UNIVERSITAT POLITÈCNICA DE CATALUNYA

INTELIGENCIA ARTIFICIAL

Práctica de búsqueda local: Bicing

Rubén Catalán, Ismael El Basli y Alex Garcés

Q1 2023-24

Índice

1. Descripción del problema

En este proyecto, se presenta el problema de la correcta distribución de las bicicletas del servicio urbano Bicing en sus diferentes estaciones. Para ello, Bicing hará uso de información recolectada acerca de las diferentes estaciones así como un conjunto de furgonetas para realizar la distribución.

La información que se aporta de cada estación y se utilizará para guiar la solución es la siguiente:

- Previsión de bicicletas no utilizadas durante esa hora (pueden ser movidas sin afectar a los usuarios que quieran utilizar bicicletas en esa estación)
- Bicicletas que serán dejadas por los usuarios en la hora siguiente (independiente de aquellas que movamos mediante furgonetas)
- Previsión de la demanda de la estación durante la siguiente hora (guiará el número de bicicletas que dejamos)

1.1. Elementos del problema

Se considera la ciudad como un cuadrado de 10x10 kilómetros donde cada manzana tiene un tamaño de 100x100 metros.

En el problema simplificado, trabajamos con un conjunto de E estaciones y B bicicletas constante. Una estación puede guardar un número ilimitado de furgonetas, y estas estaciones se encuentran en los cruces entre calles de las diferentes manzanas. Por ende, la distancia entre estas nos la da la función:

$$d(i,j) = |i_x - j_x| + |i_y - j_y|$$

Se tienen F furgonetas para realizar el transporte de bicicletas entre estaciones. Por cada hora, cada una de las furgonetas se puede mover a uno, dos, o cero destinos.

Las furgonetas pueden cargar entre 0 y 30 bicicletas, y se pagará un euro por cada bicicleta transportada que acerque la estación a la demanda. A su vez, si nos llevamos una bicicleta de una estación tal que nos alejamos de la demanda prevista, se nos cobrará un euro.

Por último, se debe considerar también la distancia de los viajes, que supondrá otro coste en la solución. Sea n el número de bicicletas en una sola furgoneta, el coste de transporte se calcula como:

$$((n + 9)\ div\ 10)$$

Sabiendo esto, el objetivo es claro: buscar los mejores orígenes, destinos y cargas de cada furgoneta para maximizar los beneficios de la solución y/o minimizar los costes.

1.2. Criterios de la solución

En búsqueda de las mejores soluciones, ese necesario preocuparse por varios componentes del problema:

- Límite de capacidad de furgonetas (30 bicis)
- Entre 0 y 2 estaciones visitadas por furgoneta
- La furgoneta carga bicis *solo* del origen, y no puede haber más de una furgoneta partiendo de la misma estación
- Más de una furgoneta puede dejar bicis en una misma estación
- Dos criterios:
    - Maximización: de los beneficios por traslado
    - Minimización: del coste por transporte

1.3. Justificación del uso de búsqueda local

El problema que se plantea es complejo (el espacio de búsqueda es muy grande) y buscamos maximizar y minimizar ciertos criterios, conformándonos con buenas soluciones sin necesidad de obtener el óptimo global. Por este motivo, es correcto usar algoritmos de búsqueda local, podemos plantear la resolución del problema como la mejora de una solución inicial. Pensar en resolverlo con búsqueda heurística (por ejemplo usando A*) no es buena idea ya que, como se ha dicho, en ningún momento se busca el óptimo global y los algoritmos de búsqueda heurística se encuentran en el espectro de la alta complejidad.

2. Implementación

2.1. Implementación del estado

En este proyecto, se ha decidido representar el estado del problema como una asignación de origen, destinos y bicis dejadas en cada destino a cada una de las furgonetas disponibles. Es decir, podemos tener un estado por cada una de las asignaciones posibles. Para ello, y teniendo en cuenta que la representación debe ser eficiente en espacio y tiempo, se ha optado por usar las siguientes estructuras de datos:

- Una matriz de enteros **furgonetas[nfurgs][5]**. El tamaño de la matriz es de nfurgs $\times$ 5, donde nfurgs = máx (número de furgonetas, número de estaciones). Esto es debido a que si se nos plantea un escenario con mas furgonetas que estaciones, hay ciertas furgonetas que no se pueden usar (ya que como mucho una puede partir de un origen concreto), por lo que no conviene tenerlas en cuenta en la representación. Para cada fila de la matriz hay 5 columnas, que corresponden al origen, destino 1, bicis dejadas en el destino 1, destino 2 y bicis dejadas en el destino 2 para la furgoneta que corresponde a dicha fila. Se han usado variables *static* para poder acceder usando *furgonetas*[0][ORIGEN] en vez de *furgonetas*[0][0], para que el código sea más fácil de entender. Además, por convenio, cuando una furgoneta tiene destino2 = -1, no tiene destino 2, y cuando tiene destino1 = -1, la furgoneta no se usa en la solución.

- Un *array* de booleanos **origen_ocupado[nestaciones]**, con tantos elementos como número de estaciones. La posición i-ésima del *array* tiene el valor *true* si hay una furgoneta que ya recoge bicicletas en la estación i-ésima, *false* en caso contrario. Esto resulta imprescindible para poder controlar la aplicabilidad del operador cambiar origen que se describe más adelante.

- Un *array* de enteros **bicis_dejadas[nestaciones]**, con tantos elementos como número de estaciones. La posición i-ésima del *array* tiene el número de bicis dejadas en la estación i-ésima. Se ha optado por incluir esto en la representación, ya que, como se vuelve a mencionar más adelante, no se dispone de ningún operador que varíe el número de bicis que se dejan en cada estación del trayecto. En todo momento se dejarán en el destino 1 las que hagan falta, y las que sobran en el destino 2. Como que varias furgonetas pueden dejar bicicletas en una misma estación, conviene saber las que se han dejado en cada una.

