UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Proyecto de IA Bicing
Experimentación de IA

Santiago Cervera Pérez
Nicolás Gonzalo Longueria Millicay
Víctor Hernández Barragán

Contents

1 El problema

El problema abordado en este estudio se refiere a la optimización de la distribución de bicicletas en un sistema de préstamo de estas en un entorno urbano, con un enfoque específico en el servicio proporcionado por la empresa Bicing en Barcelona. El objetivo principal es desarrollar un enfoque basado en inteligencia artificial para gestionar la redistribución de bicicletas entre estaciones en una hora con el propósito de garantizar la disponibilidad de bicicletas para los usuarios en el momento que las necesiten.

En este problema se considera que la ciudad tiene forma de un cuadrado de 10x10 kilómetros, con calles que forman una cuadrícula de manzanas de 100x100 metros. En esta se encuentran las E estaciones que nos proporciona Bicing y con un total de B bicicletas distribuidas entre ellas. El número de bicicletas total es constante, sin la posibilidad de agregar o quitar bicicletas. Se asume que el espacio de almacenamiento de una estación es ilimitado y la distancia entre dos puntos de la ciudad se calcula utilizando la función de distancia euclidiana [1]:

$$d(i,j) = |i_x - j_x| + |i_y - j_y| \tag{1}$$

Para resolver el problema se cuenta con una flota de F furgonetas, cada una con capacidad máxima de 30 bicicletas. Cada furgoneta puede realizar un único viaje en una hora, transportando bicicletas entre una estación y un máximo de 2 estaciones, con la restricción de que no se permite que dos furgonetas recojan bicicletas de la misma estación en una misma hora.

El problema incluye con un sistema de beneficio, que consiste en pagar un euro por cada bicicleta que se transporte y que acerque el número de bicicletas de una estación a la demanda prevista. Por el contrario, se cobrará un euro por cada bicicleta que se transporte y que aleje a una estación de su previsión. El transporte de bicicletas tiene un costo adicional basado en la distancia recorrida [2].

$$Coste(n_b)/km = \lfloor (n_b + 9)/10 \rfloor \tag{2}$$

Por último añadir que existen dos tipos de escenarios direrentes de demanda: la *equilibrada*, en la que la demanda de bicicletas de cada estación tiende a ser equitativa, y la *horapunta*, en la que algunas estaciones tienen más demanda que otras.

1.1 El problema a nivel técnico

Tanto una Estación, como el conjunto de Estaciones, son clases programadas, donde Estaciones es un conjunto de Estación. Esta última contiene información relevante para el programa como el numero de bicicletas no usadas, la demanda para esta proxima hora y las coordenadas. Las furgonetas F es lo que hemos tenido que implementar, explicado posteriormente.

1.2 Busqueda Local

Para abordar la resolución de este problema, optaremos por la metodología de búsqueda local. Esta técnica implica comenzar con una solución inicial y explorar los posibles estados futuros mediante la aplicación de operadores al estado actual o anterior. La búsqueda local se destaca por su eficiencia y rapidez en la obtención de soluciones válidas para nuestro problema, en comparación con otros enfoques como la busqueda exhaustiva.

Este problema se presta particularmente bien a la aplicación de la búsqueda local debido a que cumple con ciertos requisitos fundamentales:

- Es un problema de optimización que busca generar un estado final conforme a una función de calidad predefinida.
- El camino específico seguido para alcanzar la solución óptima resulta irrelevante en el contexto de este problema.

Estas características del problema permiten la aplicación efectiva de la técnica de búsqueda local como enfoque de resolución.

2 Representación del problema

2.1 Representación del problema

Teniendo en cuenta lo anteriormente mencionado, representaremos el problema a partir de una matriz. Para ser exactos esta será una matriz de F x 5, donde F representa la cantidad de furgonetas que tenemos, por lo tanto, de manera implícita cada fila representa el identificador de una furgoneta. Las 5 columnas en orden representan:
s: La estación inicial desde donde empezará la furgoneta ($0 \leq s \leq EstacionesTotales$). Será -1 sino representa ninguna estación. Ninguna estación inicial será igual a otra.
sp1: La primera parada de la furgoneta ($0 \leq sp1 \leq EstacionesTotales$). Será -1 sino representa ninguna parada.
sp2: La segunda parada de la furgoneta ($0 \leq sp2 \leq EstacionesTotales$). Será -1 sino representa ninguna parada. La segunda parada no será igual a la primera.
tbic: La cantidad de bicicletas recogidas en la estación inicial (s) ($0 < tbic \leq 30$).
bic1: La cantidad de bicicletas dejadas en la parada 1 ($0 < bic1 \leq tbic$).

Hay que tener en cuenta, que estamos dejando de manera implícita la cantidad de bicicletas dejadas en la parada 2, que serían igual a la diferencia entre las bicicletas totales y las dejas en la parada 1, es decir, $bic2 = tbic - bic2$.

Una ejemplo gráfico de un problema sería el siguiente:

	s	$sp1$	$sp2$	$tbic$	$bic1$
F_0	2	5	6	10	5
F_1	7	1	0	25	23
...	...	...	...	...	...
F_n	12	9	-1	15	15

En el ejemplo anterior: La furgoneta 0 empieza en la estación 2 y recoge 10 bicicletas, hace una primera parada en la estación 5, donde deja 5 bicicletas y una segunda parada en la estación 6, donde deja 5 bicicletas. La furgoneta 1 empieza en la estación 7 y recoge 25 bicicletas, hace una primera parada en la estación 1, donde deja 23 bicicletas y una segunda parada en la estación 0, donde deja 2 bicicletas. La furgoneta n empieza en la estación 12 y recoge 15 bicicletas, hace una primera parada en la estación 9, donde deja 15 bicicletas y no realiza una segunda parada.

Esta representación es ideal para el problema, ya que estos son los mínimos elementos que nos permiten entender el desarrollo de todo el problema de la manera más simple.

2.2 Análisis del tamaño del espacio de búsqueda

En este caso el tamaño de espacio de búsqueda se resume a cuantas posibles combinaciones validas de representación de nuestro problema existen. Empezamos analizando la cantidad de estados con diferentes estaciones iniciales, un estado tiene F furgonetas por lo tanto F estaciones iniciales distintas, el orden de cómo se distribuyan estas estaciones no es relevante así que no tenemos en cuenta esas posibilidades, entonces las posibles combinaciones que podemos hacer con F furgonetas en un conjunto de E estaciones es igual a $C(E, F) = \frac{E!}{F!(E-F)!}$.

Ahora analizamos la cantidad de combinaciones posibles en una fila, asumimos que la estación inicial es una constante, entonces, la segunda columna podría elegir entre $E-1$ elementos (no contamos la estación inicial), la tercera columna podría elegir entre $E-2$ elementos (no contamos la estación inicial ni la primera parada), la cuarta columna varia entre 0 y 30 y la quinta es igual a la cuarta. Teniendo en cuenta todo esto podemos concluir que las combinaciones de una fila es $(E-1) \cdot (E-2) \cdot 30 \cdot 30$ y como tenemos F filas el coste se multiplicará por F. Combinando todo lo anterior el espacio de tamaño de búsqueda será aproximadamente:

$$\frac{N!}{F!(E-F)!} \cdot F(E-1) \cdot (E-2) \cdot 30 \cdot 30$$

Esto es número aproximado del mismo, ya que los casos reales posibles son mucho más limitados, y también estamos despreciando ciertos estados, como aquellos en los que las furgonetas no poseen estaciones iniciales. Pero en definitiva es una aproximación razonable. Ademas con este aproximado

2.3 Operadores del problema

Para los operadores, hemos hecho 2 sets de operadores que funcionan junto a las respectivas funciones heurísticas. El primer set consiste en un conjunto de operadores muy simple con un heurístico compuesto. Estos son los operadores que hemos definido para el progrma.

