Práctica Bicing

Búsqueda Local

Inteligencia Artificial GEI Q1 Curso 2023-2024

Oriol Roca, Álvaro Terrón y Alex Bueno

Profesor: Ignasi Gómez

ÍNDICE

1 Identificación del problema

Bicing es una empresa que se dedica al servicio de bicicletas públicas.Su principal objetivo es cumplir con el abastecimiento de bicicletas necesarias en cada estación de forma eficiente en una estaciones repartidas por la ciudad en una hora específica. Por tanto se nos dan unas furgonetas para poder repartir las bicicletas según unos criterios y restricciones que se especificarán más adelante para evaluar las soluciones y se nos facilitan unas clases Java.

1.1 Elementos del problema

Hay dos elementos que serán fundamentales para el problema: las estaciones y las furgonetas. La combinación de estos dos elementos formarían unas “rutas” que seguirán las furgonetas a la hora de repartir. Así que podemos asumir que estos elementos formarán parte del estado.

1.1.1 Las estaciones

Una estación tiene como propiedades sus coordenadas y tres elementos que serán muy importantes para la asignación de las rutas y bicicletas:

- Número de bicicletas no utilizadas. Son aquellas que los usuarios no utilizarán en esa hora y por tanto tenemos libertad para moverlas.
- Número de bicicletas en la hora siguiente. Es una previsión de cuantas bicicletas tendremos en una estación a causa del movimiento de los usuarios.
- Demanda. Son las bicicletas que hacen falta en una estación a la hora siguiente.

1.1.2 Las furgonetas

Una furgoneta tiene como propiedades un máximo de 30 bicicletas que puede transportar a la vez en una ruta de hasta tres estaciones (origen y dos destinos).

1.1.3 Análisis de los elementos

Existen diferentes elementos que podemos calcular a partir de estos datos iniciales:

- Las estaciones que más nos interesan que repartan bicicletas son aquellas que tengan más sin utilizar, por tanto tendremos más libertad para moverlas.
- Como tenemos la demanda de la estación y la previsión que tendrá podemos calcular cuantas bicicletas hacen falta para cubrir su demanda.

1.2 Restricciones de la solución

Bicing no nos deja asignar las rutas de cualquier manera, ni obtenemos beneficios con cualquier tipo de transporte, las restricciones que se imponen son:

- Dos furgonetas no pueden coger bicicletas en la misma estación, es decir, dos rutas no tendrán el mismo origen.
- Las furgonetas solo podrán repartir hasta 30 furgonetas repartidas entre los destinos.
- La empresa nos paga un euro por cada bicicleta que enviemos a una estación que se acerque a la demanda, por lo que no nos conviene pasarnos.
- La empresa nos cobra un euro por cada bicicleta que se aleje de la demanda, por lo que no interesa quitar bicicletas a aquellas estaciones que le hagan falta.
- Una ruta no puede ir dos veces a la misma estación.

A partir de aquí ya podríamos ir definiendo las condiciones de aplicabilidad de nuestros operadores una vez estén hechos.

1.2.1 Análisis de las restricciones

Una vez añadidas las restricciones podremos ver diferentes aspectos a considerar a la hora de utilizar los operadores:

- A la hora de crear una ruta hay que comprobar que la estación utilizada no se esté utilizando en otra furgoneta.
- Dentro de cada furgoneta comprobar que no esté llena, ni que ya se haya pasado por esa estación.
- Si a una ruta de una estación ya le repartimos las 30 que se pueden transportar, será inútil añadir más distancia a la ruta.

1.3 Criterios para evaluar la solución

El enunciado define dos criterios para evaluar cómo de buenas son las soluciones:

- Maximización del beneficio debido a llevar bicicletas a estaciones para cubrir su respectiva demanda.
- Minimización del coste debido a la distancia recorrida con el transporte de las bicicletas.

También se nos pide que se implementen dos heurísticas a partir de estos criterios. Uno solo implementa el criterio de cubrir la demanda, y el segundo los dos.

1.3.1 Análisis de los criterios

A partir de estas características podemos definir un poco como construiremos las funciones heurísticas:

- Las bicicletas que transportemos en cada furgoneta serán las que nos proporcionen el beneficio obtenido.
- Las estaciones por las que pasen las furgonetas definirán la distancia recorrida y por tanto su respectivo coste.

1.4 Justificación de Búsqueda Local

El uso de Búsqueda Local se justifica observando que se trata de un problema de optimización combinatoria, en el que se nos pide una solución óptima en base a la maximización y minimización de ciertos criterios, sin importar el camino para llegar a esa solución. Por lo que resolver este problema con un algoritmo determinista resultaría imposible debido a su complejidad, y el uso de otras técnicas como la búsqueda heurística no sería correcto debido al gran tamaño del espacio de búsqueda.

2 Estado del problema y representación

2.1 Descripción y justificación de la representación del problema

La representación del estado en nuestro problema se basa en la asignación a cada furgoneta de las estaciones y las bicicletas que deja en cada una. Por lo tanto, cada estado consistirá de las rutas que hacen las furgonetas repartiendo las bicicletas y el beneficio que supone el recorrido.

A continuación se describen más detalladamente los elementos que intervienen:

- Matriz de enteros ***Furgonetas***, de F filas y 5 columnas, siendo F el número de furgonetas. Donde cada fila representa una de las furgonetas y la ruta de cada una viene dada por las columnas 0, 1 y 2. La columna 0 es la estación de origen, la 1 es la primera que se visita y la 2 es la segunda.
  De esta forma, Furgonetas[f][c] contiene el id de la estación que va a visitar la furgoneta f en la estación de la ruta número c, si c ∈ [0,1,2] (en el caso de que en la ruta no haya una estación por visitar se le asigna el valor -1).
  En el caso de las columnas 3 y 4 contienen el número de bicicletas que deja la furgoneta en la estación de la columna 1 y 2 respectivamente.

- Objeto Estaciones ***Estacions***, es un array que contiene todas las estaciones del problema.

- Double ***Beneficio***, es el beneficio total obtenido en un estado por los traslados de bicicletas efectuados por todas las furgonetas.

Estos elementos nos ayudarán a mantener la información sobre las estaciones en cada estado y el beneficio que obtenemos pero donde realmente se notarán más los cambios será en las furgonetas, ya que es donde se almacenan las rutas creadas y las bicicletas que se transportan, por tanto será donde más se enfoquen los operadores.

Tendremos en cuenta las furgonetas para calcular el espacio de búsqueda, más concretamente la parte de las estaciones. Declaramos N como el número de estaciones y M como el número de furgonetas. Cada furgoneta tiene una combinación de tres furgonetas por lo que sería del orden de **O(M*C(N,3))** pero con excepción de que en cada combinación no puede haber estaciones repetidas en una combinación, ni diferentes combinaciones que empiecen por la misma estación, por lo que el espacio de búsqueda se reduce bastante.

