Documentación práctica

Identificación del problema

El objetivo de esta práctica es ayudar a Bicing a optimizar la distribución de bicicletas para que los usuarios encuentren bicis cuando las necesiten.
Para empezar, Bicing nos proporciona la información siguiente sobre sus estaciones. Para cada estación, sabemos:

- las bicis que no se van a usar, que son las que podremos trasladar a otras estaciones
- la previsión del número de bicis que habrá en una estación a la hora siguiente de la actual solo teniendo en cuenta cambios que realicen los usuarios
- la demanda de bicis que habrá en una estación a la hora siguiente a la actual

Con esta información podemos calcular fácilmente si en una estación faltan o sobran bicis. Nuestra tarea será mover las bicis de las estaciones en las que sobran bicis a las estaciones en las que faltan.

Estamos situados en una ciudad cuadrada cuyas estaciones están en los cruces entre calles. Cada estación tiene unas coordenadas por lo que podemos calcular la distancia entre dos estaciones (distancia Manhattan). Hay E estaciones y B bicicletas repartidas por ellas. Existen dos escenarios posibles de demanda:

- equilibrada: la demanda de cada estación es similar
- hora punta: la demanda de algunas estaciones es mucho más elevada que en otras

Disponemos de F furgonetas que nos permitirán mover las bicis de una estación a otra. Cada furgoneta puede transportar 30 bicis como máximo y puede realizar un único viaje de una hora a dos estaciones. Cada furgoneta debe empezar en una estación inicial distinta de las demás.
Tenemos un acuerdo con Bicing que nos permitirá obtener beneficios económicos si nuestra flota de furgonetas ayuda a cubrir la demanda de las estaciones. Concretamente, ganamos 1€ por cada bici que transportemos que haga que el número de bicis de una estación se acerque a su demanda. De forma contraria, perderemos 1€ por cada bici transportada que aleje una estación de su previsión. Otro factor importante es el de la gasolina. Las furgonetas consumen una cierta cantidad de gasolina por kilómetro y esto tiene un coste proporcional al número de bicis que llevamos en la furgoneta.

El objetivo es obtener una asignación de estaciones para cada furgoneta para una hora concreta. Es decir, dado un escenario concreto debemos indicar qué viajes deberá realizar cada furgoneta así como las bicis que traslada en cada estación.

Podemos ver que el tamaño del espacio de búsqueda es muy grande ya que puede haber muchas estaciones y que encontrar la solución óptima del problema puede ser inviable. Además los operadores de cambio de estado no tienen asociado un coste, el coste nos viene dado por factores independientes del operador. Por ejemplo, mover una bici de una estación a otra podría ser un operador. No obstante, mover esta bici puede producir 1€ de beneficio, 1€ de pérdidas o no tener ningún impacto. Por lo tanto este operador no puede tener asociado un coste determinado. Además hay que añadir el coste en gasolina que supone mover esta bici, lo que hace que todo sea todavía más complicado.
En resumen, podemos afirmar que es adecuado abordar este problema como un problema de búsqueda local y por lo tanto los algoritmos que utilizaremos serán los usados para búsqueda local: Hill Climbing y Simulated Annealing.

El planteamiento que hemos establecido es el siguiente:

- estado inicial: será una solución del problema, estudiaremos más adelante cómo la vamos a generar
- operadores: nos permitirán explorar el espacio de soluciones y de esta forma cambiar la solución, pero no tienen un coste asociado
- función de calidad: diseñaremos un heurístico que permita valorar lo buena que es una solución

Implementación del estado

Cuando hemos realizado la implementación del estado, hemos tenido en cuenta que fuera lo más eficiente posible tanto desde un punto de vista espacial como temporal.

En primer lugar, antes de pensar en las estructuras de datos que implementaremos, hemos hecho hincapié en qué datos eran necesarios para representar el estado de forma correcta. Para cada furgoneta, necesitamos saber su estación inicial (E), las posibles 2 estaciones a las que puede ir a dejar bicis (P1 y P2), el número de bicis que deja en la primera (nP1) y en la segunda (nP2). Nótese que el número de bicis que

