ANÁLISIS DEL PROBLEMA

Un servicio de préstamo de bicicletas (Bicing) se puede encontrar con el problema de tener estaciones de servicio con una demanda de bicicletas superior al número que se encuentran en ese momento.
Para ser capaces de resolver el problema, la empresa Bicing nos proporciona tres datos para cada estación.

1. El número de bicicletas que no se moverán en la hora actual (**NumBicicletasNoUsadas**).
2. El número de bicicletas que habrá al final de la hora (**NumBicicletasNext**).
3. Demanda prevista de bicicletas para la próxima hora (**Demanda**).

El problema también se simplifica suponiendo que la ciudad es un cuadrado de 10x10 kilómetros con manzanas de 100x100 metros. Aunque tampoco es necesario tener presente esta simplificación, ya que tenemos una función que genera automáticamente los elementos del problema, devolviendo un vector de estaciones, con toda la información ya mencionada y sus coordenadas (**Estaciones(nest, nbic, dem, seed)**). Lo que sí que habrá que tener en cuenta es el cálculo de la distància entre estaciones. Al ser bloques no se podrá usar la distancia euclidiana (ya que no existen calles diagonales).

$$\mathbf{d(i,j) = |i_x - j_x| + |i_y - j_y|}$$

Existen dos escenarios diferentes de demanda: *Equilibrada* y *hora punta*. El escenario **equilibrado**, la demanda de bicicletas de cada estación es parecido, y en **hora punta**, algunas estaciones tienen más demanda que otras (de forma considerable).

Para la resolución del problema, disponemos de una o varias furgonetas que nos permiten trasladar bicicletas entre estaciones y evitar estaciones con escasez respecto la demanda.
Nos informan que Bicing nos **pagara** un euro por cada bicicleta que transportemos que haga que el número de bicicletas de una estación se acerque a la demanda. Y que por el contrario, nos **cobrarán** un euro por cada bicicleta que transportemos que aleje a una estación de su previsión.
También nos informan que el **transporte** de bicicletas supone un coste en función del número de bicicletas que transportemos en una furgoneta y los kilómetros recorridos.
Ante està información, hay que tener presente que en el caso que no hagamos ningún traslado de bicicletas, obtendremos un beneficio de 0 euros. Entonces no hay que preocuparse por perder dinero, ya que si nuestros traslados dieran un beneficio negativo, únicamente cancelaríamos todos ellos.
Ya podemos prever que seremos capaces de obtener mayor beneficio en un escenario de **hora punta**.

La solución que hay que aportar son los traslados que harán las furgonetas para una hora concreta, indicando el origen de donde recogen bicicletas y los destinos donde las repartirá. Y obviamente el número que recogerá y dejará en cada una de las estaciones.

Para el problema tenemos varias restricciones y consideraciones a tener en cuenta:

- El número de bicicletas total es constante, no se puede quitar o añadir bicicletas al sistema.
- El número de bicicletas que caben en una estación es ilimitado.
- No es necesario usar todas las furgonetas.
- Las furgonetas cargan bicicletas solo en la estación origen.
- No puede haber dos furgonetas que cojan bicicletas de la misma estación.
- Cada furgoneta en una hora solo puede hacer un único viaje.
- Por cada viaje solo se pueden transportar bicicletas a como máximo 2 estaciones.
- Las furgonetas tienen una capacidad màxima de 30 bicicletas.
- Varias furgonetas pueden dejar bicicletas en la misma estación.

Únicamente tendremos que adaptar nuestra solución a estas restricciones, pero no nos va a suponer mayor inconveniente.

Vemos que el espacio de búsqueda de este problema es inmenso. Disponemos de **F** furgonetas y **E** estaciones con **B** bicicletas en total. Cada solución serán tres estaciones (origen, primer destino, segundo destino) con su respectivo número de bicicletas a recoger y repartir. Como este número podemos hacer que sea directamente dependiente de cada combinación de estaciones, no lo tendremos en cuenta.

Entonces, cada furgoneta tiene una estación origen, sin que se pueda repetir y dos estaciones de destino que sí se pueden repetir entre todas las furgonetas. El espacio de búsqueda será de:

$$\binom{E}{F} * E^F * E^F$$

Donde E es el número total de estaciones, y F el número de todas las furgonetas.

Con únicamente 5 furgonetas y 25 estaciones, el espacio de búsqueda es del orden de 10^{18}, superior a lo que podríamos tratar con funciones heurísticas de forma eficaz.

Nuestra solución es tratarlo como un problema de búsqueda local, donde partiendo de una solución inicial la vamos modificando para encontrar una mejor.