- Un objeto *static* **estaciones**. Se trata de la estructura que contiene la información de todas las estaciones del problema. Al ser *static*, se comparte por todos los objetos de la clase, de forma que se copia una única vez.

Se ve claramente como la lógica que se ha seguido para decidir qué estructuras de datos usar es el ahorro de tiempo y espacio. Inicialmente se planteó hacer un objeto furgoneta y tener un vector de ese objeto, pero es obvio que usando una matriz ahorramos en tiempo y espacio. Por otro lado, información relativa a si un origen ya está ocupado o el número de bicis que se han dejado en una estación, es algo que se

consulta constantemente cada vez que se quiere aplicar cierto operador, por lo que se ha considerado que el espacio extra que supone guardar esa información en un *array* es compensado por el ahorro temporal que se obtiene. Si no se usaran estos dos últimos *arrays*, cada vez que se quisiera cambiar el origen a una estación, se debería recorrer todas las furgonetas para ver si alguna de ellas tiene cómo origen el origen nuevo que se quiere asignar a la furgoneta, cosa que tiene coste $\mathcal{O}(nfurgs)$. Por otro lado, cada vez que se quisiera determinar cuántas bicis se pueden dejar en el destino 2, para saber las que se pueden quitar del destino 1, de deberían recorrer todas las furgonetas para ver el total de bicis dejadas en la estación que corresponde al destino 1, para saber cuántas se pueden quitar realmente, el coste seria también de $\mathcal{O}(nfurgs)$.

Por último, por lo que hace al tamaño del espacio de búsqueda que se deduce de la implementación planteada: tenemos $nfurg$ furgonetas donde $nfurg =$ máx(numero de furgonetas, número de estaciones), y para cada furgoneta tenemos $\mathcal{O}(nest \times nest \times nest) = \mathcal{O}(nest^3)$ posibilidades, ya que tenemos como orígenes posibles todas las estaciones, de la misma forma que pasa con los destinos 1 y 2. Por lo tanto, es fácil ver que el espacio de búsqueda es de $\mathcal{O}((nest^3)^{nfurg})$. Se puede ver que en ningún momento se tiene en cuenta el número de bicis dejadas en el espacio de búsqueda, ya que siempre se dejan, poniendo como límite las que se pueden coger del origen, todas las que hagan falta en el destino 1, y las que sobren en el destino 2.

2.2. Operadores

Después de barajar varias opciones, se ha optado por la implementación de 3 operadores:

1. **cambiar_destino1(ifurg, iest)**: Le asigna la estación iest como destino 1 a la furgoneta ifurg. Este operador no tiene condición de aplicabilidad, si iest coincide con el origen de la furgoneta, la furgoneta pasa a no hacer nada. Por otro lado, si iest coincide con el destino 2 de la furgoneta, la furgoneta pasa a tener únicamente destino 1 iest. Se puede ver como este operador resulta ser muy flexible, ya que aparte de asignar un nuevo destino1, permite que una furgoneta pase a no ser usada o que pase a tener sólo destino 1. Puede haber soluciones donde ciertas furgonetas no se usen. En cuanto al factor de ramificación del operador, es de $\theta(nfurg \times nest)$, ya que para cada furgoneta le podemos cambiar el destino 1 a cualquiera de las estaciones disponibles.

2. **cambiar_destino2(ifurg, iest)**: Le asigna la estación iest como destino 2 a la furgoneta ifurg. En este caso, sí que hay condición de aplicabilidad: la furgoneta ifurg debe tener un destino 1 asignado, y el destino 1 no debe coincidir con iest. En caso de que el iest coincida con el destino 1, la furgoneta pasa a no tener destino 2. El factor de ramificación de este operador, coincide con el de cambiar_destino1, es $\theta(nfurg \times nest)$.

3. **cambiar_origen(ifurg, iest)**: Le asigna la estación iest como origen a la furgoneta ifurg. Como condición de aplicabilidad: no hay ninguna furgoneta

que ya tenga como origen iest, y iest no debe coincidir ni con el destino 1 ni con el destino 2 de la furgoneta ifurg, en caso que existan. En este punto es donde se saca partido al *array* origen_ocupado, que nos permite consultar en $\mathcal{O}(1)$ si un origen ya se encuentra ocupado por una furgoneta. Igual que en los operadores anteriores, el factor de ramificación es de $\theta(nfurg \times nest)$, ya que para cada posible furgoneta, tenemos tantos orígenes posibles como el número de estaciones.

Se ha decidido usar estos operadores porque no generan no-soluciones y permiten explorar todo el espacio de soluciones si se usan juntos. Cabe mencionar que se probó de usar más operadores como *swap* de destinos (tanto entre furgonetas como destinos de una misma furgoneta), pero la mejora era ínfima en relación al incremento de la ramificación, además que tienen sólo sentido cuando en la heurística usada se tenga en cuenta el coste de la distancia recorrida. Por otro lado, también se pensó en añadir un operador que modifique el número de bicis dejadas en los destinos 1 y 2, pero se consideró también que eso suponía un aumento considerable de la ramificación, y que la estrategia que se usa a la hora de dejar bicicletas ya permite llegar a buenas soluciones: cuando se aplica cualquier de los tres operadores mencionados antes, se recalcula las bicis que se dejarían en el destino 1 y el destino 2. Se calcula el máximo número de bicis que se pueden coger del origen, cosa que coincide con max(0, min(bicis no usadas, bicis que habrá a la siguiente hora - demanda)). Partiendo de aquí, se dejarán en el destino 1 todas las bicis que necesite el destino 1 para cubrir su demanda, poniendo como cota superior el número de bicis que se pueden coger del origen. Si hay destino 2, de las bicis que quedan por coger del origen, se cogen también las que hagan falta.