coge de la estación inicial ne se puede obtener sumando nP1 y nP2, por lo que no lo añadimos a la representación del estado.
Aparte de la información de las furgonetas, también necesitamos información sobre las estaciones. Para estas utilizamos la clase Estaciones que ya tenemos implementada. Como esta parte de la representación no cambia según el estado, será una estructura estática.
A continuación hemos pensado cuales eran los elementos que serían necesarios para el cálculo del heurístico y hemos decidido mantener actualizadas una serie de variables para no tenerlas que calcular en cada aplicación del heurístico. Mantener actualizadas estas variables no es costoso ya que cuando aplicamos un operador solo se modifica una parte del estado y nosotros sabemos qué variables del estado estamos modificando. Por lo tanto, hemos decidido guardar el número total de bicis bien colocadas, es decir el número de bicis que nos hace ganar 1€. De la misma forma, también guardamos el número de bicis mal colocadas, que nos hacen perder 1€. También guardamos el coste total de la gasolina que usaremos para calcular el segundo heurístico. Finalmente, hemos creado una clase Struct que guarda información dinámica sobre cada estación: para cada estación, podemos saber qué furgoneta la tiene como inicial, el número total de bicis que ha recibido y el número de bicis que le han sido sustraídas.

Una vez hemos pensado en toda la información que debe contener el estado, debemos decidir con qué estructuras de datos lo implementaremos. Debemos evitar tener información redundante y procurar que las estructuras sean eficientes tanto a la hora de acceder como de modificar. Hemos creado la clase Furgoneta con la información mencionada anteriormente y representado el estado con un vector de furgonetas. La estructura Estaciones ya nos viene implementada y las variables útiles son los otros atributos del estado. Para este fin hemos creído oportuno implementar la clase GeneraProblema, que es la que guarda las Estaciones del problema y la distancia entre ellas, para no tener que realizar el cálculo cada vez.

El espacio de búsqueda del problema son todas las posibles soluciones. Una solución es una configuración válida de viajes para cada furgoneta. Recordemos que cada furgoneta puede tener asociada o no una estación inicial y dos estaciones donde dejar bicis. Esto es O ($|E|^3$) para cada furgoneta.
Además, para cada la estación inicial cada furgoneta puede coger entre 1 y 30 (como máximo) bicicletas y distribuirlas por las otras dos estaciones de la siguiente manera: j bicis en la primera y 30 - j en la segunda, para una cierta j menor que el número de

bicicletas cogido de la estación inicial, que es menor que 30. Esto es O (30^2) para cada combinación furgoneta-estaciones.
Por tanto, el tamaño del espacio de búsqueda es O($F * E^3 * 30^2$) .

Operadores

La elección de los operadores es de suma importancia ya que a partir de ellos tenemos que poder explorar todo el espacio de soluciones posibles.

Hemos pensado en cómo podemos explorar todo el espacio de soluciones y hemos decidido utilizar los siguientes operadores que se aplicarán a cada una de las furgonetas:

1) cambiar estación inicial
2) modificar P1
3) modificar P2
4) traspasar k bicis de P1 a P2 (no se modifican las bicis que se cogen en E)
5) traspasar k bicis de P2 a P1 (no se modifican las bicis que se cogen en E)
6) cambiar número de bicis que se dejan en P1:
    - incrementar en 1 o en un número k de bicis
    - decrementar en 1 o en un número k de bicis
7) cambiar número de bicis que se dejan en P2:
    - incrementar en 1 o en un número k de bicis
    - decrementar en 1 o en un número k de bicis

Las condiciones de aplicabilidad son las siguientes:

1) No podemos permitir que dos furgonetas tengan la misma estación inicial
2) Siempre podemos modificar P1
3) Siempre podemos modificar P2
4) Podemos traspasar como máximo nP1 bicis a P2
5) Podemos traspasar como máximo nP2 bicis a P1
6) No podemos coger más de 30 bicis ni más bicis que las no usadas de la estación inicial
7) No podemos coger más de 30 bicis ni más bicis que las no usadas de la estación inicial

Los efectos de los operadores son los siguientes:

1) Puede existir una estación inicial en la que haya más bicis no usadas y por lo tanto tendremos más bicis disponibles para mover y obtener más beneficios
2) Puede haber otra estación en la que la demanda sea mayor y por lo tanto podemos dejar más bicis y obtener más beneficios
3) Efecto análogo al 2)
4) Puede que en P1 ya hayamos cubierto la demanda, en cambio en P2 la demanda todavía no esté cubierta
5) Efecto análogo al 4) cambiando P1 por P2 y vice-versa
6) Podemos coger más bicis de la estación inicial para generar más beneficios
7) Efecto análogo al 7) cambiando P1 por P2

El factor de ramificación de estos operadores es el siguiente:

factor_ramificación = $F*[(E-1)+(E-1)+(E-1) + 1 + 1 + 2 + 2] = F*(3(E-1) + 6)$

Hemos elegido los operadores de manera que podamos explorar todo el espacio de búsqueda del problema. Además, hemos intentado reducir el número de nodos expandidos añadiendo operadores que dependen de una constante k. En vez de ir moviendo las bicis una a una, permitimos que se muevan k a la vez para que se acelere la búsqueda.