IMPLEMENTACIÓN DEL PROBLEMA

Representación del Estado

Como nos encontramos ante un problema de búsqueda local, la representación del estado tiene que representar una posible solución y poder representarlas todas.

En nuestro problema, una solución representa los traslados que harán las furgonetas, teniendo en cuenta los siguientes criterios y restricciones.

1. Las furgonetas no pueden exceder su capacidad (30 bicicletas)
2. Las furgonetas no pueden visitar más de dos estaciones.
3. Las furgonetas cargan bicicletas solo en la estación de origen.
4. Varias furgonetas pueden dejar bicicletas en la misma estación.
5. Dos furgonetas no pueden recoger de la misma estación.

El elemento básico para nuestra solución, son los traslados de las furgonetas. Para ello habíamos pensado en hacer un único vector donde cada elemento representa una furgoneta, y su contenido fuera otro vector que representa el origen y sus destinos, con la información del número de bicicletas que son transportadas al primer destino y al segundo destino.
Està representación sería válida ya que nos permite representar cualquier solución de nuestro problema, pero vimos que el tratamiento de los vectores dinámicos sería demasiado complejo para los operadores.

Teniendo en cuenta la segunda restricción (las furgonetas no pueden visitar más de dos estaciones) y que el número màximo de furgonetas es siempre constante, hemos escogido como representación tres arrays de pares. Uno llamado **Origen** con la información (para cada furgoneta) de la estación origen y el número de bicicletas recogidas. Otro llamado **dest1** con la información del primer destino y el numéro de bicicletas repartidas. Y el ùltimo llamado **dest2**, homólogo a dest1, pero para el segundo destino.
(Esta será la representación que usaremos en adelante)

Con esta solución hemos podido satisfacer tanto la segunda como la tercera restricción, y el cuarto criterio.

La información de las diferentes estaciones es un índice de la estación correspondiente en un array que contiene todas las estaciones. Este array **est** será un array estàtico el cual será inicializado siempre antes de crear la solución inicial con la función **setEstaciones**.
Para poder representar furgonetas que no usamos (sin origen) o que no tienen alguno de los destinos, el índice de la estación lo **igualamos a "-1"**.

Un elemento esencial de nuestra solución es el array **bicisLibres**. Este representa las bicis que tiene en exceso o en escasez cada una de las estaciones. El cálculo lo hemos obtenido con el mínimo de entre las bicicletas no usadas, y la diferencia del número de bicis con la demanda para la siguiente hora. Entonces, si una estación tiene bicis en exceso, **bicisLibres** indicará el número de bicis que podemos recoger sin quitar por debajo de la demanda. Si la estación

tiene escasez, entonces tendremos un valor negativo que nos indica el número de bicis que hacen falta para llegar al equilibrio con la demanda.
Este array nos permite ahorrar mucho tiempo, al no tener que recorrer todos los traslados de las furgonetas para calcular las bicicletas que se recogen o se reparten en una estación.
Ahora para poder cumplir la restricción de la **capacidad màxima de 30 bicicletas**, únicamente limitamos el valor en **bicisLibres** a 30, y somos consecuentes usando siempre el cálculo de ese array.

La última restricción (**dos furgonetas no pueden recoger de la misma estación**), la tratamos dependiendo de los operadores. Como finalmente vamos a implementar el operador de cambio de estación origen, hemos tenido la necesidad de poner en nuestra solución un array de booleanos **esOrigen**, que nos indica cuando una estación ya es seleccionada como origen de una furgoneta.

A partir de aquí nuestra solución ya estaría completa. Para hacerla más eficiente hemos añadido los siguientes elementos:

numEst : Número de estaciones (estático)
distEst : Matriz con la distancia entre las estaciones (estático)
nfurgos : Número de furgonetas (estático)

También se han añadido todos los “getters” necesarios.

Operadores de Búsqueda

Para programar los operadores hemos querido que todos los cambios en la solución nos lleven a la mejor solución posible (para ese único cambio). Por ejemplo, si cambiamos el primer destino de una furgoneta, la solución se tiene que adaptar para que esa furgoneta reparta a ese nuevo destino el número de bicicletas que nos dé el mayor beneficio (teniendo en cuenta el límite de nuestro origen).
Esa adaptación es relativamente sencilla si usamos el array **bicisLibres**.

También en cada operación de cambio miramos de evitar incumplir las restricciones. Aunque también delegamos esa tarea a la función de sucesores.

El conjunto de operadores que hemos implementado para cubrir el mayor espacio de soluciones posibles han sido los siguientes:

- **changeOrigen(**int i, int idE):
  Descripción: Establece la estación con identificador **idE** como la estación origen para la furgoneta con identificador "**i**".

