Práctica 1: Búsqueda Local

Inteligencia Artificial

Laia Ondoño laia.ondono@estudiantat.upc.edu
Anna Llanza anna.llanza@estudiantat.upc.edu
Paula Gené paula.gene@estudiantat.upc.edu

QT 2020-2021 02/11/2020

Índice

Introducción

Con el objetivo de conocer y estudiar los algoritmos de búsqueda local, hemos realizado esta práctica centrada en la ejecución y análisis de los algoritmos Hill Climbing y Simulated Annealing, facilitados por la librería AIMA. Para poder tener una noción básica sobre estos, hemos resuelto un problema basado en la optimización logística de una empresa de distribución de productos on-line.

Durante el desarrollo de esta práctica, hemos analizado los componentes principales del problema para poder integrarlos correctamente en cada uno de los pasos. Por otro lado, hemos estudiado los conceptos relacionados con los algoritmos y las librerías, tanto la de AIMA como la de Ázamon, que nos han proporcionado. A continuación, hemos pensado la representación y elementos del problema, diseñado la estructura de datos del estado y su implementación, y los operadores que nos permiten navegar entre estados. También hemos tenido que analizar y elegir una estrategia generadora de soluciones iniciales y la creación de funciones heurísticas que permitan una correcta evaluación de la calidad de los estados. Por último, hemos llevado a cabo una serie de experimentos para poner a prueba las diferentes estrategias consideradas y una gran cantidad de variables que modifican la solución final que se obtiene, para poder sacar conclusiones.

Identificación del problema

En esta práctica, se nos presenta un problema con el cual debemos conseguir una optimización logística para Ázamon, una compañía ficticia basada en la distribución on-line. Ésta nos ha pedido un estudio sobre la aplicación de algoritmos de búsqueda local para encontrar una solución al siguiente problema: obtener para todos los paquetes a enviar en un día y a una ciudad concreta, una asignación a las ofertas de envío de ese día de las compañías de transporte.

Esta empresa ofrece ciertas diversas prioridades para su paquete, cada una en función de los días en que el paquete será recibido. La prioridad *día siguiente* asegura que el paquete se recibirá, tal y como el nombre indica, en un día. La prioridad *2-3 días* indica que el paquete llegará, cómo máximo, en un plazo de 3 días. Finalmente tenemos la prioridad *4-5 días*, la cual asegura que el paquete será entregado cómo máximo en 5 días. Cada prioridad tiene un coste asociado por paquete de 5 euros, 3 euros y 1,5 euros respectivamente.

Debemos tener en cuenta que los paquetes no se recogen inmediatamente, es decir, en algunos casos estos deberán de almacenarse hasta ser recogidos por los transportistas, cosa que supone un coste de almacenamiento por kilogramo de 0,25 euros cada día. Aun así, también se puede dar el caso contrario: que los paquetes sean enviados antes de la fecha prevista. Ésto beneficia a la compañía, dado que hace que sus clientes sean más felices y, consecuentemente, compran más en el futuro. Para este problema, cuantificaremos la felicidad de una solución contando con cuántos días de adelanto llegan los paquetes.

Encontrar la solución a este problema requiere aplicar una búsqueda local. Esencialmente, buscamos encontrar la mejor de entre las soluciones posibles en un tiempo razonable. Empezamos teniendo una solución inicial y realizamos modificaciones locales que hacen que ésta mejore. Para cada solución, o estado, buscamos cuál de los estados “vecinos” (posibles estados a los que se puede pasar desde el estado actual) supone una mayor mejora respecto al padre, y entonces realizamos el cambio para pasar a dicho estado. Este cambio se lleva a cabo aplicando diferentes operadores sobre el estado actual. En este contexto, una solución es una serie de asignaciones de paquetes a las ofertas de los transportistas en un día para una ciudad. Un estado vecino será una asignación en la que habrá algún cambio en la distribución de paquetes a las ofertas respecto a la asignación actual.