2.3. Funciones heurísticas

El factor principal que interviene en la heurística del problema es el beneficio obtenido. Este beneficio depende principalmente de:

- Bicicletas transportadas que hagan que el número de bicicletas de una estación se acerque a la demanda prevista, Bicing nos paga un euro por cada una de esas bicicletas.
- Bicicletas transportadas que hagan alejarse a una estación de su previsión, Bicing nos cobra un euro por cada una de esas bicicletas.
- El transporte de las bicicletas, Bicing nos cobra $((nb + 9)div10)$ euros por kilómetro, donde nb es el número de bicicletas que se llevan en la furgoneta y div es la división entera.

Es decir, para evaluar la calidad de una solución nos podemos fijar tanto en la maximización de lo que obtenemos por los traslados de las bicicletas como la minimización del coste del transporte. De aquí salen los dos criterios que corresponden a las dos heurísticas implementadas. Cabe recordar que los algoritmos de búsqueda local implementados en las clases del *AIMA* buscan mínimos, por lo que una solución con un valor heurístico $h1$ será escogida antes que una con un valor de $h2$ si y solo si $h1 < h2$.

2.3.1. Primer criterio

Con el primer criterio se ha implementado la función heurística 1. Este criterio ignora el coste del transporte de las bicicletas, solo considera las ganancias por acercarse a la demanda prevista o los costes por alejarse de la previsión. Está claro que el efecto que tiene esta función en la búsqueda es guiarla hacia una solución donde, independientemente de los kilómetros recorridos, se maximice el número de bicicletas dejadas que hagan acercarse a la demanda prevista en ciertas estaciones. También cabe destacar que nunca se permite alejar a una estación de su previsión ya que, como se ha mencionado en la estrategia a la hora de dejar bicis, nunca se cogen más de las que es posible coger del origen. Esto es debido a que en el momento en que se coge una bicicleta que nos hace alejarnos de la previsión ya se está perdiendo un euro, y como mucho se puede recuperar si se deja la bicicleta en una estación donde hace falta. Por lo tanto, si se considera que el transporte es gratis, como mucho nos quedamos igual, y si costara dinero, nos quedaríamos igual o peor. Por el motivo mencionado previamente (los algoritmos consideran mejor solución la que tiene un valor heurístico menor) y considerando que $n1$ =número de bicicletas que hacen acercarnos a la demanda y $n2$ =número de bicicletas que hacen alejarnos de la previsión, las ponderaciones que aparecen en los elementos de la heurística son: $-1 \times n1 + 1 \times n2$.

2.3.2. Segundo criterio

Con el segundo criterio se ha implementado la función heurística 2. Este criterio tiene en cuenta lo mismo que el primero, pero sumando el coste del transporte. Si el coste de transporte es ct, ahora las ponderaciones son: $-1 \times n1 + 1 \times n2 + 1 \times ct$. El efecto que tiene esta segunda función heurística en la búsqueda es guiarla hacia una solución que maximice el beneficio teniendo en cuenta todos los aspectos que pueden hacer ganar o perder dinero. Por lo tanto, no se fija únicamente en intentar llevar todas las bicicletas que se puedan, ya que el coste transporte interviene en la búsqueda.

2.4. Estrategias para hallar la solución inicial

Para hallar la solución inicial se han implementado tres estrategias:

- **Estrategia trivial**: Se asigna un origen a cada una de las furgonetas, siguiendo el orden por defecto en el que se encuentran las estaciones. No se asigna ningún destino, es decir, las furgonetas no hacen nada y el beneficio inicial es de cero euros. Por lo tanto, el coste de generar esta solución es lineal respecto al numero de furgonetas del estado (recordar que pueden ser menos que las del problema planteado): $\mathcal{O}(nfurgs)$.

- **Estrategia aleatoria**: Se asignan los orígenes a las furgonetas de la misma forma que en la estrategia anterior, pero una vez asignados se hace un *shuffle* de los orígenes. De esta forma, se puede decir que los orígenes se escogen de forma aleatoria. Después de esto, también se escoge de forma aleatoria si cada furgoneta va a tener o no destino 1, en caso que si, éste se asigna de forma

aleatoria (no debe coincidir con el origen). Si se da el caso que la furgoneta tiene destino 1 asignado, entonces también se decide de forma aleatoria si va a tener o no destino 2, y si lo va a tener, este se vuelve escoger de la misma forma que el destino 1 (el destino 2 no debe coincidir ni con el origen ni con el destino 1). Por lo tanto, el coste de generar esta solución vuelve a ser lineal respecto al número de furgonetas del estado: $\mathcal{O}(nfurgs)$.

- **Estrategia voraz**: Se ordena el vector de estaciones de forma que las primeras del vector son aquellas donde se pueden coger más bicicletas (sobran más), y las que aparecen al final son las que necesitan mas bicicletas para llegar a cubrir su demanda. Una vez ordenado, se asignan los orígenes a las furgonetas igual que en la estrategia trivial, pero ahora las furgonetas parten de los "mejores" orígenes. Además, a medida que se hace este recorrido se asigna como destino 1 las estaciones mas necesitadas, que corresponden a las que se encuentran al final del vector. Por lo tanto, el coste de generar esta solución es de $\mathcal{O}(nest \log(nest))$, debido a la ordenación. Está claro que está es la solución que da un mejor resultado inicial si se considera que el transporte es gratuito. En la experimentación se verá si será la que permite llegar a una mejor solución o no.

3. Experimentación

3.1. Experimento 1: Influencia de los operadores para Hill Climbing

Observación	Pueden existir conjuntos de operadores que den mayor beneficio que otros
Planteamiento	Se escogen distintos conjuntos de operadores y se observa el beneficio obtenido para cada uno
Hipótesis	Todos los conjuntos de operadores dan el mismo beneficio (H0) o hay conjuntos que dan mayor beneficio
Método	▪ Se eligen 10 semillas aleatorias. ▪ Para cada semilla se ejecutan los conjuntos de operadores elegidos. ▪ El escenario para todas las ejecuciones contendrá 25 estaciones, 1250 bicicletas, 5 furgonetas, la demanda será equilibrada y la solución inicial será la trivial. ▪ El algoritmo de búsqueda es Hill Climbing.