  Condiciones de aplicabilidad: La estación **idE** no es origen de otra furgoneta (**esOrigen**) y tiene exceso de bicicletas (**bicisLibres**).

  Efectos en la solución: Asignamos a **origen[i]** un nuevo pair con el nuevo identificador de la estación y establecemos el número de bicicletas recogidas a cero. Para que la solución sea compatible con los posibles destinos de esa furgoneta, adaptamos el número de bicis repartidas a esos

destinos, y consecuentemente el número de bicis recogidas en origen (teniendo en cuenta la solución más óptima para ese traslado).
Actualizamos **esOrigen** y **bicisLibres**.

Ramificación: Todas las combinaciones de furgonetas con estaciones que cumplan la condición de aplicabilidad. (**nfurgos*numEst**)

- **changeDest1**(int i, int idEst):
  Descripción: Establece la estación con identificador **idEst** como la estación del primer destino para la furgoneta con identificador "**i**".

  Condiciones de aplicabilidad: La furgoneta **i** tiene asignada una estación origen. La estación **idEst** tiene escasez de bicicletas (**bicisLibres**).

  Efectos en la solución: Asignamos a **dest1[i]** un nuevo pair con el nuevo identificador de la estación. Si el número de bicicletas que tiene en escasez la nueva estación es inferior al número de bicicletas repartidas al anterior destino (1), cubrimos toda la escasez del nuevo destino, y las bicicletas sobrantes las repartimos al segundo destino y al origen (priorizando la escasez en el segundo destino).
  En el caso de que el número de bicicletas en escasez del nuevo destino sea superior a las repartidas anteriormente, miramos si la estación origen tiene exceso de bicicletas, y rellenamos la escasez del destino (1) consecuentemente.
  Actualizamos **bicisLibres**.

Ramificación: Todas las combinaciones de furgonetas y estaciones que cumplan las condiciones de aplicabilidad (**nfurgos*numEst**).

- **changeDest2**(int i, int idEst):

  Descripción: Establece la estación con identificador **idEst** como la estación del segundo destino para la furgoneta con identificador "**i**".

  Condiciones de aplicabilidad: La furgoneta **i** tiene asignada una estación origen y un primer destino. La estación **idEst** tiene escasez de bicicletas (**bicisLibres**).

  Efectos en la solución: Asignamos a **dest2[i]** un nuevo pair con el nuevo identificador de la estación. Si el número de bicicletas que tiene en escasez la nueva estación es inferior al número de bicicletas repartidas al anterior destino (2), cubrimos toda la escasez del nuevo destino, y las bicicletas sobrantes las asignamos al origen.
  En el caso de que el número de bicicletas en escasez del nuevo destino sea superior a las repartidas anteriormente, miramos si la estación origen tiene exceso de bicicletas, y rellenamos la escasez del destino (2) consecuentemente.
  Actualizamos **bicisLibres**.

  Ramificación: Todas las combinaciones de furgonetas y estaciones que cumplan las condiciones de aplicabilidad (**nfurgos*numEst**).

- **swapDest1**(int i, int j):

  Descripción: Intercambia el primer destino de la furgoneta con identificador **i** con el primer destino de la furgoneta con identificador **j**.

  Condiciones de aplicabilidad: La furgoneta **i** y la furgoneta **j** (**i != j**) tiene asignada una estación origen. Al menos una de las dos furgonetas tiene asignada un primer destino.

  Efectos en la solución: Si existen los destinos, asignamos a **dest1[i]** un nuevo pair con la estación de **dest1[j]** y viceversa. Adaptamos el número de bicis que repartimos al primer como al segundo destino de cada furgoneta (a la mejor solución), y somos consecuentes con el número de bicicletas que recogemos del origen.
  Actualizamos **bicisLibres**.

  Ramificación: Todas las combinaciones de pares de furgonetas que cumplan las condiciones de aplicabilidad (**nfurgos*nfurgos**).

- **swapDest2**(int i, int j):

  Descripción: Intercambia el segundo destino de la furgoneta con identificador **i** con el segundo destino de la furgoneta con identificador **j**.

  Condiciones de aplicabilidad: La furgoneta **i** y la furgoneta **j** (**i != j**) tiene asignada una estación origen y un primer destino. Al menos una de las dos furgonetas tiene asignada un segundo destino.

Efectos en la solución: Si existen los destinos, asignamos a **dest2[i]** un nuevo pair con la estación de **dest2[j]** y viceversa. Adaptamos el número de bicis que repartimos al segundo destino de cada furgoneta (a la mejor solución), y somos consecuentes con el número de bicicletas que recogemos del origen.
Actualizamos **bicisLibres**.