Generación estado inicial

Hemos diseñado dos soluciones iniciales, una que básicamente es random y la otra que es en cierto modo greedy.

Para hacer una buena búsqueda local, no debe haber una solución demasiado buena, ya que si no impide que el algoritmo actúe puesto que ya estaremos en un máximo local. Nuestra hipótesis inicial es que el estado inicial greedy haría que el Hill Climbing fuera mejor, ya que lo dejamos cerca de una montaña y sólo tiene de acabar de subir. En cambio para el Simulated Annealing sería mejor el random, ya que como este algoritmo nos permite saltar si nos acercamos demasiado a un pico, otra montaña puede estar demasiado lejos y que no saltemos.

A continuación explicaremos el algoritmo greedy que usamos para generar el Estado Inicial 1:

- Empezamos creando un vector de tuplas. En cada tupla guardaremos el índice de la estación y la diferencia entre la previsión y la demanda (la llamaremos P-D).
- Hacemos un recorrido por todas las estaciones para llenar la tupla, tal y como hemos explicado.
- Ahora ordenamos la tupla en orden decreciente de P-D
- Ahora hacemos el mismo recorrido que en el algoritmo anterior. Lo único que varía es la selección de la estación inicial, de la estación P1 y de la estación P2. La estación inicial es la primera estación del vector una vez ordenado, es decir la estación en que la previsión es mucho mayor que la demanda. Y escogemos la estación P1 y la estación P2, como las dos que están más cerca de dicha estación.

El coste de este algoritmo es ((F+E)E)

Ahora explicaremos la solución inicial random, nosotros la hemos llamado Estado Inicial 2:

- Creamos el vector ve -> en este vector guardaremos los números del 0 al número de estaciones -1, que representan los índices de las estaciones. Esto lo hacemos ya que una estación no puede ser estación inicial de más de una furgoneta y así lo podemos controlar
- Creamos el vector vp -> en este vector guardamos hacemos lo mismo que en el vector ve pero por duplicado, es decir, guardamos dos veces el número 0 etc. Esto lo hacemos para asegurarnos que no haya una estación donde vayan muchas furgonetas inicialmente
- Inicializamos todas las estaciones, con los siguientes valores:
    - 0 bicis bien colocadas, 0 bicis mal colocadas, no está asignada a ninguna furgoneta
- Posteriormente hacemos un bucle, en cada iteración hacemos las siguientes acciones:
    - Cogemos un número aleatorio de entre 0 y la medida del vector ve y escogemos el valor que haya en esa posición como estación inicial. Eliminamos dicho valor de ve.
    - Hacemos lo mismo para vp, para la estación P1 y la estación P2
    - El número de bicicletas que cogemos de la estación inicial es el mínimo entre 0.6*bicicletas no usadas y 30

        - Dejamos la mitad de esas bicicletas en la estación P1 y la otra mitad en la estación P2
        - Inicializamos las estaciones con estos valores y creamos la furgoneta con estos valores
        - Finalmente, calculamos la gasolina que se gasta en hacer este recorrido
    - Este bucle lo hacemos el mínimo entre el número de furgonetas y el número de estaciones. Por tanto, cuando acaba dicho bucle comprobamos si aun quedan furgonetas que no se les ha asignada ninguna bicicleta, y se crean dichas furgonetas y se inicializan con los siguientes valores:
        - estación inicial=-1, estacionP1=-1, estacionP2=-1, número de bicicletas que se transportan =0
    - Finalmente se calculan las bicicletas bien colocadas de cada estación y las mal colocadas de cada furgoneta.

Este algoritmo tiene un coste (F+E), donde F es el número de furgonetas y E el número de estaciones.

Función heurística

La función heurística es fundamental ya que es la que nos permitirá explorar correctamente las soluciones del problema. Es la herramienta que nos va a guiar en la exploración del espacio de soluciones y debemos procurar que nos guíe de la mejor forma posible. Tenemos que diseñar una función que nos indique lo buena que es una solución de manera que si una es mejor que otra el valor del heurístico de la primera debe ser superior al de la segunda. En el caso del Hill-Climbing, la elección de la heurística es primordial ya que los problemas del algoritmo vienen derivados por la heurística que utilizamos. Antes de pensar en la función matemática que emplearemos para el heurístico, debemos pensar cuales son los factores que habrá que tener en cuenta.