2.2 Representación y análisis de los operadores

Hemos planteado 3 operadores, *AnadirEstacion*, *CambiarOrigen* y *BorrarEstacionRuta*, aunque este último al final no se ha utilizado como operador pero le hemos dado uso dentro del resto.

2.2.1 AnadirEstacion(int f, int e)

Este operador añade una estación *e* a una furgoneta *f* solo si la ruta no está llena y la estación no pertenece a la ruta. Como en nuestras soluciones iniciales (véase en el punto 2.4) ya se asignan los orígenes de la ruta no hace comprobar la restricción de los orígenes. El factor de ramificación de este operador es O(f*e) ya que a todas las furgonetas se le pueden añadir (casi) todas las estaciones. Una funcionalidad extra del operador es que asigna en el vector de furgonetas las bicicletas que repartirá a esa estación y se calcula a partir de las bicicletas no utilizadas, y la demanda que nos facilita la estación y se actualizará las bicicletas que tendrá en la siguiente hora.

2.2.2 BorrarEstacionRuta(int f)

Este operador borraba la última estación de una ruta, pero no se ha acabado implementando ya que este cambio nunca supondría un aumento en el beneficio (se deja de repartir bicicletas)y su factor de ramificación es O(f). Al final se ha implementado para optimizar tareas de cambiar estaciones entre rutas o el cambiar origen. También se devuelven las bicicletas asignadas en la ruta a su origen y se actualizan los parámetros de la estación.

2.2.3 CambiarOrigen(int f, int e)

El siguiente operador cambia el origen de la ruta de la furgoneta *f* por la estación *e* mediante añadir estación y borrar estación, solo si la nueva estación no es origen de la ruta de otra furgoneta. Su factor de ramificación es el mismo que en el operador anterior, O(f*e) porque podemos cambiar todas las furgonetas y probar con todas las estaciones. Mediante las dos funciones anteriores podemos quitar el origen de la ruta y devolver las bicicletas con BorrarEstacionRuta, y que después AnadirEstacion decida la nueva cantidad de bicicletas que serán repartidas.

Estos operadores al final nos dejarán explorar todo el espacio de soluciones y sus condiciones de aplicabilidad nos harán no salirnos de este, para llegar a la solución óptima.

2.3 Análisis de la función heurística

Los factores que intervienen en el problema son:

- La demanda cubierta al llevar las bicicletas
- El número de bicicletas transportadas a cada estación
- La distancia recorrida por las furgonetas
- Beneficio del transporte

Para la primera heurística no se ha tenido en cuenta la distancia recorrida ya que en el enunciado se comentaba que el transporte era gratuito, pero en la segunda se ha tenido que penalizar ese parámetro.

2.3.1 Función heurística simple

Se ha planteado maximizar el beneficio para cubrir la demanda de las estaciones. Por lo tanto únicamente tenemos que comprobar las bicicletas que se han repartido en las rutas para obtenerlo. El heurístico no necesita hacer más trabajo ya que en los propios operadores de añadir estación se calcula el máximo de bicis que se pueden repartir entre las que le sobran a la estación de origen y las que le faltan a la de destino para cubrir totalmente la demanda. El algoritmo de hill climbing comprueba para todas las estaciones accesibles a donde es que se puede repartir más y por tanto maximizar el beneficio, por lo tanto siempre llegaremos a un máximo local

2.3.2 Función heurística combinada

Con esta heurística también se plantea maximizar el beneficio pero el transporte tiene un coste ya especificado en el enunciado. Hay que tener en cuenta que si una ruta tiene dos destinos, el transporte hacia la primera estación contiene las bicicletas que se enviarán a las dos estaciones y después ya solo tendrá la del segundo destino. El cálculo es el mismo que el anterior, sumar las bicicletas enviadas, pero restando el coste del transporte del enunciado. Hay que tener en cuenta que si en una ruta no podemos enviar bicicletas su coste es nulo ya que sería como no enviar. Las condiciones de aplicabilidad nos siguen permitiendo explorar el espacio de soluciones ya que no hay ninguna restricción para la distancia recorrida. Por los mismos algoritmos de búsqueda, siempre buscará la ruta con más beneficio, por lo que se descartan las rutas con recorridos más largos ya que la propia heurística las penalizará.

2.4 Elección y generación de las soluciones iniciales

A la hora de generar nuestros estados/soluciones iniciales tenemos dos opciones. La primera es comenzar desde una solución muy mala que sea fácil de generar, y la otra es generar una solución mejor pero dejando un margen de mejora para que los algoritmos de búsqueda hagan su trabajo.

2.4.1 Solución inicial simple

Consiste en crear rutas asignando cada furgoneta a un origen de forma aleatoria, siempre y cuando no haya ninguna furgoneta asignada todavía a esa estación de origen, y luego para cada furgoneta vamos añadiendo las estaciones de destino de forma aleatoria también, de modo que las furgonetas llevarán bicis a una sola estación, a dos estaciones o bien no tendrán ninguna estación destino, es decir, no tendrán ninguna ruta asignada.

Esta solución es completamente estocástica, con lo cual podemos hacer una primera solución de forma fácil y eficiente que nos permite obtener distintos resultados tras cada ejecución del programa.

2.4.2 Solución inicial “greedy”

En esta solución buscamos un algoritmo voraz que nos permite aumentar las probabilidades de llegar a encontrar el óptimo global o acercarnos bastante a él, pero cabe destacar que se trata de un algoritmo determinista que para una misma entrada siempre va a generar una misma solución inicial.

El primer paso del algoritmo es determinar qué estaciones se encontrarán por debajo de la demanda en la hora siguiente y cuáles no, para poder establecer el origen de las furgonetas en las estaciones en las que realmente se van a poder transportar bicicletas. Si el número de estaciones que pueden prestar bicis es mayor que el número de furgonetas *f*, entonces las furgonetas se asignan a las *f* primeras estaciones. En cambio, si es más pequeño, colocamos todas las furgonetas que podamos a estas estaciones, y las que todavía falten por asignar las colocamos en aquellas estaciones que estarán por debajo de la demanda y no se les asignará ninguna ruta.

Una vez colocadas todas las furgonetas, para crear las rutas de todas las que se encuentren en estaciones que tengan un exceso de bicicletas para la próxima hora, colocamos tantas bicis como necesitemos en para cubrir la demanda de la primera

estación destino y si todavía caben más bicis añadiremos todas las que podamos para la segunda estación destino. En caso de que no podamos llenar la demanda de algunas de estas dos estaciones, la siguiente furgoneta empezará yendo a esa estación.