- Añadir Estación
  - *Parámetros*: Los parámetros son el identificador de la furgoneta *vn* y el identificador de la estación *st*.
  - *Precondición*: La estación *st* existe, la furgoneta *vn* existe, la estación *st* no puede estar como estación inicial con ninguna otra furgoneta y la estación inicial de la furgoneta *vn* no puede estar inicializada.
  - *Postcondición*: La furgoneta *vn* tiene como estación principal *st*.
  - *Factor de ramificación*: Cualquier furgoneta puede añadirse a cualquier estación no usada así que el factor de ramificación es de $O(|E| * |F|)$.
- Añadir Parada
  - *Parámetros*: Los parámetros son el identificador de la furgoneta *vn* y el identificador de la estación *st*.
  - *Precondición*: La estación *st* existe, la furgoneta *vn* existe, la estación de inicio de la furgoneta está inicializada y no es *st*, la parada 1 no es *st* y o parada 1 no está inicializada, o la parada 2 no lo está.

  - *Postcondición*: La furgoneta vn tiene como parada 1 o parada 2 (depende de cual no esté inicializada) la estación st.
  - *Factor de ramificación*: Cualquier furgoneta puede añadirse a cualquier parada así que el factor de ramificación es de $O(|E| * |F|)$.

- Eliminar Parada
  - *Parámetros*: Los parámetros son el identificador de la furgoneta vn y la parada 1 o 2.
  - *Precondición*: La furgoneta vn existe y, o la parada 1 está inicializada o la parada 2 lo está.
  - *Postcondición*: En la furgoneta vn, si la parada 1 es eliminada, el numero de bicis que deja en la parada 1 es 0, si el la parada 2, las bicis que dejas en la parada 1 son todas las bicis que tiene la furgoneta.
  - *Factor de ramificación*: Cualquier furgoneta puede eliminar paradas así que el factor de ramificación es de $O(2|F|)$.

- Acumular Bicis
  - *Parámetros*: Los parámetros son el identificador de la furgoneta vn y el numero de bicicletas no usadas $ntbic$ en la parada inicial de la furgoneta vn.
  - *Precondición*: La furgoneta vn existe, la parada inicial está inicializada, el numero de bicis es menor o igual al numero de bicis no usadas en la parada inicial de la furgoneta y el numero de bicis está entre 0 y 30.
  - *Postcondición*: La furgoneta recoje $ntbic$ de la parada inicial y deja en la parada 1 esta cantidad de bicis.
  - *Factor de ramificación*: Cualquier furgoneta puede recojer el total de bicis de una parada, pero como máximo 30, así que el factor de ramificación es de $O(|F| * 30)$.

- Dejar Bicis
  - *Parámetros*: Los parámetros son el identificador de la furgoneta vn y el numero de bicicletas no usadas $ntbic$ en la parada inicial de la furgoneta vn.
  - *Precondición*: La furgoneta vn existe, la estación inicial de la furgoneta vn está inicializada, $ntbic$ es menor o igual a las bicis recojidas en la estación inicial y $ntbic$ es mayor o igual que 0.
  - *Postcondición*: La furgoneta vn deja las bicicletas en la parada 1 o las deja en la parada 2.
  - *Factor de ramificación*: Cualquier furgoneta puede recojer el total de bicis de una parada, pero como máximo 30, así que el factor de ramificación es de $O(|F| * 30)$

- Swap
  - *Parámetros*: Los parametros son 2 identificadores de 2 furgonetas $vn1$ y $vn2$ y sus respectivas paradas: $sp1$ de la parada 1 y $sp2$ de la parada 2.

- *Precondición*: Las furgonetas $vn1$ y $vn2$ existen, las estaciones $sp1$ y $sp2$ existen, las furgonetas son diferentes entre sí, las estaciones son diferentes entre sí y las paradas de cada furgonetas están inicializadas.
- *Postcondición*: "Swapeas" las paradas entre las furgonetas $vn1$ y $vn2$.
- *Factor de ramificación*: Cualquier furgoneta puede "swapear" con cualquier otra furgoneta así que el factor de ramificación es de $O(|F|^2)$.

- Cambiar Estación 1
  - *Parámetros*: Los parámetros son el identificador de la furgoneta vn y el identificador de la estación st.
  - *Precondición*: La furgoneta vn existe, la estación st existe, la parada 1 está inicializada y la parada 1 es diferente a st.
  - *Postcondición*: La parada 1 es st.
  - *Factor de ramificación*: Cualquier furgoneta puede tener cualquier parada 1 que cumplan las precondiciones así que el factor de ramificación es de $O(|E| * |F|)$.

- Cambiar Estación 2
  - *Parámetros*: Los parámetros son el identificador de la furgoneta vn y el identificador de la estación st.
  - *Precondición*: La furgoneta vn existe, la estación st existe, la parada 1 y 2 están inicializadas, la parada 2 es diferente a st y la parada 1 es diferente a la parada 2.
  - *Postcondición*: La parada 2 es st.
  - *Factor de ramificación*:Cualquier furgoneta puede tener cualquier parada 2 que cumplan las precondiciones así que el factor de ramificación es de $O(|E| * |F|)$.

Y el segundo set consiste en un conjunto de operadores simple y complejos junto a una función heurística simple. Estos son los operadores:

- Añadir Estación
  - *Parámetros*: Los parámetros son el identificador de la furgoneta vn, el identificador de la estación st, el numero de bicicletas de recoje $npick$ y el identificador de la parada 1 stp.
  - *Precondición*: La estación st existe, la furgoneta vn existe, la estación st no puede estar como estación inicial con ninguna otra furgoneta, la estación inicial de la furgoneta vn no puede estar inicializada, $npick$ es un valor entre 0 y el numero de bicicletas no usadas de la parada inicial st.
  - *Postcondición*: La furgoneta vn tiene como para inicial st, que recojer $npick$ bicicletas, la parada 1 es stp y deja $npick$ bicicletas.
  - *Factor de ramificación*: Cualquier furgoneta puede tener cualquier estación inicial y cualquier parada 1, así que el factor de ramificació es de $O(|E|^2 * |F|)$

- Swap
  - *Parámetros*: Los parametros son 2 identificadores de 2 furgonetas $vn1$ y $vn2$ y sus respectivas paradas: $sp1$ de la parada 1 y $sp2$ de la parada 2.
  - *Precondición*: Las furgonetas $vn1$ y $vn2$ existen, las estaciones $sp1$ y $sp2$ existen, las furgonetas son diferentes entre sí, las estaciones son diferentes entre sí y las paradas de cada furgonetas están inicializadas.
  - *Postcondición*: "Swapeas" las paradas entre las furgonetas $vn1$ y $vn2$.
  - *Factor de ramificación*: Cualquier furgoneta puede "swapear" con cualquier otra furgoneta así que el factor de ramificación es de $O(|F|^2)$.
- Cambiar Estación 1
  - *Parámetros*: Los parámetros son el identificador de la furgoneta vn, el identificador de la estación st y el numero de bicis a añadir $nbicis$.
  - *Precondición*: La furgoneta vn existe, la estación st existe, la parada 1 está inicializada y la parada 1 es diferente a st, la parada 1 es diferente a la parada 2 y $nbicis$ es menor a las bicis recojidas de la estación de inicio.
  - *Postcondición*: La parada 1 es st, y el numero de bicis que deja en la parada 1 es $nbicis$.
  - *Factor de ramificación*:Cualquier furgoneta puede tener cualquier parada 1 que cumplan las precondiciones y solo pueden tener hasta 30 bicicletas, así que el factor de ramificación es de $O(|E| * |F| * 30)$.
- Cambiar Estación 2
  - *Parámetros*: Los parámetros son el identificador de la furgoneta vn, el identificador de la estación st y el numero de bicis a añadir $nbicis$.
  - *Precondición*: La furgoneta vn existe, la estación st existe, la parada 2 está inicializada y la parada 2 es diferente a st, la parada 1 es diferente a la parada 2 y $nbicis$ es menor a las bicis recojidas de la estación de inicio.
  - *Postcondición*: La parada 2 es st, y cambia las bicis de la parada 1.
  - *Factor de ramificación*: Cualquier furgoneta puede tener cualquier parada 2 que cumplan las precondiciones y solo pueden tener hasta 30 bicicletas, así que el coste de ramificación es de $O(|E| * |F| * 30)$.
- Acumular Bicis
  - *Parámetros*: Los parámetros son el identificador de la furgoneta vn y el numero de bicicletas no usadas $ntbic$ en la parada inicial de la furgoneta vn.
  - *Precondición*: La furgoneta vn existe, la parada inicial está inicializada, el numero de bicis es menor o igual al numero de bicis no usadas en la parada inicial de la furgoneta y el numero de bicis está entre 0 y 30.
  - *Postcondición*: La furgoneta recoje $ntbic$ de la parada inicial y deja en la parada 1 esta cantidad de bicis.
  - *Factor de ramificación*: Cualquier furgoneta puede recojer el total de bicis de una parada, pero como máximo 30, así que $O(|F| * 30)$.