En primer lugar nos centraremos en la primera heurística que se nos pide: tenemos que maximizar los beneficios sin tener en cuenta el coste de la gasolina. La primera aproximación nos sugirió utilizar la siguiente función:

h = BicisBienColocadas - BicisMalColocadas
donde BicisBienColocadas es el número de bicis que nos hacen ganar 1€
BicisMalColocadas es el número de bicis que nos hacen perder 1€

Si maximizamos este valor, a primera vista parece que podría funcionar pero si analizamos más a fondo la representación del problema que hemos escogido nos damos cuenta de que no nos guiará correctamente. En efecto, con esta heurística nos quedaremos encallados en el Hill-Climbing rápidamente. Esto se debe a los operadores que hemos elegido: una furgoneta maximizará los beneficios que puede obtener con su configuración actual pero cambiar de estación le aportará los mismos beneficios y por lo tanto no mejorará el valor de la heurística. En definitiva, esta heurística presenta demasiadas mesetas.
Para poner un ejemplo más práctico sobre este problema, vamos a suponer que tenemos una furgoneta situada en una estación inicial y que solo puede coger 3 bicis, ya que no hay más disponibles en la estación. Con esta heurística probablemente conseguirá colocar correctamente las 3 bicis, es decir generar 3€. Esto es así porque aplicando directamente los operadores de incrementar las bicis que se dejan en P1 y P2 los beneficios aumentarán. Ahora que hemos maximizado los beneficios que se pueden obtener desde esta estación inicial ya no hay ningún operador que pueda mejorar la solución. Por lo tanto esta furgoneta no escogerá una estación inicial en la

que pueda coger 30 bicis, que son mucho más prometedoras que las 3 actuales. En otras palabras, las furgonetas no cambiarán de estación.

Ahora que hemos visto que la heurística debe premiar que una furgoneta cambie su estación inicial por otra más prometedora ya podemos pensar en una función que lo contemple. Lo primero que haremos será mirar cuantas bicis podemos coger de la estación inicial de una furgoneta. Después miramos cuantas bicis faltan en P1 y P2 para saber cuántos beneficios podemos obtener como máximo. A este máximo de beneficio que podemos obter le substraeremos las bicicletas que ya hemos colocado, ya que ya no nos van a aportar ningún beneficio. Así, nos queda la siguiente fórmula.

$$h_{prometedores} = \sum_{f \in F} (min(disponibles(E),\ Demanda(P1 + P2)) - Colocadas(P1 + P2))$$

Por otro lado, experimentalmente se comprueba que con tener en cuenta la h presentada al principio de la sección y la $h_{prometedores}$ no es suficiente para incentivar que se muevan bicicletas, por lo que debe entrar en juego un factor que lo haga:

$$h_{colocadas} = \sum_{f \in F} Colocadas(P1 + P2)$$

Finalmente, nuestro heurístico 1 será una suma ponderada de los tres cálculos propuestos:

$$h_1 = h/k1 + h_{prometedores}/k2 + h_{colocadas}/k3$$

A partir de una serie de pruebas, observamos que para k1 = 1, k2 = 10, k3 = 5, los algoritmos se comportan bien, por lo que fijamos h1 a:

$$h_1 = h + h_{prometedores}/10 + h_{colocadas}/5$$

Para el heurístico que tiene en cuenta la gasolina (heurístico 2), tomamos como primera aproximación h2 = h1 - costeGasolina. Para el primer cálculo de h1 (h) esta aproximación no se traduce en un buen funcionamiento de los algoritmos. Sin embargo para la definitiva sí. Por tanto:

$$h_2 = h_1 - costeGasolina$$

Cabe aclarar que este cálculo supone un cambio importante con respecto al planteamiento inicial del problema, puesto que la implementación del estado mantenía cálculos que para facilitar la primera aproximación al cálculo de los heurísticos. Dado

que ésta es más compleja, sería necesario mantener una serie de cálculos más, concretamente los sumatorios necesarios para calcular $h_{prometedores}$ y $h_{colocadas}$. Pese a ser conscientes que el no mantener estos cálculos produce un perjucio importante en la eficiencia del programa, no hemos podido implementar ésta optimización por falta de tiempo. Por tanto, queda pendiente.
Sin embargo, para los experimentos realizados, los algoritmos no han tenido un tiempo de ejecución excesivo en ningún caso.