Se han seleccionado como conjuntos de operadores para el experimento los siguientes:

1. $\{cambiarDestino1, cambiarDestino2\}$
2. $\{cambiarDestino1, cambiarOrigen\}$
3. $\{cambiarDestino1, cambiarDestino2, cambiarOrigen\}$

Han sido elegidos estos conjuntos ya que, en el caso de escoger como solución inicial la trivial, los operadores $cambiarDestino2$ y $cambiarOrigen$ no aportan ningún beneficio a no ser que se ejecute el operador $cambiarDestino1$. El operador $cambiarDestino2$, por su condición de aplicabilidad, no se ejecutará a menos que ya exista un primer destino (la solución inicial solo escoge el origen de las furgoneta). El operador $cambiarOrigen$ no permitirá obtener beneficios ya que, por mucho que se cambien los orígenes de las furgonetas, éstas seguirán sin ir a ningún destino. Por tanto se han omitido aquellos conjuntos que no contienen el operador $cambiarDestino1$. También se ha omitido el conjunto que contiene solo el operador $cambiarDestino1$ ya que para casos en que haya mas estaciones que furgonetas habrá estaciones que serán mas beneficiosas que otras y será necesario que las furgonetas tengan mas de un destino para poder abastecer de bicicletas a la mayor cantidad de estaciones.

	Beneficio			Nodos Expandidos			Tiempo (ms)		
semilla	D1+D2	D1+O	Todos	D1+D2	D1+O	Todos	D1+D2	D1+O	Todos
93458	7	25	25	2	4	4	13	17	17
20655	19	54	56	3	6	7	14	19	22
5661	10	36	36	3	6	6	14	19	20
29035	30	30	30	2	2	2	13	14	15
71827	18	19	19	2	3	3	13	18	16
46406	7	38	47	4	10	11	15	22	25
10182	40	48	48	3	5	5	14	18	20
63971	48	55	55	5	7	7	17	19	23
11064	12	38	38	4	5	5	15	18	19
5709	14	25	25	2	3	3	13	16	16

Tabla 1: Beneficio, nodos expandidos y tiempo de ejecución en función del conjunto de operadores

Una vez se han tomado los datos se puede observar en la tabla 1 que en la mayoría de pruebas tanto el conjunto que utiliza $cambiarDestino1$ y $cambiarOrigen$ como el conjunto de todos los operadores dan un mayor, y el mismo, beneficio que el conjunto $cambiarDestino1$ y $cambiarDestino2$. Aun así, se puede observar que en algunas ocasiones el conjunto de todos los operadores otorga un mayor beneficio que los otros dos conjuntos. Ya que búsqueda local se usa para optimizar soluciones se ha optado por elegir el conjunto de todos los operadores por esa posibilidad observada de conseguir una mejor solución que los demás conjuntos.

3.2. Experimento 2: Influencia de la solución inicial

Observación	Pueden existir métodos de inicialización que llevan a mejores soluciones
Planteamiento	Escogemos diferentes métodos de inicialización y observamos los resultados obtenidos
Hipótesis	Todos los métodos de inicialización llevan a las mismas soluciones (H0) o hay métodos mejores
Método	▪ Se escogen 10 semillas aleatorias, una por réplica ▪ Para cada semilla, se ejecuta 1 experimento en la inicialización Trivial y Voraz ▪ Para la inicialización aleatoria, se ejecutaran 5 experimentos por semilla y se calcula la media de los resultados ▪ El escenario será el mismo que en el experimento 1, fijando los operadores D1+D2+CO ▪ Usaremos Hill Climbing como algoritmo de búsqueda

	Beneficio			Nodos Expandidos			Tiempo (ms)		
semilla	Trivial	Aleatoria	Voraz	Trivial	Aleatoria	Voraz	Trivial	Aleatoria	Voraz
12500	35	35	69	6	6	5	17	17	17
23129	34	50	76	4	7	5	15	18	19
4128	54	58	84	10	5	4	20	17	16
45342	16	44	68	3	5	3	14	18	15
92214	21	39	60	4	5	3	15	16	15
9846	51	67	92	5	5	6	15	28	18
1743	41	53	80	6	10	4	17	17	16
75632	70	84	99	13	13	3	23	24	15
82878	64	75	75	12	12	5	28	23	17
40057	0	23	85	1	5	4	11	16	15

Tabla 2: Beneficio, nodos expandidos y tiempo de ejecución en función de la solución inicial

De estos resultados, podemos observar en la tabla 2 que para todas las semillas estudiadas, se consigue una sustancial mejora de beneficio con la inicialización Voraz, que escoge destinos y orígenes de forma informada. Por estas razones, trabajaremos con la inicialización Voraz a partir de ahora.

3.3. Experimento 3: Estudio de los parámetros usados en el Simulated Annealing

Una tarea que hay que realizar es encontrar los parámetros adecuados para ejecutar el Simulated Annealing. La implementación del algoritmo tiene los siguientes parámetros:

- Número total de iteraciones
- Iteraciones por cada cambio de temperatura (ha de ser un divisor del anterior)
- Parámetro k de la función de aceptación de estados
- Parámetro λ de la función de aceptación de estados

Los mejores valores de estos parámetros se pueden deducir experimentalmente. Para ello, se usa un número de iteraciones adecuado para asegurar que se llega a converger.

Observación	El valor de los parámetros k y λ influyen en la solución a la que llega el Simulated Annealing. Algunos parámetros pueden llevar a mejores soluciones.
Planteamiento	Elegimos distintas combinaciones para valores de k y λ y observamos los resultados obtenidos con dichos valores.
Hipótesis	En las combinaciones con valores de k muy alto se obtienen los peores resultados
Método	- Se escogen 5 semillas aleatorias. - Para cada semilla, se ejecutan 15 experimentos para toda combinación posible con los valores $k = \{1, 5, 25, 125\}$ y para $\lambda = \{10, 1, 0.01\}$. - Se hacen las medias de los resultados obtenidos en cada combinación. - Se comparan los resultados obtenidos. - El escenario será el mismo que en el experimento 1, fijando los operadores D1+D2+CO y la estrategia voraz como estrategia para encontrar la solución inicial.