En ningún caso la furgoneta podrá sobrepasar el límite de 30 bicis y en el caso en que la furgoneta esté llena y no pueda llenar la demanda de la primera estación destino, ya no tendrá ninguna segunda destinación.

3 Experimentación

3.1 Estudio de la combinación de operadores

En este apartado, se realizan múltiples ejecuciones de la búsqueda para la función heurística simple, es decir, la que no tiene en cuenta el coste de los transportes de las furgonetas, en el algoritmo Hill Climbing. En este caso, para la solución inicial hemos decidido escoger la segunda ("Greedy"), ya que es una solución inicial determinista y de esta forma, cambiando las semillas se podrán observar los cambios de una forma más clara.

Observación	El beneficio obtenido y los nodos expandidos de nuestra solución pueden ser influenciados por la combinación de operadores
Planteamiento	Se utilizan diferentes combinaciones de operadores: C1: añadirEstacion C2: añadirEstacion, cambiarOrigen C3: borrarEstacion, añadirEstacion, cambiarOrigen
Hipótesis	Un mayor número de operadores hará que podamos explorar más estados y por lo tanto, llegar a una mejor solución
Método	● Creamos 10 semillas aleatorias ● Para cada semilla, ejecutamos el experimento con la combinación C1, C2 y C3 ● Se utilizan los valores de entrada: 25 estaciones, 1250 bicis y 5 furgonetas para una hora punta ● Se comparan los resultados obtenidos y se identifica cuál es la mejor combinación de operadores

Una vez obtenidos los resultados, como se muestra en la Figura 1 y en la Figura 2, hay una clara diferencia entre el beneficio obtenido en las búsquedas con solo el operador *añadirEstacion* y con las que utilizan una combinación de *añadirEstacion* y *cambiarOrigen* y añadiendo *borrarEstacion.*

	Beneficio			Nodos expandidos		
Semilla	C1	C2	C3	C1	C2	C3
560	70	120	120	4	5	5
1034	69	150	150	3	8	7
23	40	120	120	4	9	7
298	70	120	120	3	6	6
4076	51	120	120	2	6	5
5173	65	120	120	3	7	6
92	61	150	150	3	6	6
11987	62	150	150	5	7	8
321	49	120	120	3	9	9
677	76	120	120	3	5	5

Figura 1: resultados obtenidos con las 3 combinaciones de operadores

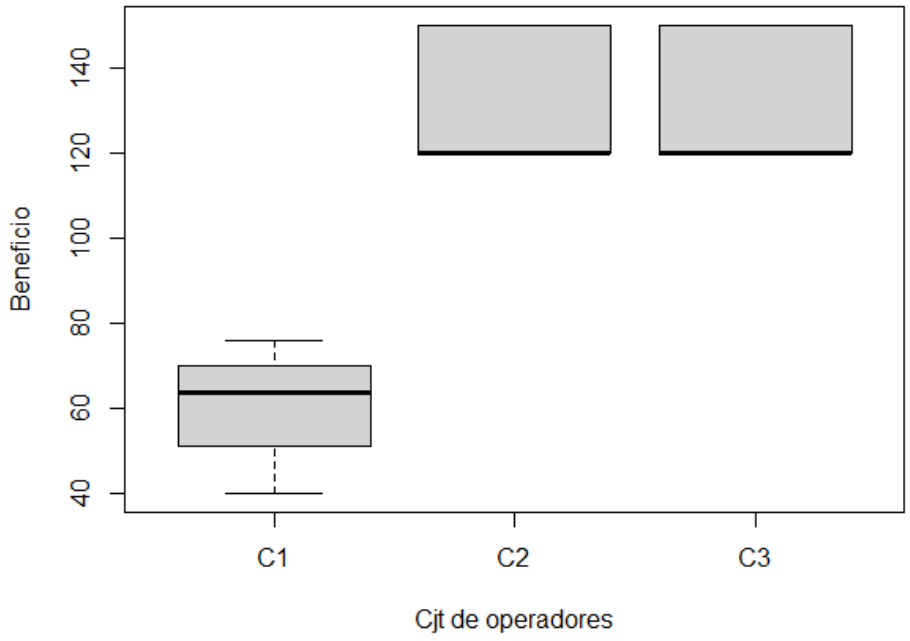


Figura 2: boxplot de los valores del beneficio obtenido por los 3 conjuntos de operadores

Si nos fijamos en la Figura 3, la media del beneficio pasa de ser de 61.3 en el caso de C1 a 129 en el caso de C2 y C3, lo que confirma la hipótesis planteada donde con más operadores se puede explorar un mayor número de estados y obtener una mejor solución.
Como se puede observar, pese a que C2 y C3 obtienen el mismo beneficio, la media de nodos expandidos en C2

	Beneficio			Nodos expandidos		
Combinación	C1	C2	C3	C1	C2	C3
media	61.3	129	129	3.3	6.8	6.4
desv. std	11.31	14.49	14.49	0.82	1.47	1.35

Figura 3: media y desviación estándar del beneficio y los nodos expandidos de cada combinación

Después de analizar los resultados, podemos concluir que la mejor combinación de operadores en este caso es C2, donde usamos los operadores *añadirEstacion y cambiarOrigen.* Ya que pese a obtener los mismos beneficios que C3, al quitar el operador de *borrarEstacion*, la media de nodos expandidos aumenta y por lo tanto puede converger antes a una solución óptima.

3.2 Estudio de las generadoras de la solución inicial

En este caso, el objetivo es determinar la mejor estrategia para generar la solución inicial de las dos que hemos implementado (aleatoria o “greedy”). Como en el anterior apartado, se utilizará el algoritmo Hill Climbing y la función heurística simple.

Observación	El tipo de solución inicial que generamos puede influir en la cantidad de nodos expandidos y el tiempo que tarda, y en el beneficio resultante
Planteamiento	Comparamos los beneficios y los nodos expandidos por la búsqueda para las dos generadoras de soluciones iniciales
Hipótesis	La solución inicial “greedy” debería general mejores resultados (mayor beneficio y más nodos expandidos) que la aleatoria
Método	● Creamos 10 semillas aleatorias ● Para cada semilla, ejecutamos la búsqueda con las dos soluciones iniciales ● Se utilizan los mismos valores de entrada que en el apartado anterior: 25 estaciones, 1250 bicis y 5 furgonetas para una hora punta ● Se comparan los diferentes resultados generados por la búsqueda con las dos soluciones iniciales y se elige el adecuado

	Beneficio		Nodos visitados	
Semilla	Random	Greedy	Random	Greedy
3805	168	150	6	7
44030	90	150	10	7
2328	120	120	8	6
39973	90	150	8	6
27212	30	90	3	6
13911	90	120	4	7
7704	90	120	6	6
44529	120	120	7	5
7584	92	150	5	6
64010	60	150	4	7

Figura 4: resultados obtenidos por las dos generadoras de solución inicial

En la Figura 4, se muestra la tabla con los resultados obtenidos. A primera vista se puede apreciar fácilmente que la solución Greedy produce mejores resultados, para analizarlo con más detalle se proporcionan dos gráficas boxplot: Figura 5 y Figura 6.