Ramificación: Todas las combinaciones de pares de furgonetas que cumplan las condiciones de aplicabilidad (**nfurgos*nfurgos**).

- **swapDest1Dest2**(int i):
  Descripción: Intercambia el primer destino de la furgoneta con identificador **i** con el segundo destino de la misma furgoneta.

  Condiciones de aplicabilidad: La furgoneta **i** tiene asignado un origen, un primer destino y un segundo destino.

  Efectos en la solución: Únicamente intercambiamos **dest1[i]** con **dest[2]** (incluyendo el número de bicicletas repartidas).

  Ramificación: El número de furgonetas que cumplan la condición de aplicabilidad (**nfurgos**).

Con estos operadores podemos explorar gran parte del espacio de soluciones. Pero hay que destacar la importancia de algunos operadores por encima de la de otros operadores. Dependiendo de la solución inicial que implementemos, hay operadores que no tendrán relevancia para nuestros algoritmos de búsqueda local.
Por ejemplo, si en nuestra solución inicial asignamos para cada furgoneta una estación origen con el mayor número de bicicletas en exceso, el operador **changeOrigen** no va a tener mucha influencia.

También podemos ver que el operador **changeDest1** tendrá más relevancia que el operador **changeDest2**, ya que este último únicamente es llamado cuando existe un primer destino, y en ese caso, la mayor parte de las bicicletas excedentes del origen ya han sido asignadas a ese primer destino.
Para **swapDest1** y **swapDest2** sucede del mismo modo, teniendo **swapDest1** más relevancia.

Y entre los operadores de "**change**" y "**swap**", vemos que los "change" tendrán más importancia que los "swap", ya que los primeros asignan de entre todas las estaciones la mejor posible dando un margen de beneficio mayor que si únicamente intercambiamos estaciones que ya han sido asignadas, donde probablemente la diferencia entre los beneficios obtenidos no sea tan destacada.

Estas observaciones se han comprobado empíricamente, viendo como el algoritmo de HillClimbing tiene mayor tendencia a conservar los cambios hechos con lo operadores más relevantes. Posteriormente en el análisis de los experimentes escogeremos el conjunto de operadores que nos den mayor eficiencia en cuanto al espacio explorado y el rendimiento.

Función heurística

Para implementar la función heurística, hemos tenido en cuenta todos los factores que intervienen en el problema.

La **primera función heurística** nos pide maximizar el beneficio obtenido por los traslados de las bicicletas, sin tener en cuenta el coste que conlleva el transporte.
En el enunciado nos dicen que ganamos un euro por cada bicicleta que transportemos que haga que el número de bicicletas de una estación se acerque a la demanda. Y perdemos un euro por cada bicicleta que transportemos que aleje a una estación de su previsión.
Teniendo en cuenta esto, hemos implementado esta primera función heurística únicamente considerando el beneficio (en euros) ganado y maximizando-lo.
El cálculo es bastante sencillo. Recorremos todos los traslados (**3*nfurgos**) y actualizamos un contador, para cada estación, de las bicicletas que repartimos o recogemos. Finalmente calculamos el beneficio mirando si nos acercamos a la demanda o si nos alejamos de la previsión.
Esta heurística también la podemos usar para evaluar la solución final, ya que no tiene en cuenta ningún otro factor.
Aunque pueda parecer sencilla, pensamos que es la más adecuada, ya que nos permite de forma rápida y generalizada comparar dos soluciones y generar la mejor de ellas.

En la **segunda función heurística**, esta vez hay que maximizar de nuevo el beneficio obtenido por los traslados de las bicicletas, y también minimizar los costes del transporte de las bicicletas.
Para la primera parte de maximización, reutilizamos el cálculo que hace la primera función heurística. Esta nos devolverá el beneficio en euros.
El coste que supone el transporte de bicicletas està reflejado en la siguiente fórmula:

km*((nb+9) div 10)

Donde **nb** es el número de bicicletas que transportamos en una furgoneta y **km** los kilómetros que recorre.
Entonces, el coste total del transporte hay que calcularlo aplicando la fórmula para cada trayecto de cada una de las furgonetas. La matriz **distEst** nos facilitara el cálculo al ya tener la distancias de cada par de estaciones precalculadas.
La segunda función heurística simplemente devolverá la resta del beneficio obtenido cons los traslados de las bicicletas y el coste del transporte de estas. Y lo maximizará.
De nuevo este heurístico puede evaluar la solución final, ya que tampoco hemos añadido ningún otro factor. Y de entre dos soluciones, siempre escogerá la mejor de ellas.