En la tabla 3 se pueden ver los resultados. Se puede ver como en los mejores resultados el valor de λ es igual a 0.01. Además, se ve como los peores resultados se dan cuando la k y la λ toman los valores mas grandes. Se deduce entonces, que para este problema concreto y el escenario escogido, una muy buena combinación a usar es $k = 1$ y $\lambda = 0.01$. Cabe mencionar que también se probó con valores más pequeños de λ pero se llegaba a resultados iguales que a los que se llega usando $\lambda = 0.01$. De todos modos quisimos, a pesar de que el experimento no lo contempla, probar con

la heurística 2. Vimos como se notaba una pequeña mejora usando $\lambda = 0.0001$. Por este motivo, se llega a la conclusión de que la mejor combinación que se puede usar es $k = 1$ y $\lambda = 0.0001$. El número de iteraciones usado, como se mencionó previamente, ha sido el suficiente para asegurar que se pueda llegar a converger, concretamente ha sido de 5000 iteraciones y 100 iteraciones por cambio de temperatura. Este valor se debe ajustar dependiendo del tipo de escenario, y siempre se debe tener en cuenta que a veces es mejor quedarnos con soluciones peores si el tiempo para encontrar una solución un poco mejor es excesivo.

Beneficio obtenido en función de k y λ				
	$k = 1$	$k = 5$	$k = 25$	$k = 125$
$\lambda = 0.01$	84.2	84.2	84.14	84.15
$\lambda = 1$	83.81	83.77	83.93	83.54
$\lambda = 10$	68.73	61.95	51.24	50.26

Tabla 3: Resultados para cada combinación de k y λ

3.4. Experimento 4: Estudio del tiempo de ejecución en función del numero de estaciones, bicicletas y furgonetas

Observación	El tiempo de ejecución varia respecto al numero de estaciones, bicicletas y furgonetas
Planteamiento	Se escogen distintos números de estaciones, bicicletas y furgonetas y se observa la tendencia que sigue el tiempo de ejecución
Hipótesis	La función está directamente relacionada con el número de estaciones, furgonetas y bicicletas
Método	▪ Se eligen 10 semillas aleatorias. ▪ Para cada semilla habrá 15 ejecuciones (una por tamaño). ▪ Los tamaños elegidos seguirán la proporción 1 a 50 entre estaciones y bicicletas y 1 a 5 entre furgonetas y estaciones. ▪ Siguiendo la proporción habrán 15 tamaños, respecto a las estaciones empezando por 25 y aumentando de 25 en 25 hasta llegar a 375 estaciones. ▪ Se usrará Hill Climbing con la inicialización voraz y todo el conjunto de operadores.

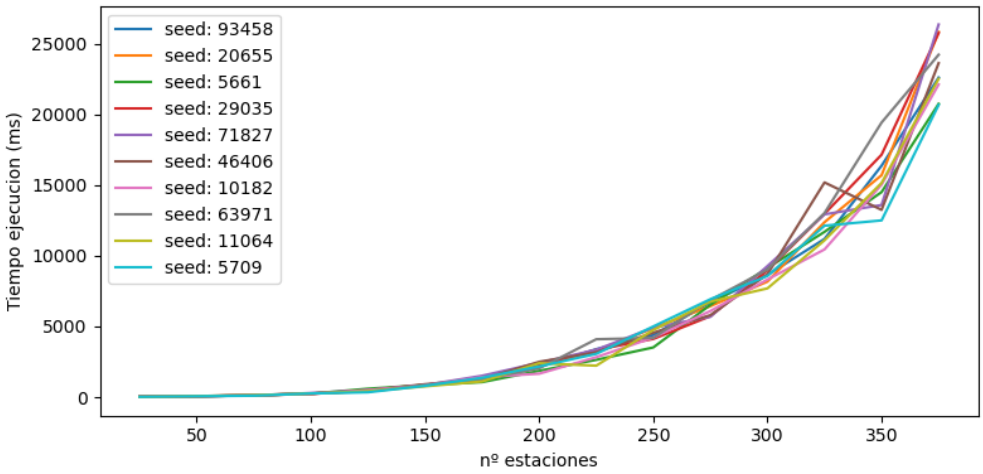

Figura 1: Evolución del tiempo de ejecución en función del tamaño de estaciones

En la figura 1 se ve como el tiempo sí crece en función del tamaño de estaciones, bicicletas y furgonetas (la gráfica si bien muestra el crecimiento en función del

numero de estaciones, al mantenerse las proporciones tendría la misma tendencia que si se mostrase en función de cualquiera de los dos otros parámetros). Aun así, se ha calculado el coeficiente de Pearson para comprobar cuan relacionados están el tiempo y el tamaño de la muestra.

$$r_{\text{tamaño}} = 0.86758612$$

El coeficiente de Pearson da un valor $r \in [-1, 1]$ siendo -1 una relación completamente inversa entre los dos parámetros (tiempo y tamaño en el caso del experimento), 1 una completamente directa y 0 para una relación nula. En el caso del experimento, el coeficiente de Pearson muestra una relación relativamente directa. No se puede determinar, con los datos obtenidos, cual es el crecimiento de la función aunque si se puede intuir que vendrá determinado mas por el factor de ramificación y el espacio de búsqueda, debido a los parámetros que afectan coste de ramificación de los operadores.

3.5. Experimento 5: Comparación de los dos algoritmos

Es importante comparar estos dos algoritmos de búsqueda local para así concluir cuál es mejor para el problema tratado en este proyecto. Para comparar los dos algoritmos nos fijaremos en tres aspectos principales: el coste temporal de las búsquedas, el beneficio total obtenido y la distancia total recorrida. Todo ello lo vamos a hacer para ambas heurísticas. Cabe recordar que los parámetros que se usaran en el Simulated Annealing en todas las pruebas son $k = 1$ y $\lambda = 0.0001$, para maximizar la eficiencia del algoritmo.

