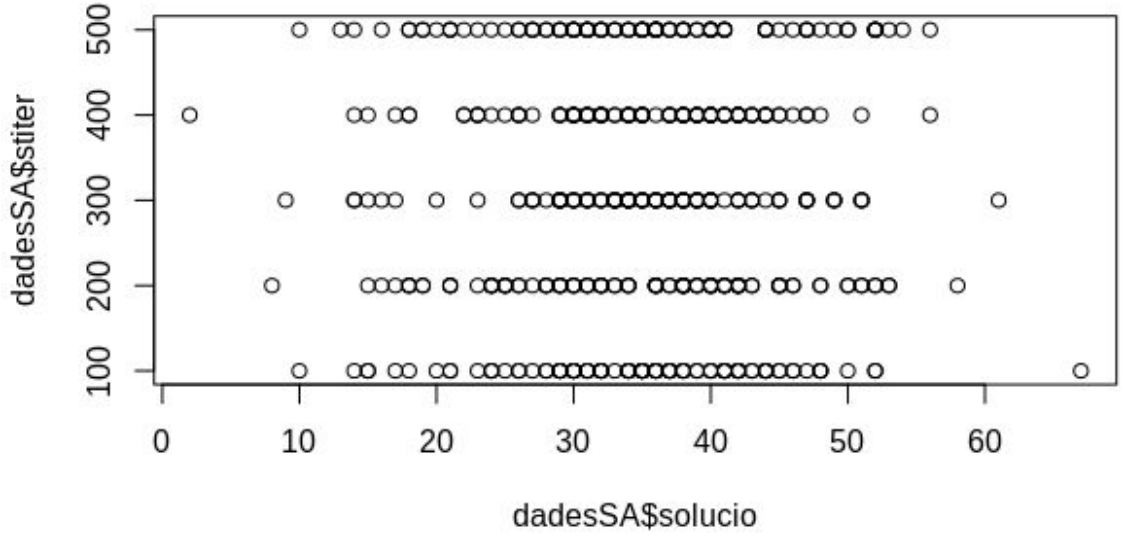
- interacciones [2000, 4000, 6000, 8000, 1000]
- stiter [100, 200, 300, 400, 500]
- k [1, 5, 25, 125]
- lambda [1, 0.1, 0.01, 0.001, 0.0001]

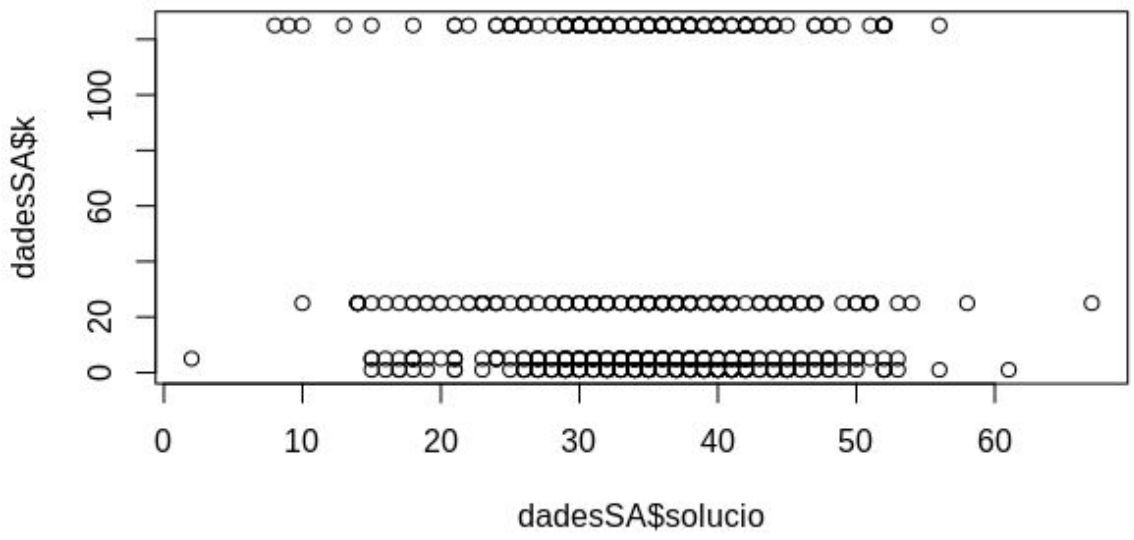
Los resultados del experimento los podemos ver en el documento “dadesSA.txt” y en el documento “dadesSA_ordenatCreixentment.txt” si lo queremos ver en orden ascendente por beneficio.

Podemos ver que los con los parámetros iteraciones = 10000, stiter = 100 , k = 25 y lambda = 0.1 obtenemos el beneficio más alto de 67. Es interesante ver los datos ordenados por resultado ya que vemos qué patrones siguen las mejores soluciones.