Una ponderación que intentamos añadir fue el considerar el número de estaciones que estuvieran en equilibrio (número de bicis igual a la demanda) y maximizar-lo. Pero debido a que nuestros operadores siempre adaptan la solución a la mejor posible, està ponderación la encontramos poco eficaz.

Solución Inicial

Para hallar una solución inicial, hemos implementado tres métodos, de los cuales finalmente dos han sido válidos.

El primero y más sencillo de ellos es simplemente crear un solución inicial vacía. En està solución únicamente inicializamos los array **origen**, **dest1**, y **dest2** con el identificador de la estación igual a **-1**.
Debido a la implementación de nuestra función heurística, esta solución inicial vacià no funcionaria para el algoritmo HillClimbing. Eso es debido a que todos los operadores (excepto **changeOrigen**) tienen como condición de aplicación que el trayecto de la furgoneta a modificar tenga asignado un origen. Entonces, el único operado posible (changeOrigen) tampoco se va aplicar ya que el hecho de cambiar la estación de origen, cuando no existe ningún destino, no nos modifica el valor de la heurística. Entonces la solución que devolverá será la misma que la inicial, una solución vacía.

Nuestros siguientes dos métodos se basan en la forma de inicializar las estaciones origen de cada una de las furgonetas (a ser posible).

Lo primero que pensamos para el **segundo método** fue en asignar como origen a las estaciones con mayor número de bicicletas en exceso, empezando con las de de mayor valor. Este método tendría que recorrer para cada furgoneta todas las estaciones buscando la estación con el mayor exceso. A priori supondría un coste de **nfurgos * numEst**. Si tenemos en cuenta la restricción de la capacidad máxima de 30 bicicletas, podemos dejar de recorrer las estaciones cuando

encontremos una estación con un exceso de 30 bicicletas, ya que no encontraremos un exceso superior.

Para el **tercer método** hemos querido ahorrarnos la búsqueda (para cada furgoneta) de la estación disponible con mayor exceso de bicicletas . En vez de eso, asignamos al origen de cada furgoneta la primera estación que tenga un mínimo de bicicletas disponibles para distribuir. Ese mínimo será un valor a determinar.

Obviando la solución vacía, observamos que el método más completo és el segundo, ya que directamente nos asigna a cada furgoneta las estaciones que nos permiten repartir más bicicletas y por tanto obtener mayor beneficio. Probablemente estas estaciones de origen no serán modificadas por ninguno de los operadores.
Aun así, este segundo método tiene un coste (**nfurgos*numEst**) mientras que el tercer método tiene un coste lineal (**numEst**) ya que únicamente nos basta con recorreremos el array de estaciones una sola vez.

Experimentalmente hemos comprobado que para ciertos valores del mínimo en el último método, se encuentra una mejor solución que con el segundo método. El problema reside en encontrar ese mínimo ideal. Para ello habría que hacerlo experimentalmente para cada problema o con una análisis previo, lo que haría perder la ventaja de rendimiento respecto al segundo método.

Por lo tanto, nos quedaremos en un principio con el segundo método para crear la solución inicial.

EXPERIMENTOS

Experimento 1 - Operadores

Queremos encontrar cuál es la mejor combinación de operadores que podemos construir con todos los que hemos creado. Tenemos los 6 siguientes:

1. ChangeOrigen
2. ChangeDest1
3. ChangeDest2
4. SwapDest1
5. SwapDest2
6. SwapDest1Dest2

El caso es que, por construcción, ningún sucesor tiene sentido si no se aplica el segundo operador, por lo que todos nuestros experimentos van a incluirlo. El estudio será como sigue: construiremos una solución inicial **aleatoria** (en el escenario que se pide, por supuesto) y ejecutaremos, en cada una de ellas, el procedimiento de búsqueda local con los siguientes conjuntos de operadores: {2}, {2, 3}, {1, 2, 3}, {1, 2, 3, 4}, {1, 2, 3, 4, 5}, {1, 2, 3, 4, 5, 6}, usando la misma semilla en cada prueba individual. Esta sucesión de ejecuciones se llevará a cabo con 10 soluciones aleatorias distintas; de manera que podremos registrar el **número de veces** en los que cada uno de los subconjuntos nos lleva a la mejor solución (o a la misma que otros, si es el caso). De esta forma seremos capaces de encontrar el conjunto **mínimo** de operadores que maximiza nuestra función de evaluación.