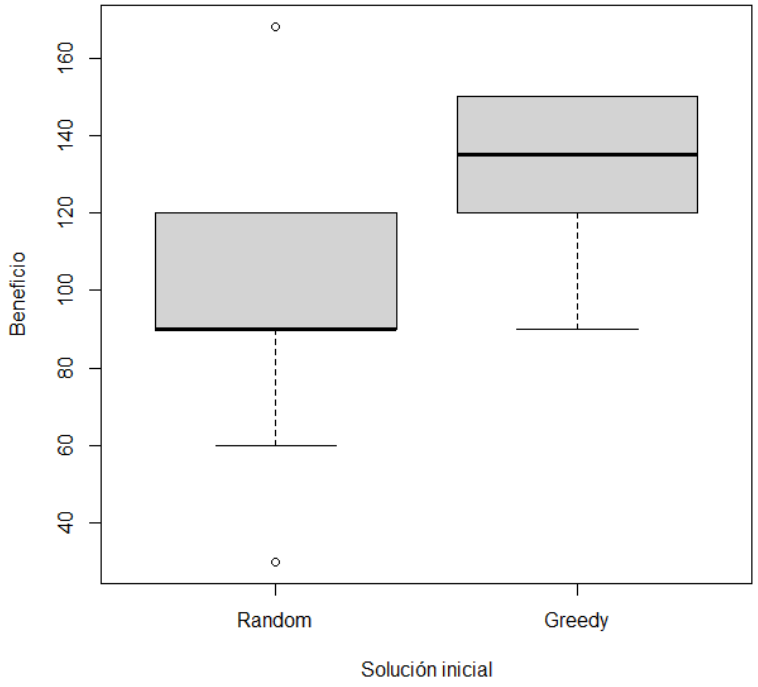


Figura 5: boxplot que compara los beneficios obtenidos por las dos soluciones iniciales

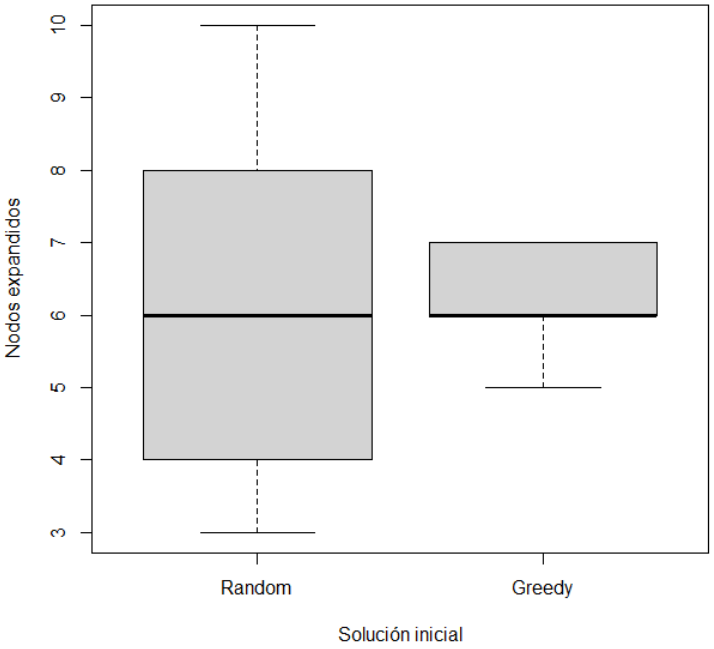


Figura 6: boxplot que compara los nodos expandidos por las dos soluciones

Como se puede observar en la Figura 5, los beneficios obtenidos por las búsquedas que utilizan la solución Greedy, son en su gran mayoría mayores que los que se obtienen con una solución inicial aleatoria, lo que afirma nuestra hipótesis de que con un algoritmo Greedy para la solución inicial se puede obtener mejor beneficio que con una aleatoria.
Cabe recalcar que el hecho de implementar dos soluciones iniciales diferentes no debería influir en el beneficio final (solo influye en el tiempo hasta llegar a la solución óptima), pero en nuestro caso, debido a cómo hemos implementado la solución inicial aleatoria y los operadores que tenemos, es posible que den distintos beneficios.

Además, en la Figura 6, se puede ver como a simple vista expanden una cantidad parecida de nodos, pero en el caso de la solución Greedy, con mucha menos desviación, lo que favorece la experimentación.

	Beneficio		Nodos expandidos	
Sol. inicial	Random	Greedy	Random	Greedy
Media	95	132	6.1	6.3
Desv. std	36.72	20.98	2.18	0.67

Figura 7: tabla con las medias y desviación estándar obtenidas

Si observamos la Figura 7, se puede ver la diferencia entre las medias de beneficio obtenidas en las dos soluciones.
Por lo que teniendo en cuenta que con la **solución inicial Greedy** se obtienen mayores beneficios y que la cantidad de nodos expandidos es casi la misma, pero con mucha menos desviación, como se puede observar en la Figura 7, hemos concluido en escogerla como solución inicial.

3.3 Estudio de los parámetros *K* y λ en Simulated Annealing

Observación	Los valores de K y λ pueden influir en los resultados del algoritmo Simulated Annealing
Planteamiento	Observar la diferencia de resultados para diferentes combinaciones de K y λ
Hipótesis	Se pueden obtener mejores resultados con valores bajos de λ y altos de k
Método	● Se generan 8 semillas diferentes ● Se ejecutan 16 ejecuciones para cada semilla con cada posible combinación entre K y λ, siendo K = {5,50,100,150} y λ = {0.01, 0.1, 0.5, 1.0} ● Se comparan los resultados

Beneficio obtenido por los diferentes valores				
	λ = 0.01	λ = 0.1	λ = 0.5	λ = 1.0
K = 5	144	150	144	150
K = 50	144	150	150	150
K = 100	141	144	150	150
K = 150	144	150	150	150

Figura 8: media del beneficio obtenido para las diferentes combinaciones de valores del Simulated Annealing

Una vez analizados los resultados, se puede observar que los beneficios obtenidos por el algoritmo con los diferentes valores son bastante uniformes. Solo se pueden apreciar diferencias para valores de λ muy pequeños respecto a los otros. En este caso se obtiene menos beneficio de media.
Es por ello, que podríamos refutar la hipótesis planteada al inicio.