Observación	Puede que alguno de los dos algoritmos de búsqueda local sea mejor que el otro.
Planteamiento	Escogemos como algoritmos de búsqueda local Hill Climbing y Simulated Annealing y comparamos los resultados obtenidos.
Hipótesis	Los dos algoritmos llevan a las mismas soluciones (H0) o hay uno mejor
Método	▪ Se escogen 15 semillas aleatorias, una por réplica ▪ Para cada semilla, se hace una ejecución usando Hill Climbing y otra usando Simulated Annealing ▪ Nos quedamos con el beneficio, la distancia recorrida y el tiempo usado. ▪ Realizamos las medias de todo y las comparamos. ▪ El escenario será el mismo que en el experimento 1, fijando los operadores D1+D2+CO y la solución inicial voraz.

3.5.1. Comparación para la heurística 1

	Beneficio		Distancia (m)		Tiempo (ms)	
semilla	HC	SA	HC	SA	HC	SA
1	94	94	31000	30900	57	90
32700	77	77	25300	31700	56	85
4128	84	84	40500	47000	50	74
6986	94	94	26700	29800	53	71
34	84	84	16800	34200	58	78
5091	86	86	26200	40600	66	73
1743	80	80	26700	32600	74	88
87628	88	88	38200	60000	44	72
41234	105	105	26300	37100	69	79
40	63	63	16100	39100	63	85

Tabla 4: Beneficio, distancia y tiempo obtenido usando el primer heurístico

Se puede observar en la tabla 4 como tanto Hill Climbing como Simulated Annealing llegan a la misma solución siempre, para esta heurística y el escenario considerado. Esto puede ser debido a que ya se parte de una solución muy buena, y se pude converger fácilmente hacia una buena solución. Se puede observar también que los tiempos de ejecución son muy parecidos, por lo que se puede considerar que para la heurística 1 y el escenario planteado, ambos algoritmos son igual de buenos. No tiene sentido comparar las distancias en este punto ya que, como se mencionó en otro apartado, esta heurística no tiene en cuenta la distancia recorrida a la hora de guiar la búsqueda. Lo mas probable es que las diferencias se empiecen a ver en la comparación para la heurística 2.

3.5.2. Comparación para la heurística 2

	Beneficio		Distancia (m)		Tiempo (ms)	
semilla	HC	SA	HC	SA	HC	SA
1	52.1	55.35	9300	13500	66	82
32700	37.88	51.16	13700	8200	71	94
4128	51.7	53.92	12600	8000	74	81
6986	53.14	55.84	12700	12400	65	87
34	53.62	58.51	11400	8100	70	91
5091	50.64	58.11	8600	8700	74	93
1743	64.35	64.98	6900	6200	86	95
87628	52.76	54.96	10700	10400	62	100
41234	69.72	72.78	10500	10200	75	90
40	54.39	54.7	4600	4600	66	82

Tabla 5: Beneficio, distancia y tiempo obtenido usando el segundo heurístico

En esta segunda comparación ya se observan diferencias notables. Si se observa la tabla 5, Simulated Annealing siempre devuelve un mejor resultado que Hill Climbing, y casi siempre usando una distancia menor o igual. Cuando la distancia usada es mayor pero se ha obtenido mas beneficio, significa que se han dejado mas bicis que han permitido acercarse a la demanda prevista en ciertas estaciones, cosa que compensa el coste del transporte. Además los tiempos usados son ínfimos, por lo que el hecho que a veces Simulated Annealing use unos milisegundos de mas, no nos hace decantarnos por soluciones peores usando Hill Climbing. La conclusión es evidente: Simulated Annealing nos permite llegar a mejores soluciones con un coste temporal razonable, por lo que es mejor en este problema.

3.6. Experimento 6: Estudio del tiempo de ejecución en función de la demanda

Observación	El tipo de demanda puede afectar en el tiempo de ejecución
Planteamiento	Escogemos distintas semillas y comparamos el tiempo de ejecución en función de la demanda
Hipótesis	El tiempo de ejecución no varia en función de la demanda (H0) o hay una con menor tiempo respecto la otra
Método	▪ Se escogen 10 semillas aleatorias, una por réplica ▪ Para cada semilla, se ejecuta 1 experimento con demanda equilibrada y otro en hora punta ▪ El escenario será el mismo que en el experimento 1: mismo numero de estaciones, bicicletas y furgonetas; y mismas semillas ▪ Se usa Simulated Annealing como algoritmo de búsqueda, el primer heuristico, la inicialización voraz como solución inicial y el conjunto de todos los operadores para los experimentos.

Ya que hasta el momento no se ha estudiado el tiempo de ejecución en Simulated Annealing se ha optado por tomar por cada semilla dos tiempos, uno con demanda equilibrada y otro con demanda en hora punta. Si bien se pide compara el tiempo con el primer escenario, éste puede diferir enormemente simplemente por tener distinta inicialización y algoritmo de búsqueda.

	Tiempo de ejecución (ms)	
semilla	Demanda equilibrada	Demanda en Hora Punta
93458	49	48
20655	49	50
5661	50	51
29035	51	51
71827	52	51
46406	49	50
10182	49	49
63971	50	50
11064	49	49
5709	50	49