La tabla de resultados hallada, para cada conjunto, utilizando 10 semillas distintas, es la que se muestra a continuación:

	{2}	{2, 3}	{1, 2}	{1, 2, 3}	{1, 2, 3, 4}	{1, 2, 3, 4, 5}	{1, 2, 3, 4, 5, 6}
1	50	50	63	63	63	63	63
2	65	65	68	68	68	68	68
3	76	76	92	92	92	92	92
4	85	85	105	105	105	105	105
5	60	60	86	86	86	86	86
6	68	68	73	78	78	78	78
7	48	48	57	57	57	57	57
8	60	60	60	60	60	60	60
9	64	64	79	79	79	79	79
10	66	66	80	85	85	85	85

A raíz de las observaciones, tenemos que decidir entre el tercer y el cuarto subconjunto de operadores. Nuestro razonamiento es el siguiente: suponemos que ambos conjuntos son "igual de buenos" (es decir, ambos tienen probabilidad p=0.5 de ofrecer el mejor resultado), y buscamos en la tabla de distribución **binomial** cuál es la probabilidad de que solo en una ocasión de las diez el cuarto subconjunto ofrezca un resultado mejor que el tercero (notemos que la única diferencia en sus respectivas soluciones obtenidas se da en la réplica 6). Esta probabilidad es 0.0098. Siendo esta tan pequeña, concluimos que ambos (el tercer y cuarto conjunto) son equivalentes para el problema, por lo tanto nos decantamos por el **tercero**, por tener éste menos operadores.

Experimento 2 - Solución Inicial

Para escoger la estrategia de generación de nuestra solución inicial, de forma totalmente análoga al modo de trabajar del experimento anterior, vamos a calcular, para un total de 10 ejecuciones, el coste que obtenemos aplicando nuestro conjunto de operadores sobre las dos estrategias que hemos desarrollado: una *psuedo-aleatoria* (que asigna los orígenes de las furgonetas siguiendo el orden lógico de las estaciones), y otra que coloca inicialmente las furgonetas en las estaciones más excedentarias (estrategia avariciosa).

Para este caso, omitiremos la tabla de resultados ya que, como el lector podrá deducir, en todos y cada uno de los escenarios la solución inicial generada con el segundo método nos lleva a un mejor resultado. Por lo tanto (dado que la probabilidad de que de 10 pruebas, 10 las gane la versión avariciosa, es de 0.001; basándonos nuevamente en la distribución binomial), concluimos que la generación avariciosa es la mejor opción para nuestro algoritmo.

Experimento 3 - Parámetros para SA

El objetivo de este experimento es encontrar valores apropiados para los parámetros usados en la técnica de simulated annealing. Para determinarlos hemos seguida la metodología mostrada en el enunciado de esta práctica. Lo primero es hallar los valores de k y λ. Hemos observado que el número de iteraciones depende, esencialmente, de λ, influyendo también, aunque en menor medida, el número de pasos por iteración. Así que decidir k y λ nos permitirá conocer el número máximo de iteraciones, que aproximaremos a la potencia de 10 inmediatamente superior para obtener un número con suficientes divisores para facilitar la búsqueda de un buen valor para el número de pasos por iteración.

Para ello establecemos los conjuntos de k y λ que exploraremos. Inspirándonos en el análisis realizado para TSP en el enunciado escogemos $k = \{1, 5, 25, 125\}$ y $\lambda = \{1, 0.1, 0.01, 0.001\}$. Esta cota para λ se debe al gran incremento en el tiempo de ejecución del algoritmo para valores más pequeños.
La siguiente tabla muestra el beneficio esperado en € (10 ejecuciones por pareja de valores) en el escenario de los experimentos anteriores para cada combinación de los valores escogidos:

k / λ	1	0,1	0,01	0,001
1	67,3	64,5	80,7	70,8
5	69,4	69,6	73,1	72,7
25	70,3	70,9	69	67,7
125	76,3	61,9	75,8	72,8

Nos quedamos la combinación que ofrece un mayor beneficio esperado, $k = 1$ y $\lambda = 0{,}01$. Fijamos 100000 iteraciones máximas, más que suficiente para los nodos que explora el algoritmo con estos parámetros según hemos observado experimentalmente (en particular, siempre alrededor de los 74000). Vemos en la siguiente tabla el beneficio esperado en € para diferentes pasos por iteración:

1 paso	10 pasos	100 pasos	200 pasos	500 pasos
63,1	69,4	74	75	68,5

Por lo tanto, para los siguientes experimentos en que apliquemos simulated annealing fijaremos los siguientes parámetros: 100000 iteraciones máximas, 200 pasos por iteración, $k = 1$, $\lambda = 0{,}01$.