Sin embargo, donde sí se ha podido observar una clara diferencia intuyendo una posible regresión lineal, es en la cantidad de nodos expandidos según el valor λ.

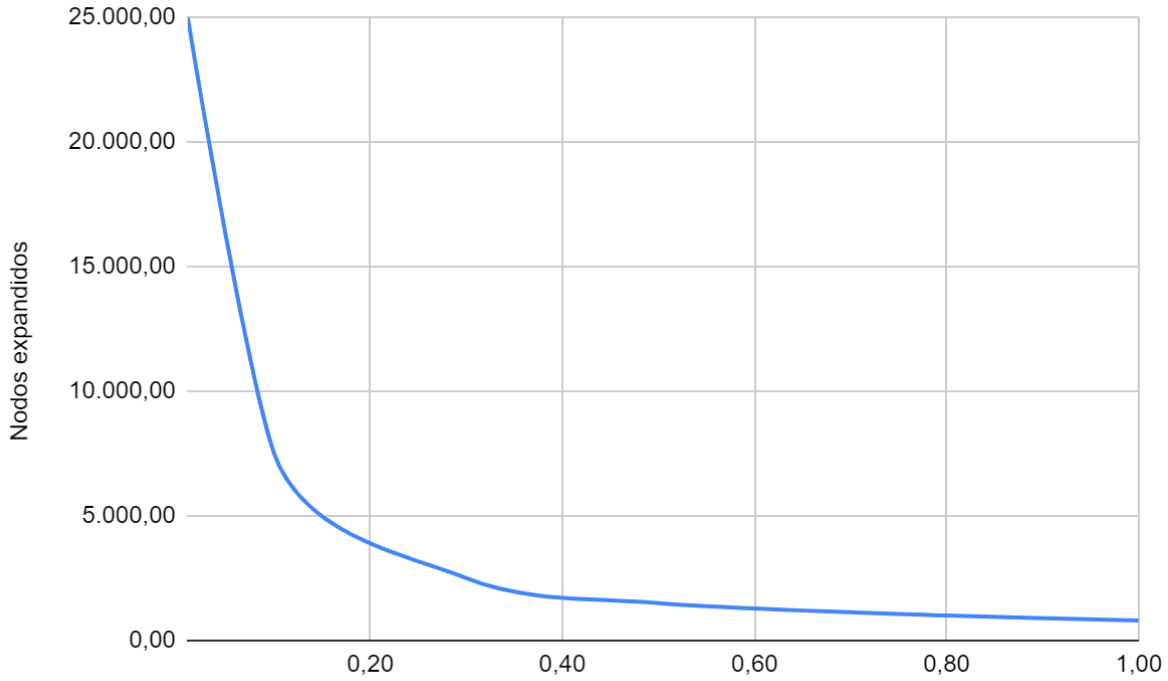


Figura 9: gráfica que muestra la regresión lineal entre el valor λ (eje x) y los nodos expandidos (eje y)

Como se puede ver en la Figura 9, a medida que aumenta el valor de λ, el número de nodos visitados disminuye logarítmicamente. Esto es debido a que λ representa la reducción de la temperatura en cada iteración, por lo que valores más bajos permiten una exploración más amplia del espacio de soluciones, aunque pueden llevar a una convergencia más lenta.

3.4 Evolución del tiempo de ejecución

Observación	El tiempo de ejecución del algoritmo y el beneficio obtenido se puede ver afectado por la cantidad de bicicletas, estaciones y furgonetas
Planteamiento	Generar semillas aleatorias, y para cada una, comprobar cómo varía el tiempo de ejecución y el beneficio según los parámetros.
Hipótesis	Si incrementamos el número de estaciones, bicis y furgonetas, también lo hará el tiempo de ejecución y el beneficio proporcionalmente
Método	• Generar 10 semillas aleatorias • Para cada semilla, realizar ejecuciones aumentando el número de estaciones empezando por 25 hasta 450, y manteniendo la misma proporción del resto de parámetros • Observar y analizar los resultados

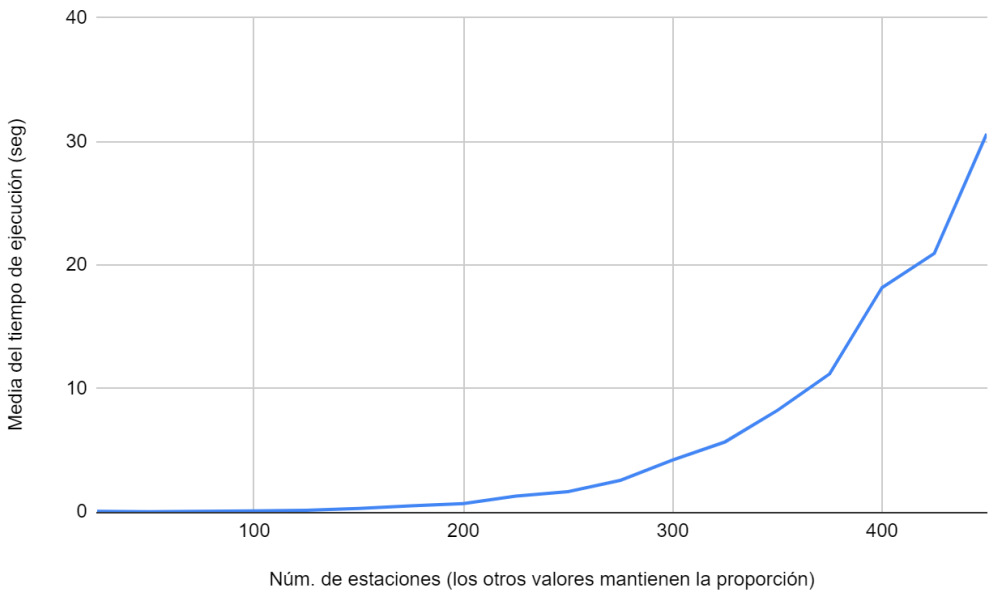


Figura 10: gráfico donde se muestra la media de los tiempos de ejeución según el número de estaciones y los valores proporcionales

Como se puede observar en la Figura 10, la media de tiempo de ejecución de las 10 semillas respecto al número de estaciones del problema y los valores proporcionales a estas, sigue una tendencia exponencial, lo que confirma nuestra hipótesis. Esto es debido a la complejidad del problema y todas las combinaciones posibles. Se podría intuir fácilmente recordando el factor de ramificación de los operadores: O(F*E).