Si nos fijamos en las iteraciones, vemos que un número alto de ellas permiten llegar a una mejor solución. En el gráfico generado por R se puede apreciar este hecho. Por otra parte el stiter no parece afectar demasiado ya que no se puede apreciar fácilmente ningún patrón tal y como indica la gráfica de R. Si nos fijamos en la K tampoco observamos ningún patrón razonable. Por último podemos ver que no hay ni una de las mejores soluciones que tenga lambda = 1 , de hecho las casi todas las peores soluciones tienen lambda = 1 lo que nos lleva a concluir que no es un valor muy adecuado. En las mejores soluciones observamos valores de lambda intermedios (entre 0.1 y 0.01 básicamente) por lo que podemos concluir que los valores extremos tampoco son buenos.


dadesSA$iteracions
10000
6000
2000
0
10
20
30
40
50
60
dadesSA$solucio


dadesSA$stiter
500
400
300
200
100
0
10
20
30
40
50
60
dadesSA$solucio


dadesSA$k
0
20
60
100
0
10
20
30
40
50
60
dadesSA$solucio

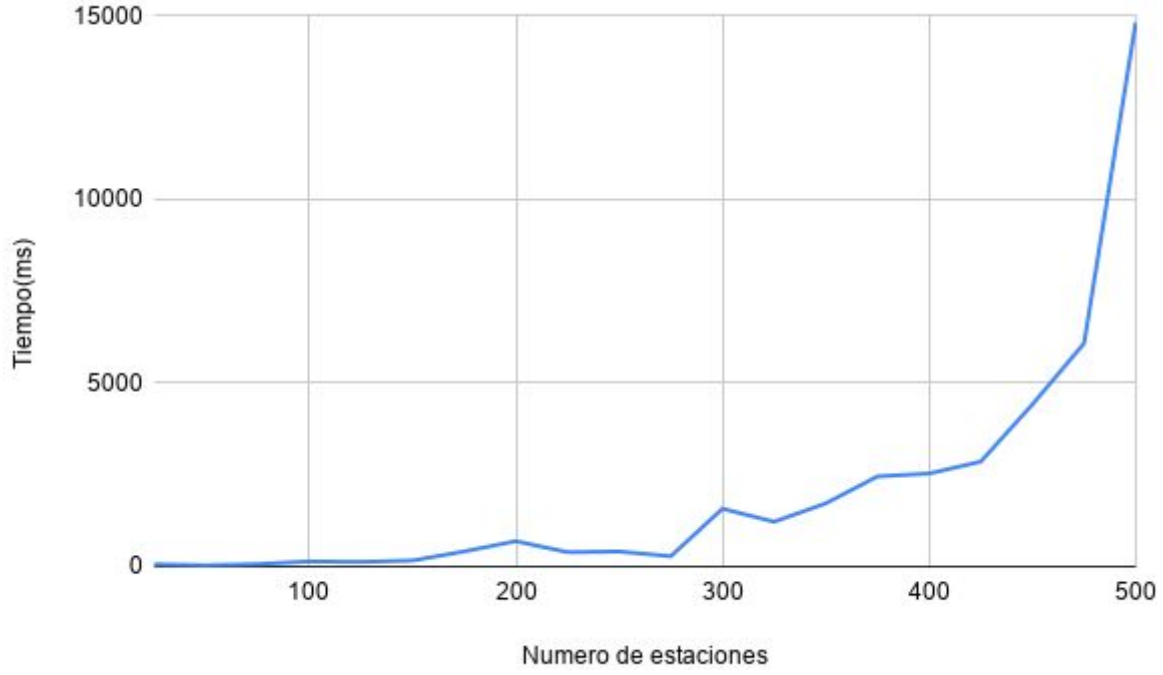
6.4.Dado el escenario de los apartados anteriores, estudiad cómo evoluciona el tiempo de ejecución para hallar la solución en función del número de estaciones, furgonetas y bicicletas asumiendo una proporción 1 a 50 entre estaciones y bicicletas y 1 a 5 entre furgonetas y estaciones. Para ello empezad con 25 estaciones e id aumentándolas de 25 en 25 hasta que veáis la tendencia. Usad el algoritmo de Hill Climbing y la misma heurística.

Observación: El tiempo para hallar la solución puede variar sustancialmente si aumentamos el número de estaciones.

Planteamiento: Ejecutamos el programa usando Hill Climbing y el escenario de los apartados anteriores varias veces cambiando el número de estaciones y observamos si se modifica el tiempo de ejecución.

Hipótesis: Todas las soluciones se obtendrán en el mismo tiempo de ejecución (H0) o el tiempo de ejecución variará sustancialmente.