Experimentos

Antes de empezar a describir los experimentos daremos nombre a heurísticas y estados iniciales para facilitar las explicaciones:

- Estado inicial 1: Estado inicial greedy
- Estado inicial 2: Estado inicial aleatorio
- Función heurística 1: Heurística que maximiza los beneficios considerando que el transporte es gratuito
- Función heurística 2: Heurística que maximiza los beneficios teniendo en cuenta el coste del consumo de gasolina de las furgonetas

Experimento 1: Conjunto de operadores óptimos

El objetivo de este experimento es determinar qué conjunto de operadores da mejores resultados para la función heurística 1 usando el algoritmo de Hill Climbing. Para realizar este experimento hemos escogido el estado inicial 2 ya que es completamente aleatorio y hemos pensado que era el más adecuado para probar qué conjunto de operadores es mejor. De la otra forma hubiéramos encontrado qué operadores eran óptimos cuando la solución escogida era greedy, lo que no nos interesa en este experimento.

Escogemos una serie de conjuntos de operadores. Obviamente no hace falta probar todos los posibles subconjuntos de operadores ya que tenemos que asegurarnos que exploramos todo el espacio de soluciones. Por ejemplo si eliminamos el operador que cambia la estación inicial de una furgoneta será imposible explorar exhaustivamente el

espacio de soluciones. Por este motivo hemos decidido llamar Conjunto Base (CB) a los operadores: IntercambiarE, cambiarP1, cambiarP2. Este conjunto se tiene que combinar con algún operador que modifique las bicis que se transportan. Precisamente son estos operadores los que vamos a probar.
Los subconjuntos de operadores que hemos probado son los siguientes:

- C1: CB + cambiar Ns
- C2: CB + incrementar/decrementar 1 Ns
- C3: CB + incrementar/decrementar 1,2,7 Ns
- C4: CB + incrementar/decrementar 1,2,7,13 Ns
- C5: CB + incrementar/decrementar 1,2,3,5,7,11,13 Ns

Hemos realizado el experimento con 10 semillas distintas y para cada semilla hemos probado cada uno de los subconjuntos de operadores. Para poder visualizar mejor los resultados hemos presentado los resultados en forma de gráficos y son los que ahora vamos a analizar.

En el gráfico 1.1 podemos ver que los beneficios que obtenemos con todos los subconjuntos es exactamente el mismo. No obstante, vemos que en el caso de la semilla 5437 el subconjunto C1 obtiene menos beneficios que los otros 4. Haciendo pruebas más exhaustivas hemos visto que se trata de un caso marginal y es debido a la aleatoriedad del estado inicial 2, así que consideramos que todos los subconjuntos dan los mismos beneficios. Por lo tanto desde este punto de vista no podemos decidir qué subconjunto de operadores es mejor.

En el gráfico 1.2 podemos ver los nodos que se han expandido para cada subconjunto. Observamos que C1 es el que expande menos nodos mientras que C2 es por mucha diferencia el que tiene una expansión mayor, además de tener valores muy dispersos. Tanto C3, C4 como C5 tienen una expansión similar, no obstante es mayor que la de C1. Por lo tanto podemos concluir que C1 es el que expande menos nodos. Para asegurarnos que realmente C1 es el subconjunto que encuentra más rápido la solución hemos analizado también el número de operadores que se aplican así como el número de sucesores que se generan. Hemos pensado que era apropiado realizar este estudio

ya que el factor de ramificación es variable entre los subconjuntos y por lo tanto no podemos analizar sólo los nodos expandidos.

En los gráficos 1.3 y 1.4 vemos respectivamente los operadores usados y los sucesores generados por cada subconjunto. Los operadores usados son todos los operadores que se han intentado aplicar mientras que los sucesores son solo aquellos operadores que han satisfecho las condiciones de aplicabilidad. Observamos que la tendencia anterior se confirma, es decir que C1 genera menos sucesores que el resto de subconjuntos. Es interesante observar que la diferencia entre el número de operadores usados por C1 y los subconjuntos C3, C4 y C5 no es tan amplia como la diferencia entre sucesores. Por lo tanto concluimos que en C1 hay más operadores que no se aplican que en el resto de subconjuntos. Es decir, descartamos más operadores en C1 y de esta manera no generamos tantos sucesores. Esto es importante ya que el algoritmo Hill-Climbing tendrá que buscar el máximo de todos los sucesores y cuantos menos generemos mejor.