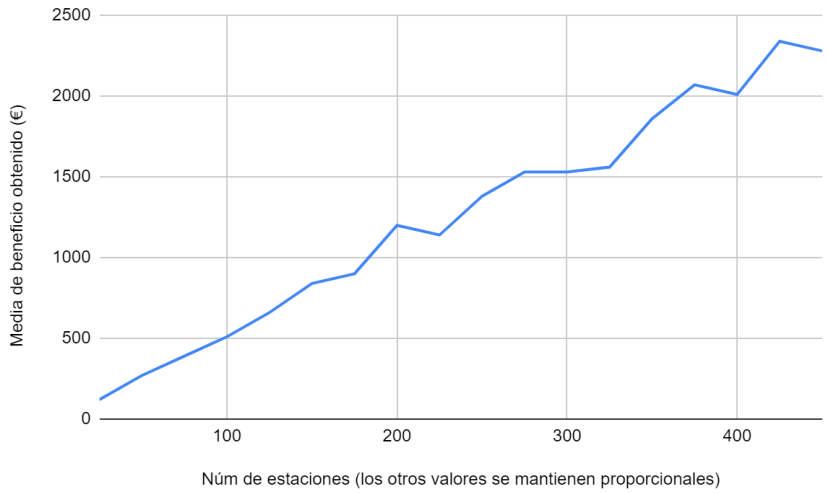


Figura 11: gráfico donde se muestra la media de los tiempos de ejeución según el número de estaciones y los valores proporcionales

Respecto al beneficio, se puede ver en la Figura 11 una tendencia directamente proporcional con el número de estaciones, bicicletas y furgonetas. Como era de esperar, mientras más traslados se puedan efectuar, más dinero se puede obtener.

3.5 Estudio de las funciones heurísticas en los algoritmos Hill Climbing y Simulated Annealing

Observación	Dependiendo de la función heurística que se utilice y en qué algoritmo, factores como el beneficio obtenido, los nodos expandidos o el tiempo de ejecución pueden varias
Planteamiento	Para cada combinación entre los dos algoritmos y las dos funciones heurísticas, medir los diferentes valores y compararlos
Hipótesis	Los dos algoritmos obtienen beneficios parecidos pero si se utiliza la función heurística que tiene en cuenta el coste de transporte se obtiene menos beneficio
Método	● Se generan 10 semillas aleatorias ● Para cada posible combinación de (Algoritmo - F. heurística), se ejecuta el programa con las 10 semillas y se hace una media del beneficio obtenido y el tiempo de ejecución ● Se analizan y comparan los resultados

Beneficio	HC	SA
F. heurística simple	94.3	117
F. heurística combinada	24.4	58.5

Figura 12: Tabla con la media del beneficio obtenido en las diferentes combinaciones

Texe (ms)	HC	SA
F. heurística simple	94.3	117
F. heurística combinada	24.4	58.5

Figura 13: Tabla con la media de tiempo de ejecución obtenido en las diferentes combinaciones

Si nos fijamos en los valores de las combinaciones con la función heurística combinada, se puede ver como el beneficio es mucho menor respecto a la heurística simple, ya que ésta también tiene en cuenta los costes de traslados de las furgonetas y por tanto disminuye el beneficio, hecho que afirma una parte de la hipótesis que habíamos planteado.

En cuanto al tiempo de ejecución, también se puede observar una clara diferencia entre las dos funciones heurísticas, siendo la combinada mucho mejor que la simple. Esto puede ser debido a que al ser una función heurística más compleja, es decir, que evalúa más criterios, la búsqueda es más guiada y converge antes hacia una solución óptima.

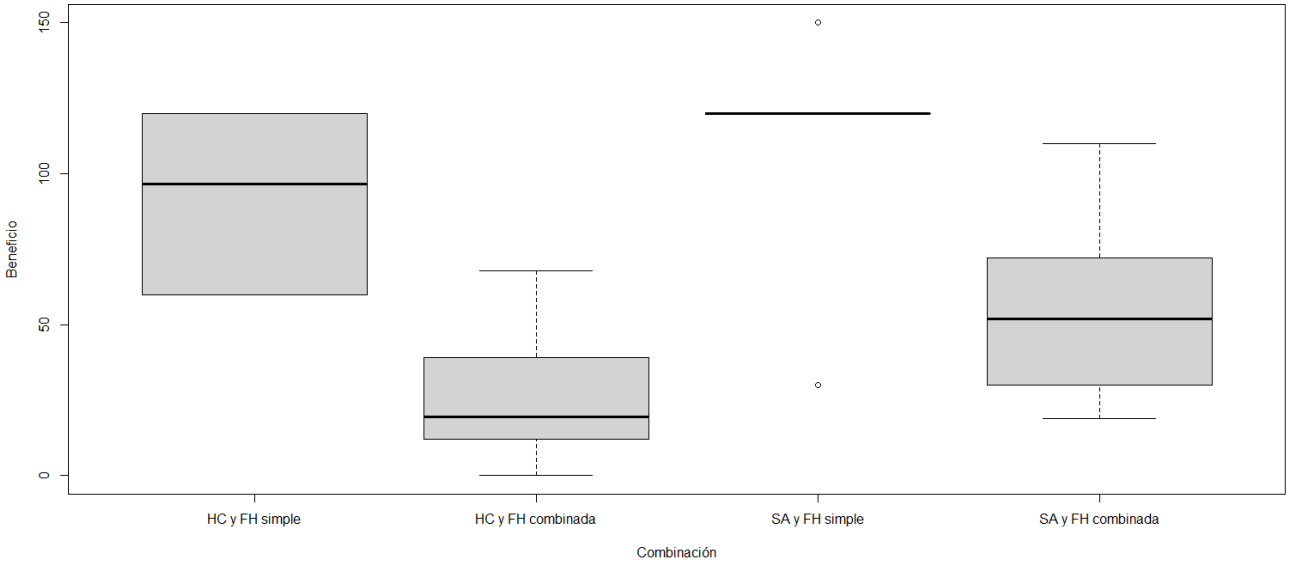


Figura 14: boxplot que compara el beneficio entre las diferentes combinaciones

Si observamos el boxplot de la Figura 14, podemos ver y analizar una clara comparativa entre los beneficios obtenidos por las 4 posibles combinaciones.
Aquí se observa claramente que la combinación de Simulated Annealing con la función heurística simple, obtiene la mejor media de beneficio y además con valores muy concentrados. Si analizamos las funciones heurísticas combinadas (combinan todos los criterios), en este caso también obtiene mejor beneficio Simulated Annealing.