- Dejar Bicis
  - *Parámetros*: Los parámetros son el identificador de la furgoneta *vn* y el numero de bicicletas no usadas *ntbic* en la parada inicial de la furgoneta *vn*.
  - *Precondición*: La furgoneta *vn* existe, la estación inicial de la furgoneta *vn* está inicializada, *ntbic* es menor o igual a las bicis recojidas en la estación inicial y *ntbic* es mayor o igual que 0.
  - *Postcondición*: La furgoneta *vn* deja las bicicletas en la parada 1 o las deja en la parada 2.
  - *Factor de ramificación*: Cualquier furgoneta puede recojer el total de bicis de una parada, pero como máximo 30, así que el factor de ramificación es de $O(|F| * 30)$.
- Intercambiar
  - *Parámetros*: Los parametros son el identificador de la furgoneta *vn* y el numero de bicicletas *nbicis*.
  - *Precondición*: La furgoneta *vn* existe, las paradas 1 y 2 estan inicializadas y $-1*$numero de bicis recogidas $+1 <=$ *nbicis* $<=$ numero de bicis recogida.
  - *Postcondición*: Intercambia las bicicletas dejadas de la parada 1 en la parada 2.
  - *Factor de ramificación*: Cualquier furgoneta se le puede intercambiar las bicis, así que el factor de ramificación es de $O(|F| * 30)$.
- Añadir Parada
  - *Parámetros*: Los parámetros son el identificador de la furgoneta *vn*, el idenfiticador de la parada *st* y el numero de bicicletas *ntbic*.
  - *Precondición*: La furgoneta *vn* existe, la estación *st* existe, la estación de origen esta inicializada, *ntbic* es mayor que 0 o menor que el numero de bicicletas recogidas en la parada inicial y o la parada 1 no está inicializada, o la parada 2 no lo está.
  - *Postcondición*: Si la parada 1 no está inicializada, la estación de la parada 1 es *st* y el numero de bicicletas que deja es *ntbic*, y su la parada 2 no está inicializadam la estación de la parada 2 es *st* y el numero de bicicletasd que deja es *ntbic*.
  - *Factor de ramificación*: Calquier furgoneta puede añadir cualquier para dentro de las condiciones mencionadas, así que el factor de ramificación es de $O(|F| * |E|)$.
- Añadir Parada 2
  - *Parámetros*: Los parámetros son el identificador de la furgoneta *vn*, el idenfiticador de la parada *st* y el numero de bicicletas *ntbic*.
  - *Precondición*: La furgoneta *vn* existe, la estación *st* existe, la estación de origen esta inicializada, *ntbic* es mayor que 0 o menor que el numero de bicicletas recogidas en la parada inicial, la parada 2 no está inicializada y la parada 1 sí está inicializada.

  - *Postcondición*: La parada 2 es *st* y el numero de bicicletas que deja en la parada 1 se resta con *ntbic*, es decir, el numero que deja en la parada 2 es *ntbic*.
  - *Factor de ramificación*: Cualquier furgoneta puede tener como parada 2 así que el factor de ramificación es de $O(|F| * |E|)$.

- Cambiar Paradas
  - *Parámetros*: Los parámetros son el identificador de la furgoneta *vn*, las 2 estaciones *sp*1 y *sp*2, y el numero de bicicletas *nbicis*.
  - *Precondición*: La furgoneta *vn* existe, las estaciones *sp*1 y *sp*2 existen, la estación principal está inicializada, *sp*1 es diferente a *sp*2, tanto *sp*1 y *sp*2 no son la estación de origen y *nbicis* es menor o igual a el numero de bicicletas que recoje en la estación de origen.
  - *Postcondición*: Las nuevas paradas 1 y 2 son *sp*1 y *sp*2 respectivamente y el numero de bicicletas que dejas en la parada 1 es *nbicis*.
  - *Factor de ramificación*: Cualquier furgoneta puede tener tanto cualquier parada 1 como cualquier parada 2 con un maximo de 30 bicis así que el factor de ramificación es de $O(|E| * |F|^2 * 30)$.

Creemos que estos operadores son adecuados para el problema ya que abarcan un espacio de busqueda total dado el problema. También podemos demostrar esto último ya que miramos todas las posibles combinaciones tanto de estaciones como de bicicletas recogidas como dejadas. Después de realizar varias pruebas previas a la experimentación, nos hemos dado cuenta que hay una cantidad de operadores que no se usan y otros que se usan muy poco (Experimento 1). Estos operadores no se usaban ya que hay otros que ya abarcaban todo ese rango de soluciones posibles, como por ejemplo, en el segundo set de operadores, el operador *Cambiar parada 1* hacia lo mismo que lo que hace una parte del operador *Añadir Estación*.

2.4 Generación de soluciones iniciales

Para realizar el estudio y análisis del problema hemos trabajado con 2 soluciones iniciales distintas. Para hablar del coste de cada solución inicial, diremos que las estaciones serán E y las furgonetas F.

2.4.1 Acercamiento básico

Esta solución simplemente es un estado vacío, donde ninguna furgoneta esta asignada a ninguna estación inicial, por lo que su coste es simplemente la asignación vacía de cada estación inicial y paradas, que tiene un coste $O(|F|)$.

2.4.2 Acercamiento aleatorio

Esta solución se genera de manera aleatoria basándose en orden de las estaciones, se asigna una estación diferente a cada furgoneta y paradas, se recogen todas las bicicletas no usadas, y se reparten de igual manera entre las paradas 1 y 2. Esta solución requiere que existan 3 veces la cantidad de estaciones que furgonetas, para así poder repartirlas sin problemas. El coste de esta solución radica en generar un vector ordenado de las E estaciones, por lo que su coste es $O(|E|)$.

2.4.3 Acercamiento greedy

Esta solución se genera escogiendo las estaciones mediante un valor equivalente al número de bicis que quedarán en cada estación en la próxima hora posteriormente a la satisfacción o no de la demanda.

En caso de que esta haya sido satisfecha, el valor sera mayor o igual que 0, y serán las F estaciones con este valor más alto, las estaciones iniciales de las F furgonetas ya que se puede recoger el mayor número de bicicletas sin alterar la satisfacción de la demanda.

En caso contrario este valor será negativo, y representará el número de bicis a añadir necesario para satisfacer la demanda, indicado en negativo. Serán las F estaciones con este valor más bajo, es decir las que necesitan que se les añadan más bicicletas, donde las F furgonetas depositarán las bicicletas recogidas en sus respectivas estaciones iniciales.

El coste de este acercamiento es $O(|E|log(|E|))$, ya que este es el coste del algoritmo merge sort que se utiliza para ordenar las E estaciones, también se ejecutan bucles de coste $O(|E|)$ y $O(|F|)$, pero al ser $|E|log(|E|)$ mayor este prevalece como el coste total.

2.5 Función heurística

Debido a que no sabemos cómo se irá desarrollando la experimentación hemos elegido uno de los heurístico propuestos y hemos creado uno combinado los 2 propuesto, además de que a estos les hemos agregado unos detalles para estudiar la diferencias y quedarnos con el mejor. Tambien que estos heurísticos los hemos agrupado con distintos operadores, ya que tenemos la hipótesis de que operadores más complejos irían mejor con un heurístico simple, mientras que un heurístico un poco más complejo (sumas ponderadas) basta con operadores más simples. Así que los heurísticos implementados al inicio fueron:

1. Maximización de lo que obtenemos por los traslados de las bicicletas
2. Ganancia total
3. 1 + ponderación
4. 2 + ponderación

El heurístico 2 es la ganancia total, que básicamente es la fusión de los 2 heurísticos propuestos, es decir, maximización de lo que obtenemos por los traslados de las bicicletas + minimización de los costes de transporte de las bicicletas.

Las ponderaciones realizadas en el heuristico 3 y 4 es la misma, y consiste en multiplicar por 10 el heuristico base y agregar un pequeño bonus dependiendo de la baja demanda de las estaciones iniciales y la alta demanda de las paradas.