Experimento 4 - Tiempo de ejecución

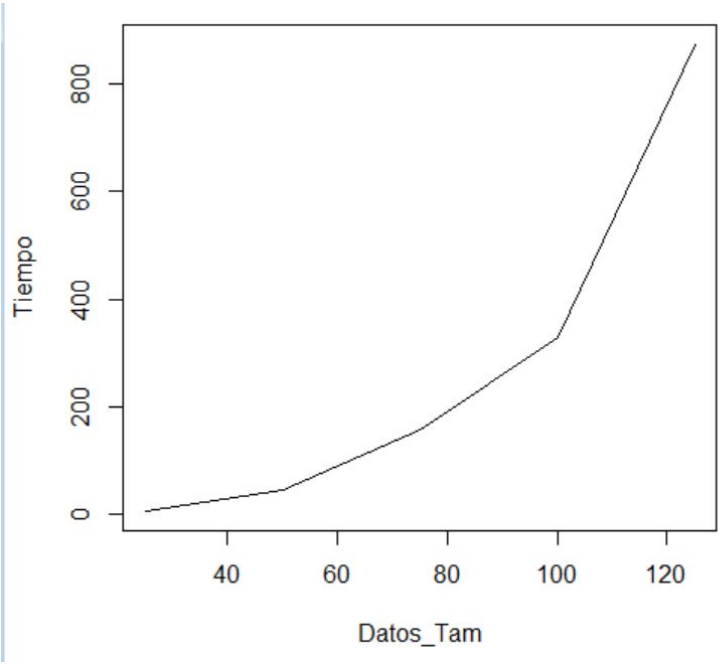
Podemos asumir que el crecimiento del tiempo estará en función del factor de ramificación y del tamaño del espacio de búsqueda. Para nuestra implementación, tenemos que el **factor de ramificación** de nuestros dos operadores es, potencialmente, **F•E**, donde F es el número de furgonetas; y E, el número de estaciones. Por otro lado, el espacio de búsqueda para el origen es **O(E sobre F)**. Dado que una vez hechas las asignaciones de origen, podemos asignar cualquier furgoneta a cualquier estación (exceptuando la que tiene como origen), tenemos que la expresión resultante para el tamaño del espacio de búsqueda puede expresarse así: $\frac{E!}{F!(E-F)!} \cdot E^{F} \cdot E^{F}$

Asumiremos también que el tiempo para los problemas de un mismo tamaño es comparable. Realizaremos varias ejecuciones para cada tamaño y usaremos los valores medios para ajustar la función del tiempo.
Basándonos en los datos expuestos en el parágrafo anterior, deducimos que el tiempo ha de seguir, forzosamente, una función creciente respecto al tamaño del problema.

El procedimiento a seguir se resume de la siguiente manera: iremos incrementando de 25 en 25 estaciones (partiendo de la configuración inicial) y preservando las proporciones entre furgonetas, estaciones y bicicletas. Repetiremos esto en 10 escenarios distintos, y obtendremos la media y la desviación del tiempo de ejecución para cada orden de magnitud.
Los resultados obtenidos se muestran en esta tabla:

Tamaño	25	50	75	100	125
Media [ms]	5.9	45.7	159.1	328.5	873.1
Desviación	5.66	66.24	59.48	124.11	163.75

El siguiente gráfico de líneas nos ofrece una visualización clara del comportamiento del tiempo de ejecución.



Función del tiempo respecto al tamaño del problema

Sin mayor necesidad de análisis puede apreciarse que la gráfica crece como una función **exponencial**.

Experimento 5 - Diferencias entre el HC y el SA

Para estimar la diferencia entre el beneficio, la distancia y el tiempo de ejecución de las soluciones halladas entre el HC y el SA realizamos, para cada una de estas medidas, la media estadística de la diferencia entre sendos algoritmos. Estas medias se han obtenido haciendo, nuevamente, 10 repeticiones para 10 semillas distintas; calculando, finalmente, una media de la media (valga la redundancia) de los beneficios, distancias y tiempos. Esto es remarcable sobre todo para el SA, ya que al aceptar soluciones peores, presenta mucha más variabilidad en los recorridos que se producen para ejecuciones distintas de una misma semilla.

Por nuestros cálculos anteriores, podemos estimar que la media del beneficio obtenido con HC aplicando el primer heurístico es de 76.5€. También sabemos que el tiempo medio de ejecución es de 5.9 ms y que la distancia recorrida en media es de 26.33 kilómetros.

La siguiente tabla resumen presenta los resultados obtenidos para el SA con el primer criterio, utilizando los parámetros hallados en el tercer experimento.

	Beneficio	Distancia[Km]	Tiempo[ms]
1	72	39.9	171
2	78	23.8	156
3	92	57.7	117
4	105	50.9	137
5	92	59.4	116
6	83	32.4	123
7	60	41	89
8	60	57.3	89
9	79	51	121
10	93	64.8	78

Con todo, obtenemos una media de 81.4€, 47.82 kilómetros y 119.7 ms para el beneficio, la distancia recorrida y el tiempo de ejecución, respectivamente.