Tabla 6: Tiempo de ejecución en función de la demanda

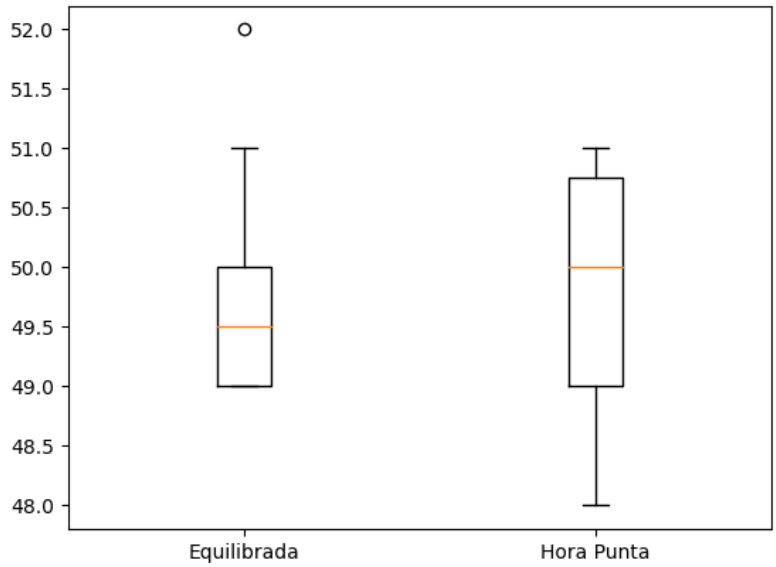

Figura 2: Distribución del tiempo de ejecución en función de la demanda para SA

Se observa en la figura 2 que la variación en el tiempo de ejecución es mínima entre las dos demandas. También se ve en la tabla 6 como la diferencia entre tiempos para todas las semillas no supera la milésima de segundo. Esto puede deberse a que Simulated Annealing va a asignar su sucesor más en función de la probabilidad $P(estado)$ que de la calidad de estos durante un numero determinado de iteraciones. Puede deberse también al tamaño del escenario.

$$P(estado) = e^{\frac{\Delta E}{\mathcal{F}(T)}}$$
$$\mathcal{F}(T) = k \cdot e^{-\lambda \cdot T}$$

Debido a los resultados se determina que no hay una variabilidad suficiente como para determinar que el uso de una demanda implique un tiempo de ejecución menor que con la otra, así que se asume que no varia en función de estas.

3.7. Experimento 7: Búsqueda del número óptimo de furgonetas

Observación	El número de furgonetas puede afectar al beneficio conseguido
Planteamiento	Escogemos diferentes números de furgonetas y anotamos sus resultados si mantenemos el resto de factores igual
Hipótesis	El número de furgonetas no afecta a la solución (H0) o puede mejorarla
Método	▪ Se escogen 10 semillas aleatorias, una por réplica. ▪ Para cada semilla, se ejecuta 1 experimento con diferentes números de furgonetas. ▪ El número de furgonetas se incrementará de 5 en 5 para cada prueba hasta no ver mejoras. ▪ El escenario será el mismo que en el experimento 1, con los tres operadores y las mismas semillas. ▪ Hill Climbing como algoritmo de búsqueda. ▪ Se buscará el número de bicicletas a partir del cual ya no hay beneficios.