3 Experimentación

3.1 Experimento 1

En este experimento se nos plantea comprobar que operadores dan mejores resultados en el siguiente escenario:

1. El numero de estaciones es 25.
2. El numero total de bicicletas es 1250.
3. El numero de furgonetas es 5.
4. La demandas es equilibrada.
5. Heurístico de que maximiza la ganancia obtenida por mover bicicletas

Este experimento se hará con el algoritmo de Hill Climbing. Para este experimento, como aún no hemos experimentado con las diferentes soluciones iniciales para decidir cuál es la mejor, lo haremos con las 3 planteadas.

1. **Observaciones:** Es lógico pensar que, dependiendo de la solución inicial con la que empieces, no todos los operadores se usaran de la misma manera. También hay que tener encuneta que el primer set de operadores es simple, si intentáramos empezar con la solución básica este no avanzaría nunca, ya que lo cambios realizado por los operadores no brindaría ningún beneficio, por lo que para el primer set usaremos el heurístico base + ponderaciones.
2. **Planteamiento:** Para estudiar y realizar este experimento cuantificaremos el número de veces que se usa cada operador, de esta manera podremos sacar las conclusiones adecuadas con más comodidad.
3. **Hipótesis:** Dependiendo del estado inicial se usarán más unos operadores que otros, pero lo que si creemos es que rara vez se utilizara el operador de Eliminar ya que esto supondría perder un beneficio, también pensamos que usando la solución básica el set de operadores complejos será mejor que el simple, pero quizás esto pueda cambiar cuando las soluciones iniciales sean las “greedy” o “mixed”. Aunque la solución ”greedy” justamente maximiza lo mismo que evalúa el heurístico, por lo que probablemente no haya ningún cambio usando esta solución.
4. **Metodología:** Usaremos el escenario comentado anteriormente para generar las soluciones iniciales. Usaremos el algoritmo Hill Climbing. Ejecutaremos el programa 20 veces con cada solución inicial con diferentes seed. Calculamos el número de veces en promedio que se usa cada operador con cada solución inicial.
5. **Resultados:**

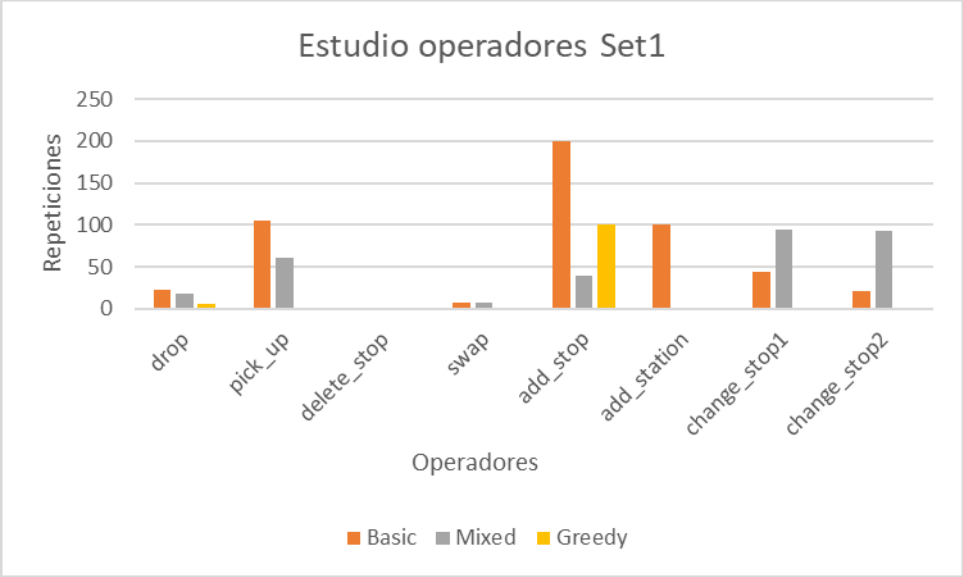


Figure 1: Repeticiones de los operadores del set1

Figure 2: Repeticiones de los operadores del set2

Conclusiones: En los anteriores gráficos podemos ver como se desempeñan los operadores dependiendo de la solución inicial, y también cual de los 2 sets de operadores tiene mejor capacidad a la hora de maximizar las ganancias obtenidas por las bicicletas. También hemos acertado en la hipótesis la cual proponía que operadores que redujeran el beneficio se ignorarían como lo es **delete_stop**. Pero también de manera sorpresiva, hay un montón de operadores definidos en el set2 que de primera pensábamos que iban a generar una diferencia en el experimento y al final no lo hicieron, como lo son: **drop**, **swap**, **change_stop1**, **change_stop2**, **change_stops** y **switch**. Teniendo en cuenta estos resultados a partir de ahora en set1 y set2 no usaremos ninguno de los anteriormente mencionados. Por último, podemos observar como la solución que menos operadores usa, es la greedy, como habíamos dicho en la hipótesis y además también se cumple lo que pensábamos que en soluciones complejas se desempeña mejor con el set de operadores

simples (Figure 1) set1 da un mejor resultado en la solución greedy frente al set2, pero en promedio el set2 es mejor. Para reforzar la decisión de haber eliminado estos operadores lo que hicimos fue hacer unas métricas antes de haber eliminado los operadores (las mismas que en el experimento 2) y las comparamos con las nuevas métricas luego de haberlos eliminados (Figura 4,5,6)

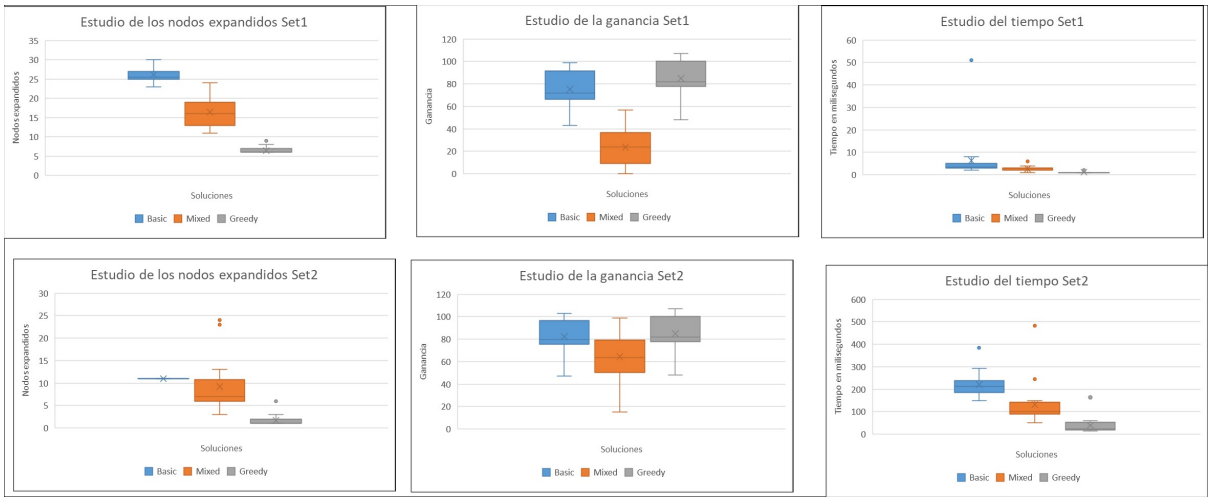


Figure 3: Repeticiones de los operadores del set2

Como observamos no existe diferencia sustancial entre el set1 antes de eliminar el operador delete stop y después de eliminarlo, por lo que la decisión de eliminarlo está fuertemente validada. Ahora en cambio al haber eliminado dichos operadores del set2, lo que estamos obteniendo lógicamente es una menor cantidad de nodos expandidos, lo que puede llegar a provocar la mayor de las diferencias y la razón por la que creemos que este cambio fue positivo, que es en la notable diferencia en los tiempos de ejecución que no impacta en las medias de las ganancias obtenidas, aunque si afecta negativamente al mixed, pero creemos que la ganancia en tiempo de ejecución lo vale.

3.2 Experimento 2

Para el segundo experimento se nos propone determinar que estrategia de generación de la solución inicial genera mejores resultados para la función heurística y escenario utilizados en el anterior experimento, y de nuevo, utilizando el algoritmo de Hill Climbing. Nota: Todos los tiempos están en milisegundos.