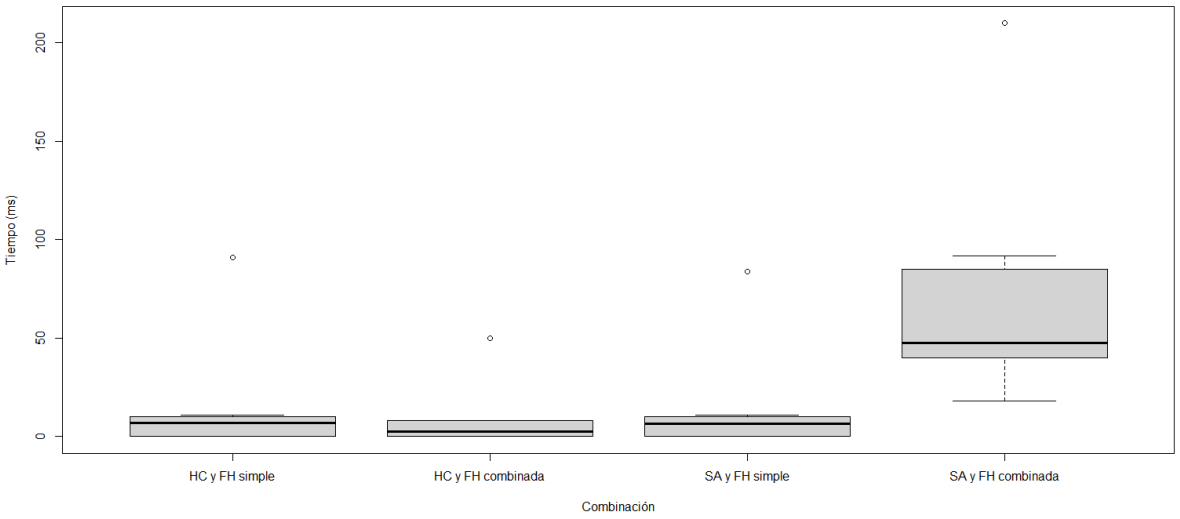


Figura 15: boxplot que compara el tiempo de ejecución entre las diferentes combinaciones

En el caso del tiempo de ejecución, si analizamos el boxplot de la Figura 15, podemos ver que Hill-Climbing con las dos funciones heurísticas y Simulated Annealing con la heurística simple, tienen tiempos de ejecución muy similares y cercanos a 0 ms, a excepción de alguna muestra puntual.
En cambio, Simulated Annealing con la función heurística combinada tiene una media de tiempo de ejecución bastante mayor respecto a las demás combinaciones.

3.6 Estudio del tiempo de ejecución en hora punta

Observación	El hecho de que la demanda se encuentre o no en hora punta puede afectar al tiempo de ejecución
Planteamiento	Para diferentes semillas, ejecutar el problema cuando la demanda está hora punta y cuando no lo está y comparar el tiempo de ejecución (utilizando los mismos valores de estaciones, furgonetas y bicis para que la comparación sea correcta.
Hipótesis	El tiempo de ejecución con la demanda en hora punta será mayor que cuando no lo está
Método	● Generar 10 semillas aleatorias ● Se utiliza el algoritmo Hill Climbing y la función heurística simple ● Para cada semilla, realizar una ejecución en hora punta y otra en hora normal (todas con los valores del primer escenario) ● Comparar y analizar los resultados

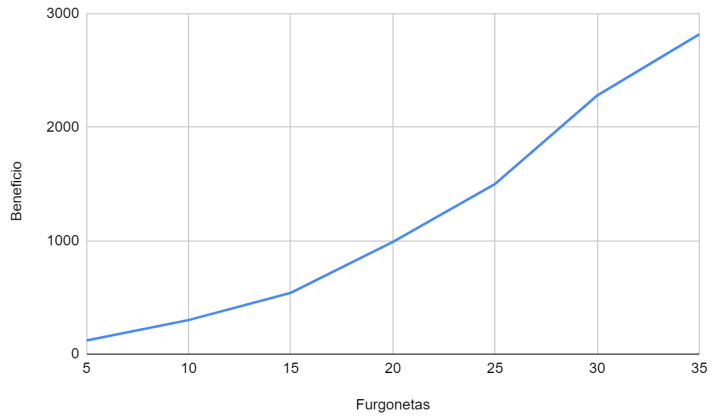
	Tiempo (segundos)	
Seed	Hora punta	Hora equilibrada
234	0.003	0.054
531	0.014	0.046
567	0.002	0.05
23	0.004	0.049
452	0.005	0.048
7286	0.004	0.05
923	0.004	0.053
683	0.005	0.051
2341	0.003	0.05
622	0.007	0.048

Figura 16: tabla de tiempos de ejecución con hora punta y hora equilibrada para las diferentes semillas

En la Figura 16, se puede ver como los tiempos de ejecución cuando la demanda está en hora punta son mucho más pequeños que cuando no lo está.
Estos resultados refutan completamente la hipótesis planteada al inicio, esto se podría deber a que cuando la demanda se encuentra en una hora equilibrada, prácticamente todas las estaciones tienen la demanda cubierta. Por lo que al utilizar la función heurística que solo busca maximizar el beneficio sin tener en cuenta el coste del transporte, a Hill-Climbing le cuesta más encontrar soluciones mejores (ya que no puede mejorar el beneficio) y por lo tanto tarda más en converger en una solución óptima.
Cabe recalcar que la anterior justificación es solo una suposición, y por lo tanto lo único que podemos afirmar es el hecho de refutar la propia hipótesis inicial.

3.7 Estudio del número de furgonetas óptimo

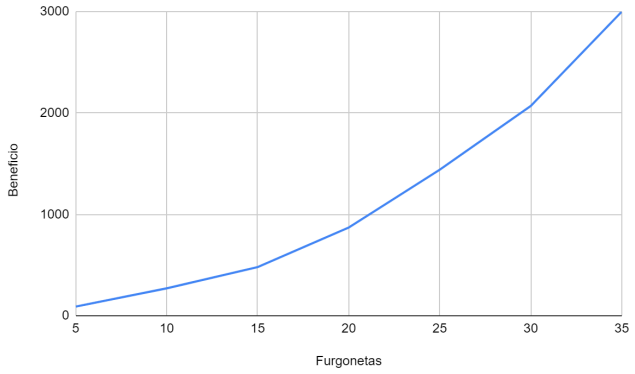
Observación	El número de furgonetas utilizadas puede influir en el beneficio obtenido
Planteamiento	Realizar múltiples ejecuciones de Hill-Climbing, incrementando el número de furgonetas utilizadas y observar cómo cambia el beneficio resultante. Hacer lo mismo con la demanda en hora punta
Hipótesis	Incrementar el número de furgonetas proporciona una mejor solución
Método	• Escoger una semilla aleatoria • Realizar 7 ejecuciones incrementando el número de furgonetas en 5 • Analizar y comparar resultados • Repetir el proceso con demanda en hora punta



Furgonetas	Beneficio
5	120
10	300
15	540
20	990
25	1500
30	2280
35	2820

Figuras 17 y 18: gráfico y tabla de los beneficios obtenidos con los diferentes números de furgonetas para la demanda en hora equilibrada

Como se puede observar en las Figuras 17 y 18, el beneficio obtenido es proporcional al número de furgonetas utilizadas como era de esperar, de forma que se afirma la hipótesis planteada



Furgonetas	Beneficio
5	90
10	270
15	480
20	870
25	1440
30	2070
35	3000

Figuras 19 y 20: gráfico y tabla de los beneficios
obtenidos con los diferentes números de furgonetas para la demanda en hora punta

En el caso de la demanda en hora punta, como se puede ver en las Figuras 19 y 20 sigue la misma tendencia que en la hora equilibrada, es decir, el beneficio resultante es directamente proporcional al número de furgonetas utilizadas.