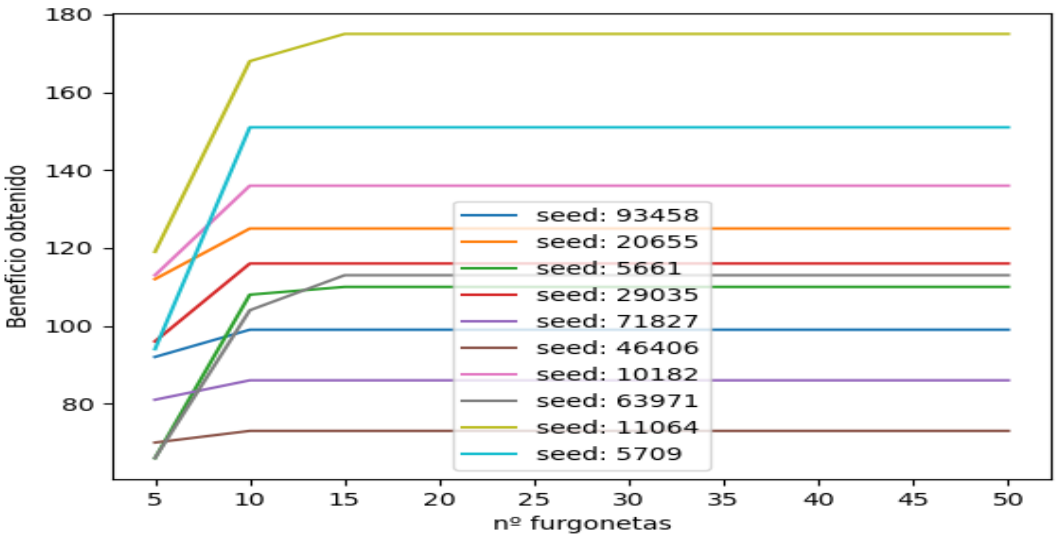

Figura 3: Demanda equilibrada

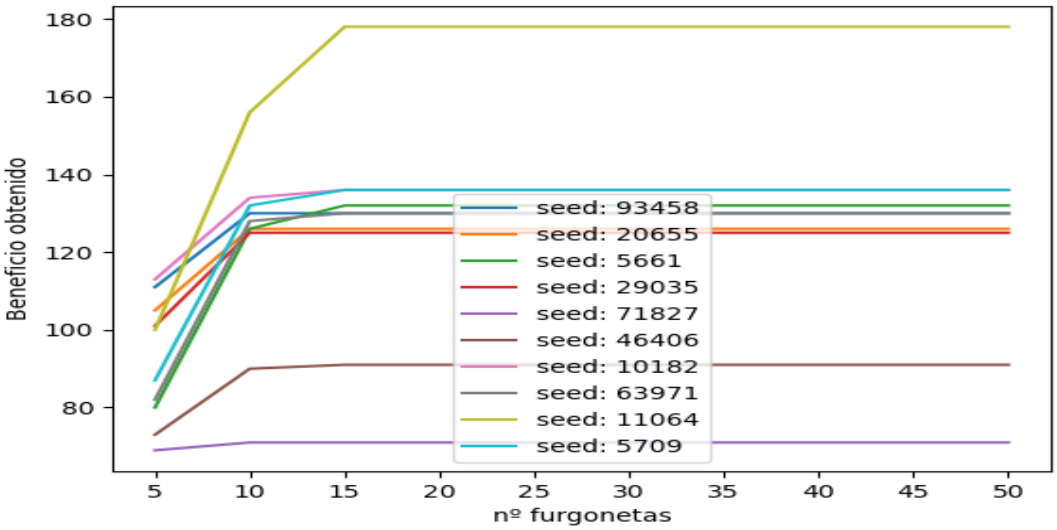

Figura 4: Hora punta

Se puede ver en la figura 3 que hay un incremento en el beneficio en el cambio de 5 a 10 furgonetas, y hay una mejora menor en el cambio de 10 a 15 para 2 de las 10 semillas escogidas; a partir de ahí el beneficio se mantiene constante. El mismo caso se observa para demanda en hora punta, como podemos ver en la figura 4, con la diferencia de que ahora son 3 de las 10 semillas las que sufren una mejora menor para el cambio de 10 a 15 furgonetas. Con estos datos, podemos deducir que, para este escenario, 15 es el número de bicicletas correcto para maximizar el beneficio.

3.8. Experimento especial

En el experimento especial se nos pedía el resultado del beneficio total obtenido y la longitud total del recorrido de las furgonetas que se obtiene en un escenario con 25 estaciones, 1250 bicicletas y 5 furgonetas en un escenario equilibrado, y cuánto tiempo se tardaba en encontrar la solución (en milisegundos). Se debía usar usar la semilla 1234 y usar los operadores, la inicialización y la heurística escogidos en los experimentos 1 y 2. Debemos mencionar que la versión más actual de la implementación nos sigue dando el mismo beneficio y tiempo empleado: 72 euros y aproximadamente 50 ms, pero la distancia recorrida ahora es distinta, se recorren 27900 metros, y no 33700. Esto es debido a que detectamos un pequeño error en la implementación sobre cómo dejamos bicis en las estaciones, y al solucionarlo la distancia obtenida cambió.

4. Conclusiones

Como conclusión principal en este proyecto hemos visto como es muy importante entender bien el problema que se nos plantea antes de intentar resolverlo. Resolver un problema de estas características es algo no trivial que requiere de un profundo estudio. En todo momento, los tres miembros del equipo hemos estado planteando enfoques para los distintos elementos que forman parte de la resolución del problema, de forma que incluso a medida que íbamos obteniendo resultados de los experimentos surgían ideas que podían (o no) mejorar el resultado. Sin embargo, para poder llegar a entregar la práctica en la fecha establecida, decidimos establecer una solución fija a partir de la cual experimentar. Está claro que siempre se puede mejorar una solución, pero disponemos de tiempo limitado.

Sobre los algoritmos usados para resolver el problema, hemos comprendido a la perfección porque es adecuado usar algoritmos de búsqueda local en este caso. El problema es demasiado complejo como para intentar obtener la mejor solución en tiempo razonable, por lo que nos bastan buenas soluciones. Hemos visto como Hill Climbing y Simulated Annealing trabajan para obtener buenas soluciones y, a pesar de que Simulated Annealing nos ha ido mejor en este trabajo, el hecho de usar un algoritmo u otro depende en gran parte del tipo de problema al que nos enfrentamos, entre otras cosas. Además, hemos podido ver como el tiempo de ejecución para Hill Climbing aumenta de forma considerable cuando el tamaño de los elementos que intervienen en el problema aumenta. Una característica a destacar es que Simulated Annealing es muy versátil en el sentido que podemos modificar muchos parámetros para intentar llegar a mejores soluciones, el truco está en experimentar.

Cabe destacar que no hemos querido incluir un operador de mover bicicletas porque ya lo implementamos y nunca se llegaba a usar para mejorar una solución, debido a la forma en que asignamos bicis dejadas que se ha mencionado en un apartado previo.

En conclusión, en inteligencia artificial es importante plantear un buen esquema inicial para resolver el problema, pero muchos de los parámetros que intervienen en la búsqueda de una solución se obtienen en base a la experimentación.

A. Proyecto de innovación

A.1. Descripción del tema escogido

En nuestro proyecto de innovación estamos estudiando la inteligencia artificial del videojuego F.E.A.R. (2005) de Monolith Productions, que fue, en su momento, una de las mejores y más avanzadas IAs diseñadas para un videojuego hasta ese momento.

A.2. Reparto del trabajo

- Ismael El Basli: Introducción, búsqueda de información sobre pathfinding y task networking en videojuegos y conclusión.
- Alex Garcés Gracia: Uso del autómata de tres estados en F.E.A.R. y el sistema de precondiciones y planificación.
- Rubén Catalán Rua: Búsqueda de información acerca de las técnicas de IA utilizadas en el videojuego. Uso de A* y el grafo con pesos como espacio de estados.

Es importante aclarar que esta es la división inicial del proyecto, que está sujeta a cambios, y no representa el trabajo realizado hasta el momento.

A.3. Dificultades encontradas

Buscar información acerca de un videojuego de casi 2 décadas está resultando ser algo tedioso ya que este no tuvo tanto renombre como otros en el momento. No se han encontrado otras dificultades hasta ahora.

Referencias

[1] Tommy Thompson. (2023-5-10). *History of AI in Games - Video Game - F.E.A.R.* modl.ai. Visitado el 18/10/2023.

[2] AI and Games. (2015-3-13) *Facing Your F.E.A.R. [AI & Games Lecture #3]. [Video file]. YouTube.* https://www.youtube.com/watch?v=rf2T_j-FlDE. Visitado el 21/10/23

[3] AI and Games. (2020-5-6) *Building the AI of F.E.A.R. with Goal Oriented Action Planning — AI 101. [Video file]. YouTube.* https://www.youtube.com/watch?v=PaOLBOuyswI. Visitado el 21/10/23

[4] Tamas Papp. (2020-4-23) *F.E.A.R.: shortest path for realistic AI.* lancaster.ac.uk. Visitado el 21/10/2023

[5] Xiao Cui y Hao Shi. *A*-based Pathfinding in Modern Computer Games.* University of London & Victoria University Melbourne. (2010-11). Vistado el 22/10/23