1. **Observaciones:** Esto ya lo realizamos en el experimento 1, pero tendremos que repetirlo ya que debido al experimento 2, hemos eliminado operadores.
2. **Planteamiento:** Para este experimento compararemos los tiempos, nodos expandidos, coste y beneficio total de cada solución para poder sacar conclusiones coherentes y adecuadas.
3. **Hipótesis:** Partimos de la hipótesis que los estados iniciales que se generan de una manera más greedy empezarán con mejores soluciones, pero expandirán poco y darán peores resultados que las demás. También esperamos que las ganancias sean mejores entre las soluciones que utilizan el set2 de operadores frente las del set1, pero debido a esto también el tiempo del set2 será superior al del set1.
4. **Metodología:**
   - Usaremos el mismo escenario del experimento anterior.
   - Aplicaremos el algoritmo de Hill Climbing.
   - Ejecutaremos el programa 20 veces con diferentes seed para cada solución inicial.
   - Compararemos la heurística, el tiempo y los nodos expandidos de cada solución inicial para poder sacar conclusiones con un amplio conocimiento de las capacidades de cada estado inicial.
5. **Resultados:**

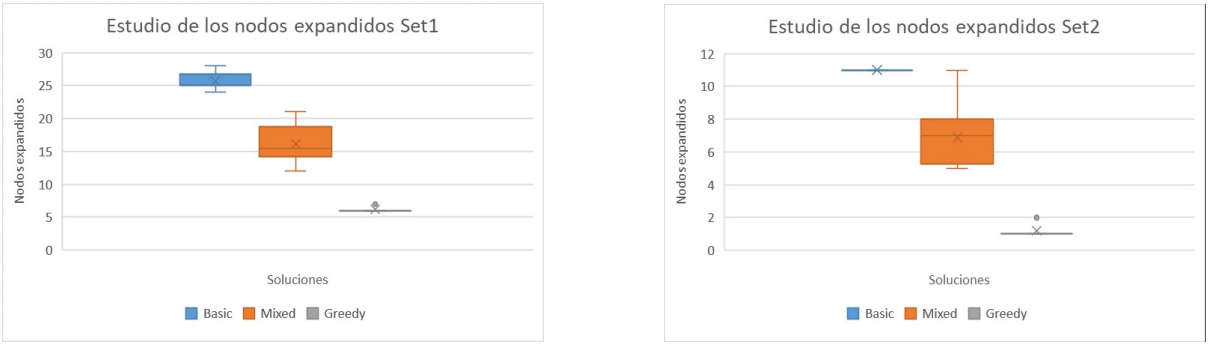
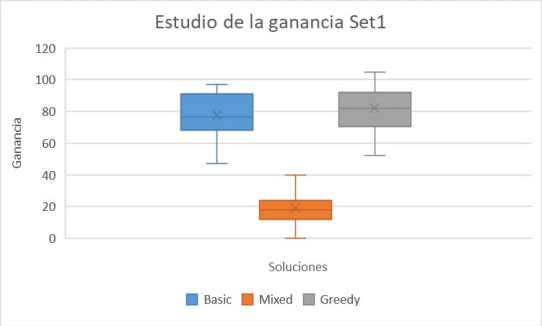


Figure 4: Expansión de nodos

Figure 5: Tiempo de ejecución



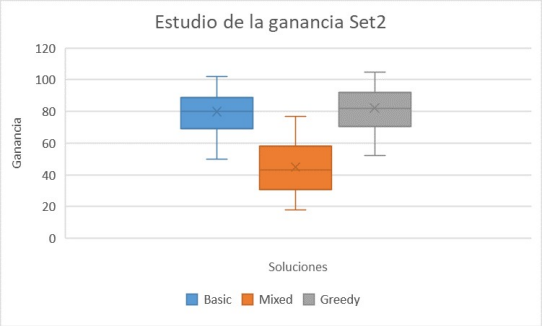


Figure 6: Ganancia total

Conclusiones: Con los resultados anteriores podemos llegar a las siguientes conclusiones, el set2 expande menos nodos que el set1, y en ambos sets la de expansión va decreciendo en el orden Basic, Mixed y Greedy. El tiempo que tarda el set2 es considerablemente mayor al set1, y mientras que en el set1 no hay mucha diferencia entre cada solución, en el set2 existe una gran diferencia entre la solución Basic y el resto. Además, también podemos observar lo mismo que pasaba con la expansión de nodos, esta va decreciendo en el mismo orden. Y por ultimo las ganancias la cual gana el set2 frente el set1, la ganancia no llega a ser mucho mayor pero tampoco despreciable.

Todo lo anterior coincide con la hipótesis que teníamos, exceptuando los resultados de las soluciones greedy, las cuales terminaron siendo las mejores. Para esta contradicción podemos pensar que el estado utilizado es demasiado pequeño para que el resto de soluciones iniciales se vean beneficiadas.

Debido a las conclusiones obtenidas de este experimento, valoramos que las diferencias de costes en tiempo que poseen las soluciones del set2 en comparación del set1 son aceptable, así que a partir de ahora trabajaremos únicamente con el set2 y aunque la solución inicial Greedy haya sido la más beneficiosa, la ignoraremos y trabajaremos con la Basic, ya la cantidad de nodo que expande es insignificante, cosa que es obvia ya que es un algoritmo greedy que maximiza justamente este parámetro, por lo tanto siempre debería ser la mejor solución para esta cuestión.

3.3 Experimento 3

En este tercer experimento nuestro objetivo es determinar los parámetros que mejor resultado ofrecen para el algoritmo Simulated Annealing, de nuevo con la misma heurística, operadores y estrategia de generación de la solución inicial escogidos en los anteriores experimentos.

1. **Observaciones:** Diferentes valores en los parámetros de Simulated Annealing darán resultados diferentes.
2. **Planteamiento:** Para cada conjunto de parámetros nos quedaremos con la heurística para después poder compararlos.
3. **Hipótesis:** Los valores extremos no funcionaran de la mejor manera.
4. **Metodología:**
    - Usaremos el mismo escenario que en el experimento 2.
    - Usaremos el algoritmo de Simulated Annealing con 10^4 pasos.
    - Exploraremos los siguientes conjuntos de parámetros.
    - stiter = [1, 50, 100, 150, 200]
    - k = [1, 2, 25, 400, 8000]
    - λ = [1, 0.1, 0.01, 0.001, 0.0001]
    - Para cada conjunto de parámetros ejecutaremos el programa 10 veces y haremos el promedio de la heurística.
    - Nos quedaremos con el promedio máximo entre todas las combinaciones.
5. **Resultados:**

Conclusiones:

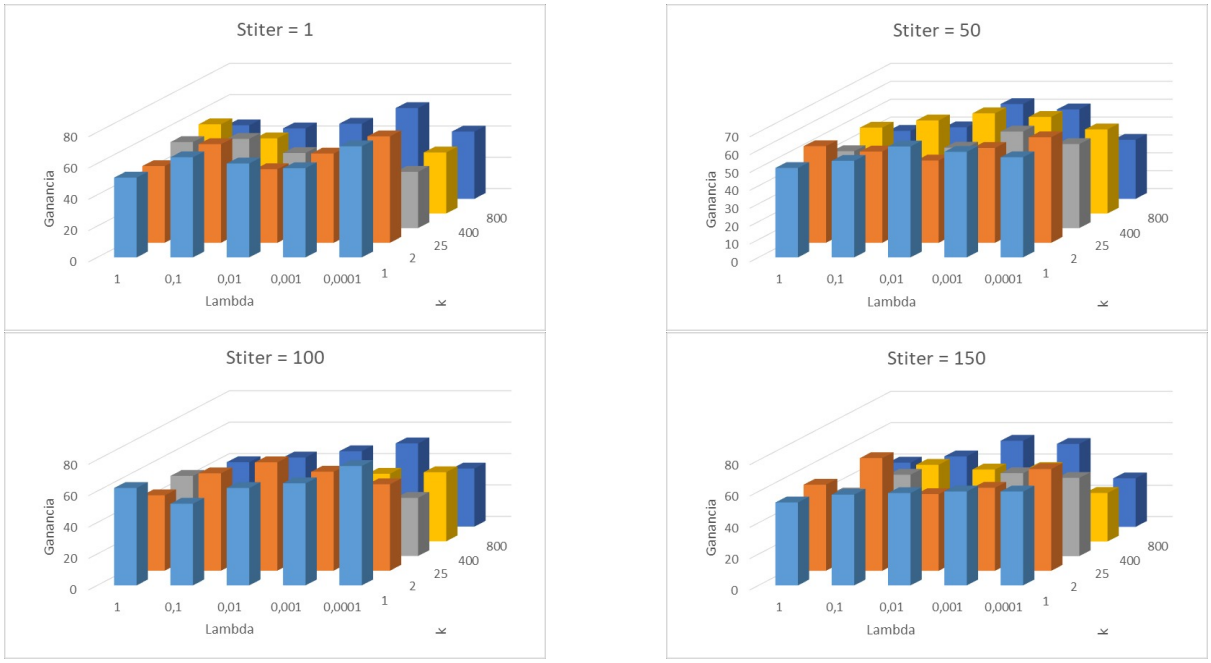


Figure 7: Estudio paramtros Simulated Annealing Stiter de 1 a 150

Figure 8: Estudio paramtros Simulated Annealing Stiter de 200

Conclusiones: En estos gráficos al ser 3D es complicado de apreciar las diferencias en la ganancia y ver a simple vista cual es el mejor (no pudimos agregar la malla), pero viendo los datos podemos concluir que los parámetros que nos dan mejores resultados son stiter = 100, $\lambda = 0{,}0001$ y k = 1. Lo que nos contradice la hipótesis que pensábamos que los extremos serian peores, y de hecho parecen ser mejores en términos generales. Nota: Todos los tiempos están en milisegundos.

3.4 Experimento 4

En el cuarto experimento se nos pide estudiar la evolución del tiempo de ejecución para hallar la solución en función del número de estaciones, furgonetas y bicicletas asumiendo una proporción 1 a 50 entre estaciones y bicicletas y de 1 a 5 entre furgonetas y estaciones. Nuevamente, utilizaremos Hill Climbing y la misma función heurística hasta ahora.

1. **Observaciones:** Incrementando los parámetros del problema se vería un crecimiento en el tiempo de ejecución.
2. **Planteamiento:** En cada ejecución y en cada caso, aumentamos el valor de un parámetro específico de manera uniforme.
3. **Hipótesis:** Debido a la gran cantidad de sucesores que generan lo operadores del set2, especulamos que cuanto más grande el número de estaciones, bicicletas y furgonetas lógicamente mayor será el tiempo, pero este incremento no será lineal, sino que podría llegar ser exponencial o logarítmico.
4. **Metodología:** Como indica el enunciado del mismo experimento iremos aumentando las estaciones en 25 en 25 empezando desde 25 hasta 125 estaciones manteniendo todo el tiempo las proporciones mencionadas. Cada uno de estos pasos se hará 10 veces con semillas diferentes y haremos el promedio del tiempo.
5. **Resultados:**

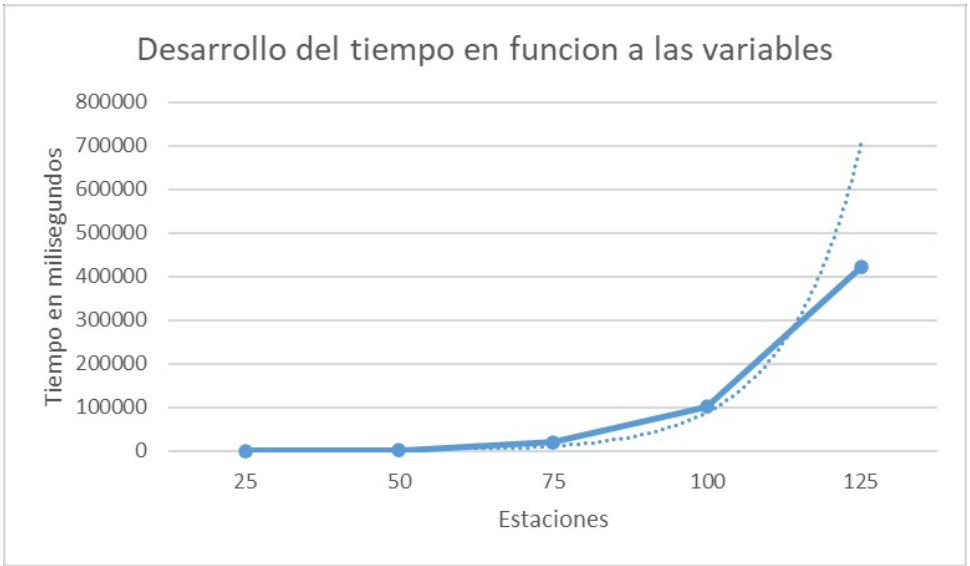


Figure 9: Desarrollo del tiempo en función de las variables

Conclusiones: Con 4 iteraciones ya es suficiente para poder observar una clara tendencia, esta tiene una forma exponencial como se puede apreciar. Para comprender mejor esta tendencia podemos aplicar un modelo de regresión exponencial y así ajustar la tendencia a una curva exponencial como se ve en el gráfico. El modelo de regresión exponencial nos da como resultado una curva $y = 23.687e^{2.0594x}$, con una $R^2 = 0.9873$, la R^2 es un variable que nos indica la semejanza, cuanto más se aproxima a 1, de la curva generada por el modelo de regresión exponencial y la curva obtenida por los datos, en este caso el R^2 es muy cercano a 1, por lo que la curva obtenida es confiable y a partir de esta podemos predecir aproximadamente cuanto tiempo tomará para las próximas estaciones.

3.5 Experimento 5

En este experimento debemos comparar el algoritmo de Hill Climbing y Simulated Annealing. Nota: Todos los tiempos están en milisegundos.

1. **Observaciones:** Veremos cuál de los 2 algoritmos es mejor en cada heurístico.
2. **Planteamiento:** Partimos de la solución básica (la que es vacía) y compararemos los 2 algoritmos teniendo en cuenta el beneficio obtenido, la distancia total recorrida y el tiempo de ejecución para cada heurístico.
3. **Hipótesis:** Nuestra hipótesis inicial es que, en las comparaciones de los heurísticos, el heurístico que maximiza el beneficio por el traslado de bicicletas tendrá un mejor desempeño para ello lógicamente, y en cambio el heurístico que intenta maximizar el beneficio total (beneficio de bicicletas – el gato de transporte) tendrá un mejor rendimiento en el recorrido, ya que para maximizar a la ganancia total hace falta minimizar las distancias, en tema de tiempo entre heurísticos, creemos, que tardará mas el heurístico del beneficio total, ya que este debe buscar un equilibrio entre maximizar el beneficio entre bicicletas y minimizar las distancias entre las estaciones. En cuestión a los algoritmos, realmente no sabemos cómo se podrán desempeñar, ya que al ser el Simulated Annealing dependiente de la suerte, puede ser que logre encontrar un máximo local mayor o incluso el absoluto, o en cambio caer en un mínimo local que no le permita recuperarse.
4. **Metodología:**
    - Usaremos el mismo escenario que en el experimento 1.
    - Usaremos el algoritmo Hill Climbing con los cambio realizados en los experimentos anteriores.
    - Usaremos el algoritmo Simulated Annealing con los parámetros obtenidos en el experimento 2.
    - Utilizaremos el heurístico de la ganancia por traslado de bicicletas y de beneficio total (ganancia por traslado de bicicletas - coste de transporte).
    - Ejecutaremos 10 veces cada combinación entre heurístico y algoritmo, y calcularemos beneficio obtenido, la distancia total recorrida y el tiempo de ejecución.
    - Haremos una comparativa entre algoritmo/heurístico y veremos si podemos determinar a un ganador.
5. **Resultados:**