En conclusión, el mejor subconjunto de operadores es el C1.

Experimentando con esta heurística no podemos determinar si el operador *intercambiarP1P2 (*que, recordemos, cambia el orden de visita de P1 y P2) es o no útil. Como en no tenemos en cuenta el coste de la gasolina, nunca se va a aplicar este operador ya que no aporta nada. En cambio, si tenemos en cuenta la gasolina sí que es relevante. Vamos a ver si es útil disponer de este operador o si los operadores *modificarP1* y *modificarP2* simulan correctamente el intercambio de estaciones. Para ello hemos realizado un estudio análogo al anterior, pero con el otro heurístico.
Para este experimento:
CR = C1
CR2 = CR + *intercambiarP1P2*

En el gráfico 1.1bis vemos que los beneficios obtenidos tanto en P1 como en P2 son similares. Volvemos a ver que hay un caso en el un valor es ligeramente superior a otro pero se trata de un valor marginal.

En el gráfico 1.2bis vemos que los nodos expandidos también son similares.

Finalmente, en los gráficos 1.3bis y 1.4bis volvemos a ver que no podemos apreciar una diferencia entre los operadores usados y los sucesores generados.

Llegamos a la conclusión que el operador *intercambiarP1P2* no aporta nada a la resolución del problema, es decir es similar utilizar este operador que no hacerlo. Así, por simplicidad y eficiencia, no incluimos este operador en el subconjunto de operadores.

Experimento 2: Estrategia de generación de la solución inicial

El objetivo de este experimento es determinar qué solución inicial es mejor: la 1 o la 2. Ya hemos descrito exhaustivamente cómo se generan estos dos estados iniciales en un apartado previo. Para decidir cuál de las dos escogeremos realizaremos una serie de 10 experimentos con semillas diferentes. Para cada semilla veremos qué beneficios obtenemos con cada solución.

En el gráfico 2.1 podemos observar que la solución 1 siempre obtiene más o los mismos beneficios que la solución 2 por lo que desde el punto de vista de la calidad de la solución es mejor la 1.

Adicionalmente, hemos visto en un apartado anterior el coste de generar cada solución. El coste de la solución 1 es E veces más elevado que el de la solución 2, donde E es el número de estaciones del problema.

Visto que el número de estaciones no es muy elevado, hemos decidido quedarnos con la solución 1 ya que nos proporciona más beneficios.

Experimento 3: Parámetros del Simulated Annealing

El objetivo de este experimento es determinar cuales son los parámetros que dan mejor resultado para el Simulated Annealing (SA de ahora en adelante). El SA viene definido por 4 parámetros: el número máximo de iteraciones, número de iteraciones por paso de temperatura, parámetro k y el parámetro λ.
En primer lugar queremos estudiar k i λ por lo que fijaremos los dos primeros parámetros. Recordemos el SA se basa en aceptar con una cierta probabilidad sucesores peores que el actual. Esta probabilidad es variable, al principio, cuando la temperatura es elevada es muy probable escoger un sucesor peor y a medida que disminuye la temperatura también disminuye la probabilidad hasta llegar a 0. Cuando llega a 0 nos tenemos que asegurar que sigan habiendo iteraciones para poder alcanzar el máximo relativo.
El parámetro k es el que determina cuándo va a comenzar a decrecer la función de aceptación. Cuanto mayor es k más tarda en comenzar a decrecer.
El parámetro λ es el que determina lo rápido que va a decrecer la función de aceptación.

Para ello usaremos la función heurística 1, los operadores C1 y la solución inicial 1. Como se trata de un experimento en el que vamos relativamente a ciegas, hemos decidido probar una gran cantidad de valores para cada semilla. En los gráficos se ve un tercer elemento que es el color. Cuanto más rojo es el color más frecuente es el valor que representa. Por tanto, intuitivamente, buscamos los valores de número de iteraciones por temperatura, k y λ que tengan más color rojo en la parte superior del gráfico.

En el gráfico 3.1 vemos los beneficios obtenidos en función de k. Observamos que las combinaciones más frecuentes se encuentran para los valores pequeños de k, entre 1 y 5. Además estos valores son los que proporcionan más beneficios. Concluimos que el valor óptimo de k se sitúa entre 1 i 5.