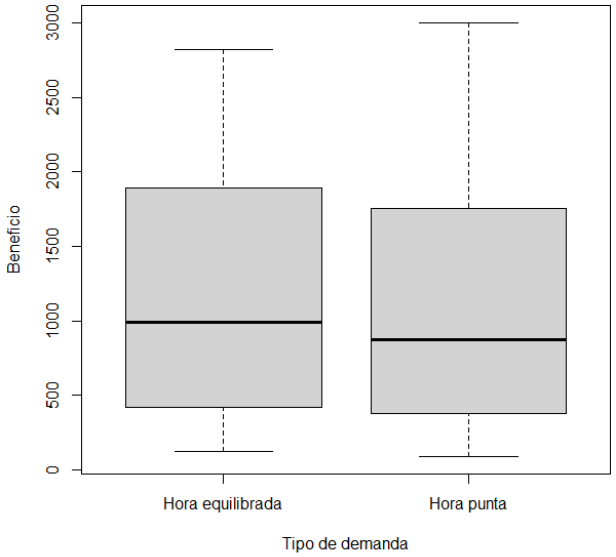


Figura 21: boxplot que compara los beneficios obtenidos estando en hora punta y equilibrada con las diferentes cantidades de furgonetas

Si se analiza la Figura 21, se puede ver como los beneficios de las diferentes cantidades de furgonetas en hora punta son ligeramente inferiores respecto a la hora equilibrada.

3.8 Comparación entre Hill-Climbing y Simulated Annealing

Después de haber realizado los experimentos anteriores ahora vamos a comparar los algoritmos Hill Climbing y Simulated Annealing para concluir cuál de los dos resulta más eficiente para resolver el problema de este examen.

Para comparar los dos algoritmos vamos a analizar su coste temporal durante la búsqueda y la bondad de las soluciones obtenidas. Tal y como hemos observado en el experimento 3, después de haber explorado los valores de los parámetros utilizados en el Simulated Annealing, hemos llegado a la conclusión que los valores que nos permiten maximizar la eficiencia del algoritmo son $K = 150$ y $\lambda = 1.0$.

Para la comparación de coste temporal y de bondad de las soluciones haremos referencia a los resultados que hemos obtenido en el experimento 5.
Por un lado, respecto a la bondad de las soluciones podemos observar que Simulated Annealing es superior a Hill Climbing ya que, en media, el beneficio que obtenemos es mayor y por tanto nos ofrece una solución más cercana a la “óptima”.
Por otro lado, en cuanto a eficiencia temporal podemos deducir que Hill Climbing es mejor que Simulated Annealing ya que Hill Climbing tiene un tiempo de ejecución muy pequeño usando tanto el heurístico simple como el combinado mientras que el Simulated Annealing obtiene tiempos parecidos para el heurístico simple, pero resulta más costoso usando el combinado.

Por tanto, podemos concluir que con Simulated Annealing obtenemos mejores resultados pero usando Hill Climbing obtendremos la mayor eficiencia temporal al ejecutar nuestro programa.

4 Proyecto de innovación

4.1 Descripción del tema

Alexa es un asistente virtual controlado por voz que fue desarrollado por Amazon. Utilizando inteligencia artificial Alexa responde a todas las preguntas del usuario y permite interactuar con la tecnología que usamos las personas a diario.

4.2 Reparto del trabajo

Para llevar a cabo la recolección de información para este trabajo no nos hemos dividido el trabajo entre los 3, sino que hemos preferido buscar todos información por nuestra cuenta sin restringirnos a ningún tema específico para así poder encontrar la mayor variedad de información posible.

4.3 Referencias

[1] “What Is Alexa and What Does Alexa Do?” makeuseof.com. Fecha de acceso: 7 de octubre de 2023. https://www.makeuseof.com/what-does-alexa-do/

[2] "How is Alexa using artificial intelligence?" medium.com. Fecha de acceso: 7 de octubre de 2023. https://medium.com/bestai/how-is-alexa-using-artificial-intelligence-02dfec4bbb5d

[3] “How Amazon Alexa works? Your guide to Natural Language Processing (AI)” towardsdatascience.com. Fecha de acceso: 9 de octubre de 2023. https://towardsdatascience.com/how-amazon-alexa-works-your-guide-to-natural-language-processing-ai-7506004709d3

[4] “AI for Noobs: How Amazon Alexa Works” hackernoon.com. Fecha de acceso: 10 de octubre de 2023. https://hackernoon.com/ai-for-noobs-how-amazon-alexa-works

[5] “Is Alexa an AI?” itchronicles.com. Fecha de acceso: 10 de octubre de 2023. https://itchronicles.com/artificial-intelligence/is-alexa-an-ai/

[6] “How does Alexa use Artificial Intelligence?” aihubspark.com. Fecha de acceso: 10 de octubre de 2023. https://aihubspark.com/how-does-alexa-use-artificial-intelligence/

[7] “How Does Amazon Alexa Algorithm Work“ codecondo.com. Fecha de acceso: 13 de octubre de 2023. https://codecondo.com/how-does-amazon-alexa-algorithm-work/

[8] “The Technology In Amazon Alexa – The Tech Behind Series” intuji.com. Fecha de acceso: 13 de octubre de 2023. https://intuji.com/the-tech-behind-amazon-alexa/

[9] “The Real Reasons That Amazon’s Alexa May Become The Go-To AI For The Home” fastcompany.com. Fecha de acceso: 15 de octubre de 2023. https://www.fastcompany.com/3058721/the-real-reasons-that-amazons-alexa-may-become-the-go-to-ai-for-the-home

[10] “Alexa at five: Looking back, looking forward” amazon.science. Fecha de acceso: 15 de octubre de 2023. https://www.amazon.science/blog/alexa-at-five-looking-back-looking-forward

4.4 Dificultades

La mayor dificultad que nos hemos encontrado ha sido buscar cuáles son las distintas técnicas de inteligencia artificial que utiliza Alexa.