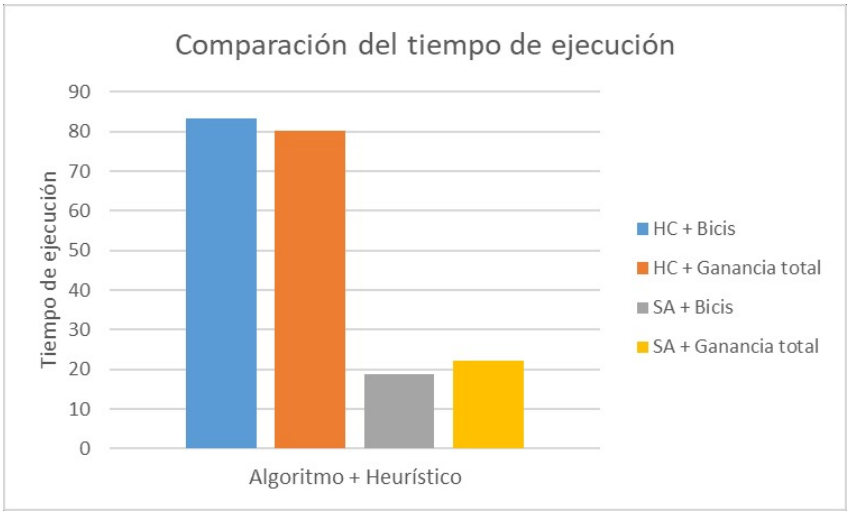


Figure 10: Comparación del tiempo de ejecución entre algoritmo + heurístico

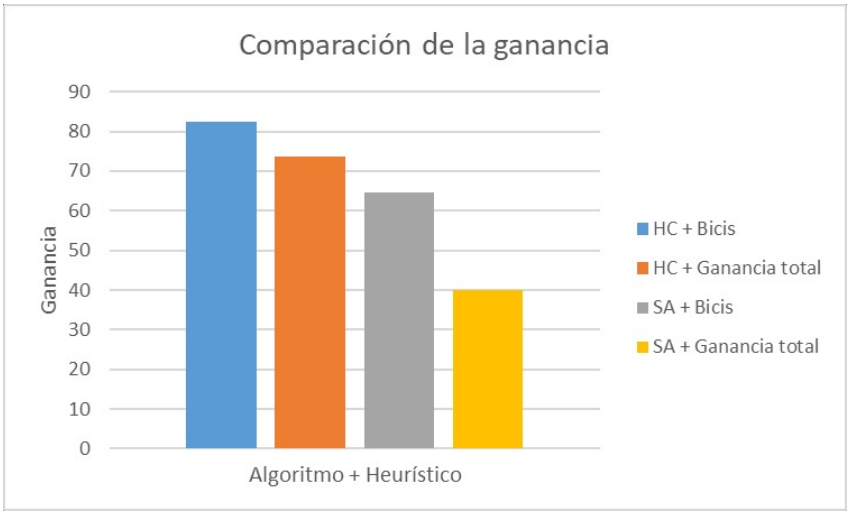


Figure 11: Comparación de la ganancia entre algoritmo + heurístico

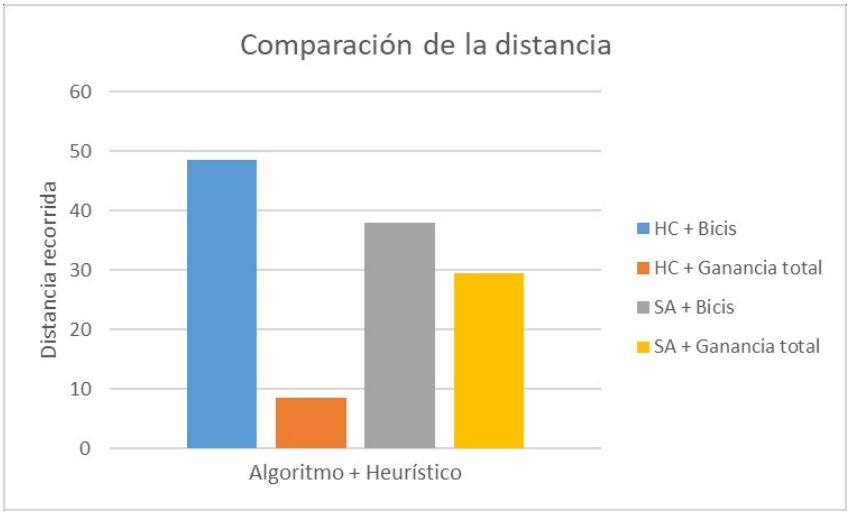


Figure 12: Comparación de la distancia entre algoritmo + heurístico

Conclusiones: Los resultados obtenidos en el estudio concuerdan notablemente con nuestra hipótesis inicial. En particular, se observa que el Hill Climbing (HC) aplicado con el heurístico de ganancia al trasladar bicicletas logra el mayor beneficio en esta tarea. Por otro lado, el HC utilizando el heurístico de ganancia total demuestra un rendimiento sobresaliente en términos de reducción de distancias en comparación con otras estrategias.

Los gráficos generados en el estudio proporcionan una comprensión visual de estos hallazgos. Además, es evidente que el algoritmo de Simulated Annealing (SA) no alcanza el mismo nivel de desempeño que el Hill Climbing en nuestra aplicación específica. Sin embargo, es importante destacar que el Simulated Annealing ofrece la ventaja de requerir significativamente menos tiempo de procesamiento en comparación con el Hill Climbing.

Así que podemos concluir que la elección entre el SA y HC dependerá de nuestras necesidades específicas. Si estamos buscando una solución razonable o aceptable y estamos dispuestos a sacrificar un poco de calidad en beneficio de una ejecución más rápida, el Simulated Annealing puede ser una elección adecuada debido a su menor costo en tiempo.

Por otro lado, si nuestro objetivo principal es obtener la mejor solución posible, incluso si esto implica un mayor tiempo de procesamiento, el Hill Climbing con los heurísticos mencionados podría ser la opción preferida, ya que parece ofrecer un rendimiento más sólido en términos de calidad de la solución.

La elección final entre SA y HC dependerá de las prioridades específicas de cada proyecto, como el equilibrio entre la calidad de la solución y la eficiencia en el uso de recursos.

3.6 Experimento 6

En este experimento veremos cuales pueden ser las diferencias en tiempo de ejecución de una solución equilibrada y en hora punta. Nota: Todos los tiempos están en milisegundos.

1. **Observaciones:** En la hora punta las demandas son más variables, veremos si esto es un problema más complejo de análisis o por lo contrario mas sencillo.
2. **Planteamiento:** Compararemos los nuevos datos con los obtenidos para el mismo algoritmo, pero utilizando una solución equilibrada del experimento 5.
3. **Hipótesis:** Inicialmente, podemos percibir el problema de la hora punta como algo más complejo, pero creemos que es lo contrario. Durante la hora punta, algunas estaciones experimentan una demanda significativamente mayor que otras, lo que, en realidad, facilitará que el algoritmo seleccione estaciones de inicio con un alto número de bicicletas y las distribuya en paradas con una demanda elevada.
4. **Metodología:**
   - Usaremos el mismo escenario que en el experimento 1.
   - Usaremos el algoritmo Hill Climbing con los cambio realizados en los experimentos anteriores.
   - Utilizaremos el heurístico de la ganancia por traslado de bicicletas.
   - Las estaciones tendrán una demanda de hora punta.
   - Ejecutaremos 10 veces el algoritmo con estaciones con una demanda en hora punta aleatorias. Tomaremos las muestras que tarda en ejecutarse el algoritmo y luego haremos el promedio.
   - Haremos una comparativa entre el tiempo del mismo algoritmo pero con una demanda equilibrada.
5. **Resultados:**

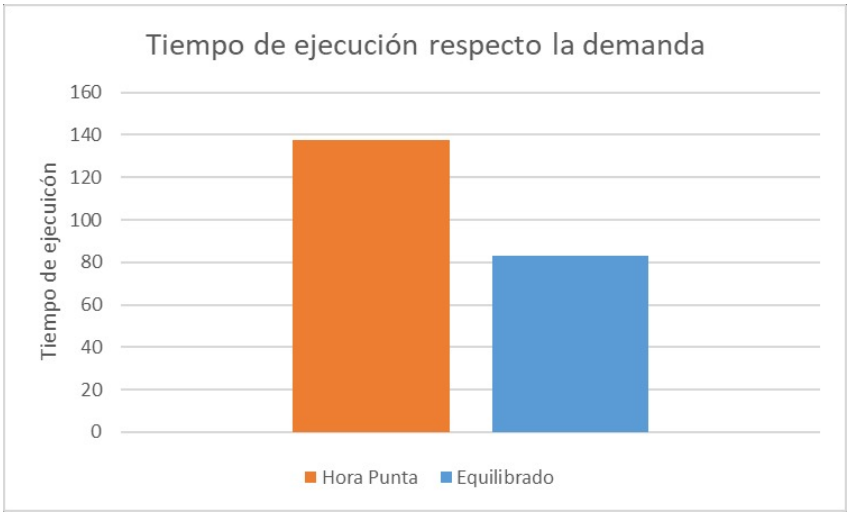


Figure 13: Comparación del tiempo de ejecución respecto la demanda.