Debemos notar no sólo que el beneficio que se obtiene con el SA (para el primer criterio) es superior al que se consigue con el HC, sino que además no existe ningún caso en el cual con el HC se obtenga un mejor resultado que con el SA. Por el contrario, en 7 de las 10 réplicas, SA sí que supera la respectiva solución del HC. Por lo tanto, debemos concluir que, si sólo buscamos maximizar el beneficio (el transporte es gratis), nuestra mejor opción es el SA, a pesar de que su tiempo de ejecución consumido, y su distancia recorrida son considerablemente superiores al de la ejecución con el HC.

Sin embargo, las cosas no son tan claras para el segundo criterio. Esto lo vamos a ver analizando las siguientes dos tablas, que sintetizan la media de los resultados obtenidos para el HC (la primera) y el SA (la segunda) aplicando la segunda heurística.

	Beneficio[€]	Distancia[Km]	Tiempo[ms]
1	49.5	8	15
2	57	7.1	12
3	68.3	9.5	19
4	64.5	9.4	8
5	65.8	8.6	15
6	52	8.8	11
7	43.9	9.9	9
8	36.4	12.1	12
9	45.4	10.2	10
10	64.6	7.3	13

	Beneficio[€]	Distancia[Km]	Tiempo[ms]
1	47.8	11.7	154
2	62.8	7.1	127
3	63.3	11.1	169
4	72.7	13.5	108
5	67.8	8.8	139
6	62	8.7	105
7	44.3	9.5	91
8	37.2	9.3	93
9	57.8	12.9	86
10	68.4	7.8	112

Para este nuevo paradigma conseguimos un beneficio medio de 54.74€ con el HC, y de 58.41 con el SA. Si bien es cierto que el resultado sigue siendo ligeramente favorable para el SA, las diferencias de rendimiento (solución a solución) son considerablemente menos holgadas respecto al anterior criterio. Y dado el hecho de que el tiempo medio de ejecución consumido por el SA (118.4 ms) es casi 10 veces el tiempo empleado por el HC (12.4 ms), quizá la elección del SA ya no sea tan clara como sí lo era en el anterior caso.

Experimento 6 - Diferencias en el tiempo de ejecución en equilibrio y hora punta

Para este experimento hemos optado por usar SA, que aunque tiene un peor desempeño en cuanto a lo temporal, brinda unos beneficios mayores. Además, el superior consumo de tiempo de esta técnica sobre HC se acentúa aún más al ejecutarla en una situación de hora punta, por lo que la comparación de ambos escenarios (equilibrio u hora punta) resulta más interesante.

En el experimento anterior obtuvimos los tiempos de 10 ejecuciones del programa usando simulated annealing. Recuperando esas medidas calculamos un tiempo medio de ejecución de 118,4 ms. Repetimos ahora las medidas para un escenario de hora punta. Los experimentos no se han realizado con la misma semilla, pero eso no resta valor a las comparaciones, ya que nos interesa el tiempo medio de ejecución, no las diferencias individuales entre unos ciertos escenarios arbitrarios. Se han obtenido las siguientes medidas:

	Tiempo en ms
1	103
2	92
3	86
4	78
5	80
6	91

7	174
8	114
9	87
1o	137
Media:	104,2

Vemos que los tiempos medios de ejecución son bastante similares, aunque en el caso de hora punta son inferiores. Esto puede ser debido a que las diferencias entre estaciones bajo demanda y estaciones con excedentes son más exageradas, sin muchas estaciones cercanas a su demanda, facilitando la exploración de soluciones al existir un menor número de combinaciones con beneficios similares.

Experimento 7 - Diferencias en el tiempo de ejecución en equilibrio y hora punta

En este experimento la intención es hallar una aproximación del número óptimo de furgonetas para maximizar los beneficios. Seguimos el método descrito en el enunciado del experimento e incrementamos el número de furgonetas de que disponemos. Realizamos la prueba con simulated annealing por el mismo motivo que en el apartado anterior: da mejores resultados.
Las siguientes tablas muestran el beneficio en € esperado para diferente número de furgonetas, en equilibrio y hora punta, respectivamente:

Nº de furgonetas	5	10	15
Beneficio	50,83	72,67	63,95

Nº de furgonetas	5	10	15
Beneficio	64,27	88,95	78,96

Observamos que los valores no convergen, sino que existe en ambos casos un máximo en el beneficio cuando usamos 10 furgonetas.