En el gráfico 3.2 vemos los beneficios obtenidos en función de log(λ). Observamos que la combinación con más beneficios es en la que log(λ) = -1. Por lo tanto el valor óptimo de λ es 0.1.

En el gráfico 3.3 se puede observar que el número de iteraciones no tiene especial influencia, aunque parece que se comporta ligeramente mejor para valores inferiores o iguales a 10.

Por último, observando los datos directamente, una de las combinaciones que da en promedio mejor beneficio es: número de iteraciones por temperatura = 10, k = 1 y λ = 0.1. Dado que se ajusta a las conclusiones que hemos sacado de los gráficos, elegimos estos parámetros para la ejecución del SA de ahora en adelante.

Experimento 4: Tiempo de ejecución

El objetivo de este experimento es estudiar la evolución del tiempo de ejecución en función del número de estaciones, furgonetas y bicis.

En la gráfica 4.1 observamos el tiempo de ejecución del problema en función de las estaciones. Constatamos que el tiempo de ejecución es lineal pero que a partir de este número aumenta de forma exponencial respecto al tamaño del problema. Este comportamiento es similar en todas las semillas estudiadas así que podemos concluir que el algoritmo tiene coste exponencial.

Experimento 5: Hill Climbing vs Simulated Annealing

El objetivo de este experimento es comparar los resultados que se obtienen utilizando Hill Climbing (HC) y Simulated Annealing (SA). Para todos los experimentos utilizaremos 10 semillas y probaremos las dos heurísticas.

La primera variable que estudiaremos son los beneficios. En el gráfico 5.1 vemos los beneficios que se obtienen en HC y en SA con la heurística 1. En la mayoría de semillas los beneficios son idénticos con ambos algoritmos y en algunos HC es ligeramente mejor. El gráfico 5.2 nos muestra lo mismo que el anterior pero esta vez con la heurística 2. En este vemos claramente que SA da sistemáticamente mejores resultados que HC. En algunos casos esta diferencia es pequeña pero en otros es

considerable. Sin ninguna duda, SA se comporta mejor que HC con la segunda heurística.

Ahora vamos a estudiar la distancia total recorrida. En el gráfico 5.3 vemos la distancia total recorrida que se obtiene en HC y SA con la heurística 1. Vemos de forma rápida que la distancia total recorrido por SA es siempre menor que en HC y en algunos casos es notable. El gráfico 5.4 nos muestra el mismo estudio con la heurística 2 y aquí volvemos a apreciar el fenómeno anterior: la distancia total de SA es menor que la del HC. Podemos concluir que SA minimiza las distancias mejor que HC con ambas heurísticas.

La siguiente variable es el coste de gasolina. Pese no estar en el enunciado del experimento, hemos considerado que tiene mucho más sentido estudiar el coste de la gasolina que la distancia, ya que la fórmula de cálculo de ésta y su dependencia del número de bicicletas transportadas hace que no sean proporcionales y, en definitiva, lo que produce coste es la gasolina y no la distancia. El gráfico 5.5 representa el coste de la gasolina en HC y SA con la heurística 1. La interpretación de este gráfico es más complicada que los anteriores ya que no hay un algoritmo que sea siempre mejor que otro. En algunos casos HC se comporta mejor que SA y en otros al revés. No obstante en ciertos casos el gasto de HC es mucho mayor que SA mientras que cuando SA es mayor que HC lo es solo ligeramente. De momento no realizaremos ninguna afirmación al respecto. Vamos a ver qué ocurre con la heurística 2. El gráfico 5.6 representa lo mismo que el anterior pero utilizando la heurística 2. Aquí la tendencia es similar, es decir que no hay un algoritmo que se comporta mejor que el otro. En algunos casos la diferencia es mínima y en otros es sustancialmente mayor. Por lo tanto no afirmaremos nada ya que no tenemos datos suficientes para realizar una conclusión. De hecho no podemos estar seguros de que haya un algoritmo que se comporte mejor que el otro.

La siguiente variable que hemos querido estudiar son los beneficios reales de cada algoritmo. Definimos beneficios reales como los beneficios obtenidos por la colocación de bicis menos el coste de la gasolina. El gráfico 5.7 muestra los beneficios reales de ambos algoritmos con la heurística 2. Observamos que en la mayoría de casos SA se comporta mejor que HC. En 2 de los 10 casos HC lo hace mejor que SA, por lo que tendríamos que estudiar si estos casos son marginales o hay una proporción de casos en los que HC funciona mejor que SA.