Conclusiones: Contradictorio a nuestra hipótesis, el tiempo de ejecución durante la hora punta es mayor que el del escenario equilibrado. Sin embargo, debido a la notable diferencia, realizamos nuevamente una medición del tiempo de ejecución para la demanda

equilibrada, y la diferencia disminuyó en comparación con la de la hora punta. Por lo tanto, podríamos concluir que la demanda durante la hora punta tarda más en ejecutarse, pero la diferencia no parece ser significativamente mayor.

3.7 Experimento 7

En este experimento estudiaremos la ganancia en función de las furgonetas y si existe una deferencia dependiendo del tipo de demanda que tengamos.

1. **Observaciones:** Hay que tener en cuenta que la cantidad total de furgonetas está limitado a la cantidad de estaciones, ya que las estaciones iniciales no se pueden repetir.
2. **Planteamiento:** Observaremos la ganancia en función de las furgonetas y luego veremos si hay cambios dependiendo la demanda.
3. **Hipótesis:** Es lógico pensar que una vez la cantidad de furgonetas iguale a la de las estaciones no habrá mejora, ya que las furgonetas únicamente pueden empezar en estaciones distintas. Así que esto ya los sabemos con firmeza. pero, además, también esperamos que la evolución deje de aumentar incluso antes, ya que la repetición de bicicletas se realiza de la estación con menos demanda a las de mayor, es decir, que las estaciones que pueden repartir furgonetas es limitada, así que una vez repartido las bicicletas de estas estaciones no quedaran más estaciones con bicicletas para repartir, y a partir de ese punto no habrá mejora.
4. **Metodología:**
    - Usaremos el mismo escenario que en el experimento 1.
    - Usaremos el algoritmo Hill Climbing con los cambio realizados en los experimentos anteriores.
    - Utilizaremos el heurístico de la ganancia por traslado de bicicletas.
    - Las estaciones tendrán una demanda equilibraada y de hora punta.
    - Iremos incrementando en 5 en 5 la cantidad de furgonetas empezando desde el 5 hasta el 25 que es la cantidad de estaciones del experimento 1.
    - Cada incremento lo ejecutaremos 10 veces y haremos la media de la ganancia obtenida.
    - Colocaremos las ganancias obtenidas junto el incremento de las furgonetas e intentaremos observar un punto de estancamiento o inflexión, luego compararemos los datos entre demanda equilibrada y hora punta.
5. **Resultados:**

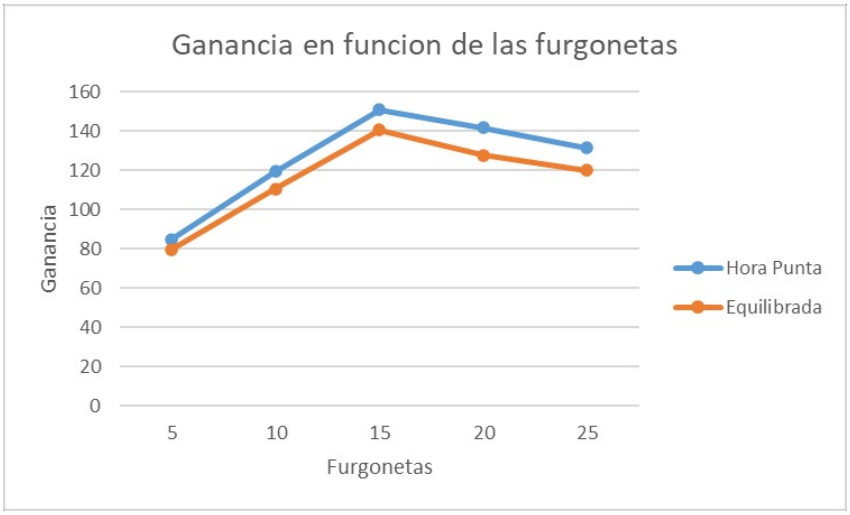


Figure 14: Ganancia en funcion de las furgonetas.

Conclusiones: Gracias al grafico anterior claramente podemos concluir que la mejor cantidad de furgonetas que podemos utilizar es 15, en caso de utilizar mas no solo es que no tendríamos mayor ganancia sino que incluso tendríamos perdidas. Estos resultados concuerdan con nuestra hipótesis inicial. Tampoco parece haber una diferencia sustancial de la ganancia en función de las furgonetas dependiendo del tipo de demanda, ambas se comportan similarmente.

4 Conclusión

A lo largo de la ejecución de este proyecto, hemos profundizado significativamente en nuestra comprensión de la operatividad de diversos algoritmos de búsqueda local, en particular, Hill Climbing y Simulated Annealing. A medida que avanzábamos en nuestra investigación, hemos constatado que, a pesar de que el problema que hemos abordado sea una simplificación de situaciones de la vida real, estos algoritmos han demostrado su capacidad para rendir eficazmente cuando se les proporciona una especificación adecuada y se implementan de manera precisa.

Un hallazgo notable ha sido que Simulated Annealing es capaz de alcanzar resultados heurísticos muy similares a los obtenidos mediante Hill Climbing, y lo hace en un intervalo de tiempo considerablemente más corto. También hemos llegado a la conclusión de que la utilización de un conjunto de operadores más complejos conlleva a mejores resultados, aunque esto viene acompañado de una mayor carga de procesamiento y consumo de memoria.

Durante la realización de este trabajo, también adquirimos conocimientos en el uso de Java, ya que hasta ese momento no habíamos tenido experiencia con este lenguaje de programación. Además, exploramos el uso de GitHub, una tecnología que previamente no habíamos empleado. Por último, hemos utilizado LaTeX, una tecnología mediante la cual hemos formateado y redactado el presente documento.

4.1 Dificultades y posibles futuras mejoras

Tras haber realizado toda la parte de experimentación, nos dimos cuenta que el programa era ineficiente, tanto en tiempo como en memoria (Ver Experimento 4). Tras hacer algunas busquedas, nos dimos cuenta que el problema surgía en la función Heurística, haciendo que cada vez que mirase un sucesor, tenga que hacer un recorrido por todas las estaciones y furgonetas. Intentamos cambiar esta función implementando un vector auxiliar para no hacer tanto recorrido y que la función Heurística sea casi constante, haciendo que cada vez se vaya recalculando el Profit en el operador, pero no pudimos acabar de implementarlo tras ver que no funcionaba bien tanto con el Simulated Annealing o como en el Set1 de operadores. Aunque no hayamos podido hacer el cambio, vimos correcto comentarlo. La mejora obtenida es aproximadamente del 33%. Estos problemas se deben un planteamiento inicial de operadores y cálculo de la heurística en los cuales se infravaloró el potencial coste en memoria y tiempo que podrían llegar a utilizar.

5 Trabajo de innovación: SAM

5.1 Descripció:

Segment Anything Model (SAM): es un nuevo modelo de IA de Meta AI que puede "recortar" cualquier objeto, en cualquier imagen, con un solo clic. Permite indicaciones flexibles, cálculo de máscaras en tiempo real y conciencia de ambigüedad en tareas de segmentación.

5.2 Reparto del trabajo:

- **Santiago Cervera:**Se encargara de recopilar información del impacto a nivel social de esta invención.
- **Nicolás Longueira:**Se encargará de hacer un análisis del funcionamiento a nivel técnico de esta tecnologia junto con Victor.
- **Victor Hernández:**Se encargará de hacer un análisis del funcionamiento a nivel técnico de esta tecnologia junto con Nicolás.

5.3 Dificultades

Las mayores dificultades con las que nos estamos encontrando son los pocos datos disponibles de esta tecnologia debido a su novedad.

5.4 Referencias

Introducción a SAM : https://segment-anything.com/
Código de SAM : https://github.com/facebookresearch/segment-anything
Demo de SAM : https://segment-anything.com/demo
Demo de SAM : https://segment-anything.com/demo