Método: Tras hacer varias iteraciones, podemos observar que el tiempo va aumentando exponencialmente a medida que incrementamos el número de estaciones junto con el número de bicicletas y de furgonetas tal y como se puede observar en el gráfico:



6.5.Dado el escenario del primer apartado, estimad la diferencia entre el beneficio obtenido, la distancia total recorrida y el tiempo de ejecución para hallar la solución con el Hill Climbing y el Simulated Annealing para las dos heurísticas que habéis implementado (los experimentos con la primera ya los tenéis).

Observación: La calidad de la solución puede variar en función del heurístico que usemos y del algoritmo que utilicemos (Hill Climbing o Simulated Annealing).

Planteamiento: Ejecutamos los dos algoritmos con los diferentes heurísticos y observamos cómo afecta en la calidad de la solución.

Hipótesis: La calidad y tiempo de generación de la solución y la distancia que se recorrerá en cada solución será la misma para cada algoritmo(H0) o la calidad cambiará sustancialmente en función de qué algoritmo usemos y con qué heurístico.

Método: Ejecutaremos el programa con los parámetros indicados en el primer experimento: el número estaciones es 25, el número total de bicicletas es 1250, el número de furgonetas 5 y la demanda es equilibrada, de manera que generemos 4 soluciones (generadas por las diferentes combinaciones entre los 2 tipos de algoritmos y los 2 tipos de heurísticos: Heuristic1 corresponderá al heurístico que tiene en cuenta el coste y Heuristic2 hará referencia al heurístico que no tiene en cuenta el coste). El beneficio, la distancia total recorrida y el tiempo de ejecución para cada solución serán los siguientes:

	SA + Heuristic1	SA + Heuristic2	HC + Heuristic1	HC + Heuristic2
Beneficio	25€	119€	53€	87€
Distancia	27km	118km	8km	23km
Texe	103ms	116ms	125ms	106ms

Veremos que, obviamente, el beneficio será mayor cuando usemos el Heuristic2 ya que este genera soluciones sin tener en cuenta el coste y, por tanto, la distancia recorrida, al no penalizarse también será mayor.

Finalmente, mediante el estudio de los resultados obtenidos la hipótesis nula (H0) se podrá descartar de manera que podremos afirmar que tanto el beneficio obtenido, la distancia total recorrida y el tiempo de ejecución para hallar la solución serán diferentes para cada combinación entre tipo de algoritmo y de heurístico y, por tanto, que la elección del algoritmo y heurístico que usemos para encontrar la solución no será trivial.

6.6.Dado el escenario del primer apartado, generad problemas en los que la demanda corresponda a hora punta y estudiad si hay alguna diferencia en el tiempo de ejecución para resolver el problema. Usad el algoritmo que mejor resultados os haya dado.

Observación: La solución puede cambiar en función de elementos del dominio del problema como el tipo de demanda.

Planteamiento: Realizamos, dado el escenario del primer apartado, dos ejecuciones con el mismo algoritmo (el que mejor solución nos haya dado) pero cambiando el tipo de demanda y observamos si se produce algún cambio en la solución

Hipótesis: La solución no cambia según el tipo de demanda que tengamos en cuenta (H0) o la calidad de la solución cambiará en función del tipo de demanda que se asigne.

Método: Para realizar el experimento, ejecutaremos el programa teniendo en cuenta el escenario del primer apartado usando el algoritmo Simulated Annealing, la función heurística que tiene en cuenta el coste y generando la solución inicial de manera aleatoria. Esta ejecución la realizaremos varias veces cambiando la semilla para generar diferentes problemas y para cada semilla ejecutaremos una vez con demanda equilibrada y una vez con demanda de hora punta para ver si se producen cambios en la solución final.

Las soluciones finales para cada ejecución serán las siguientes:

	Demanda equilibrada	Hora punta
Semilla 1	22€	28€
Semilla 2	29€	43€
Semilla 3	44€	94€
Semilla 1234	33€	43€
Semilla 12345	18€	46€

Podemos observar claramente que la solución final obtendrá mejores resultados para la mayoría de las semillas (no podemos afirmar que lo hará en todas ya que no lo hemos comprobado) cuando se trate de una demanda de hora punta. Esto se puede explicar fácilmente sabiendo que en el escenario hora punta hay unas pocas estaciones con alta demanda , que además se encuentran a poca distancia entre ellas, y muchas estaciones con baja demanda. En el caso de escenario equilibrado, las demandas de las estaciones están repartidas por todo el mapa de forma más equitativa.

Por tanto, llegaremos a la conclusión de que la hipótesis nula (H0) propuesta es falsa y, por tanto, la calidad de la solución cambiará en función del tipo de demanda que se asigne de manera que la elección del tipo de demanda para la resolución del problema no será trivial.