La última variable estudiada es el tiempo de ejecución. El gráfico 5.8 muestra el tiempo de ejecución de HC y SA con la heurística 1. Observamos que en todos los casos SA es más lento que HC, en algunos casos más del doble. Este resultado es esperado ya

que SA se basa en explorar un espacio más grande que HC. Tendríamos que valorar si el tiempo adicional empleado compensa la mejor calidad de solución que ofrece SA. Es interesante observar la gráfica 5.9, similar a la anterior pero con la heurística 2. Aquí la diferencia de tiempo entre SA y HC no está nada clara, los niveles son muy similares.

Experimento 6: Estudio hora punta

El objetivo de este experimento es estudiar la influencia de la demanda. La demanda puede ser equilibrada o hora punta. Todos los experimentos anteriores han sido realizados con demanda equilibrada y ahora nos vamos a interesar por la hora punta. Este tipo de demanda se caracteriza por haber unas pocas estaciones que tienen mucha demanda. Vamos a estudiar si hay alguna diferencia en la ejecución del mismo problema con demandas distintas. Analizaremos los beneficios obtenidos y el tiempo de ejecución. Utilizaremos el SA ya que es el algoritmo que mejores resultados nos ha dado.

La gráfica 6.1 muestra los beneficios obtenidos para cada tipo de demanda. Vemos que obtenemos más beneficios con la demanda en hora punta. Este se podría deber a que como hay mucha demanda en pocas estaciones se pueden llevar todas las bicis que sobran a estas estaciones y de esta forma se obtienen fácilmente beneficios, mientras que la demanda equilibrada puede ser más complicado encontrar estaciones donde sobren muchas bicis o falten muchas bicis.

La gráfica 6.2 muestra el tiempo de ejecución para cada tipo de demanda. El tiempo de ejecución es variable en ambos pero generalmente es parecido. Podemos concluir que parece que el tipo de demanda no afecta en el tiempo de ejecución del problema.

Experimento 7: Estudio de las furgonetas

El objetivo de este experimento es estudiar de qué manera afecta el número de furgonetas en los beneficios obtenidos. Usaremos la heurística 2 y, como siempre, probaremos con 10 semillas distintas. Conjeturamos que cuantas más furgonetas haya más beneficios podremos obtener hasta un punto en el que no podamos colocar más

bicis correctamente y se alcance un máximo de beneficios. Es decir que añadir furgonetas no va a producir ningún efecto en los beneficios ya que o bien todas las estaciones están ocupadas o ya no sobran bicis de ninguna estación inicial.

En primer lugar hemos estudiado la demanda equilibrada. En la gráfica 7.1 podemos ver los beneficios en función de las furgonetas. Vemos que cuando el número de furgonetas está entre 0 y 7 los beneficios crecen linealmente respecto al número de furgonetas. No obstante, a partir de un valor de furgonetas, aproximadamente 7, se alcanza un máximo y los beneficios dejan de aumentar. Vemos que los beneficios oscilan alrededor de una recta, que es el máximo beneficio que se puede obtener en cada uno de los escenarios posibles.

De forma análoga hemos estudiado la demanda en hora punta. En la gráfica 7.2 observamos los beneficios en función de las furgonetas. Se produce el mismo fenómeno que en el caso de demanda equilibrada: la relación entre furgonetas y beneficios es lineal al principio y después se vuelve constante ya que alcanzamos un máximo. El número de furgonetas que se necesita para alcanzar el máximo varía para cada semilla pero se sitúa entre 5 y 10 furgonetas.

El número de furgonetas para los dos escenarios es similar. Como en los dos hay el mismo número de bicis y de estaciones, es previsible que el número de furgonetas necesarias para alcanzar el máximo sea próximo.

Comparación entre Hill Climbing y Simulated Annealing

A lo largo de estos experimentos hemos comprobado las similitudes y diferencias de ambos algoritmos.
Hemos visto que para que SA funcione hace falta encontrar los parámetros óptimos y que puede ser difícil encontrarlos ya que requiere mucha experimentación. En cambio HC no requiere ningún ajuste de parámetros y por lo tanto es más rápido de implementar y se obtienen resultados aceptables.
Desde un punto de vista temporal HC se comporta mejor que SA aunque este último permite encontrar soluciones mejores cuando la situación es más completa.
Cuando usemos SA, siempre tendremos que valorar si compensa tener una solución mejor sacrificando un poco más de tiempo de ejecución.