Para decidir a qué estado es mejor pasar, tenemos una función que evalúa la calidad de la solución, conocida cómo función heurística, que aplicamos a los posibles estados sucesores. Ésta combina diferentes elementos del problema, en este caso el coste de transporte y almacenamiento y la felicidad, para ver qué estado vecino tiene los mejores

valores para cada elemento. Este problema busca minimizar el coste y maximizar la felicidad de los clientes.

Estado y representación del problema

Si queremos poder obtener una solución al problema planteado, es necesario la implementación del Estado y que en esta se disponga de los atributos imprescindibles para representarlo. Nuestra implementación define el Estado con los siguientes atributos:

- *paquetes*: representan los paquetes a entregar que deben asignarse a las ofertas. Para implementarlo hemos usado el objeto de la clase Paquetes de la librería IA.Azamon, en la cual es necesario indicarle el número de paquetes y la semilla que queremos usar.
- *ofertas*: representan las ofertas de transporte de los paquetes. Para implementarlo hemos usado el objeto de la clase Transporte de la librería IA.Azamon, creándolo indicando los paquetes, la proporción del peso respecto al total y la semilla que queremos utilizar.
- *asignacion*: representa la asignación de los paquetes a las ofertas correspondientes. Para implementarlo hemos usado ArrayList<Integer> de tamaño el número de paquetes, la cual cada posición de la lista es cada uno de los paquetes. Si estos están asignados a alguna oferta, el valor guardado es el índice correspondiente de ésta en el atributo *ofertas*. Por otro lado, si el paquete no está asignado a ninguna oferta, el valor guardado es -1.
- *capacidad*: representa la capacidad ocupada de los paquetes que tiene cada oferta, es decir, la capacidad de cada oferta indica la suma de los pesos de los paquetes asignados a esa oferta. Para implementarlo hemos usado una ArrayList<Double> que tiene el mismo tamaño que el atributo ofertas.
- *precio*: representa el precio total (double), es decir, el coste de almacenamiento y transporte total para el estado actual. Este se calcula con la función *calcularPrecio*() disponible en la clase *Estado*.
- *felicidad*: representa la felicidad (int) del estado actual. Esta se calcula con la función *calcularFelicidad*(), la cual devuelve la suma de las diferencias (positivas) entre los días mínimos de llegada del paquete según su prioridad y el tiempo que ha tardado en ser entregado.

Además, para poder realizar los experimentos, hemos añadido los atributos *operadores* (int) y *ponderacion* (int), los cuales indican que operadores queremos usar (mover o intercambiar o mover y intercambiar) y el valor para la ponderación de la felicidad en la función heurística 2 respectivamente.

Hemos decidido implementar el Estado de esta manera, porque así facilitamos el proceso de los operadores mover e intercambiar paquetes y obtenemos un tiempo de ejecución reducido. Esto se debe a que para cambiar de oferta un paquete, solo es necesario modificar el valor correspondiente de la posición del paquete en el atributo *asignacion* con el correspondiente índice de las ofertas. Los otros atributos también los hemos visto

necesarios para evitar recalcular la capacidad de cada oferta, el precio y la felicidad y por lo tanto necesitar más tiempo, cada vez que realicemos algún cambio. En referencia al tamaño del espacio de búsqueda, este corresponde al cálculo siguiente: nPaquetes^nOfertas

Representación y análisis de los operadores y estados sucesores

Inicialmente pensamos en tener cuatro operadores (añadir paquete, quitar paquete, mover paquete e intercambiar paquetes). Después, sin embargo, nos dimos cuenta de que usar los operadores añadir y quitar paquete no tenía sentido porque se podía dar que un estado no tuviera todos los paquetes asignados a alguna oferta, cosa que incumple condiciones del problema planteado. Los otros operadores, que consisten en mover un paquete e intercambiar dos paquetes, ya permiten recorrer todo el espacio de búsqueda, es decir, generar cualquier solución, con lo que finalmente nos decantamos por incluir estos dos.

Por un lado, tenemos el operador *moverPaquete(p, o)* donde p es un identificador del paquete que se quiere mover a la oferta identificada por el entero o. Para este operador, hemos implementado un método que devuelve un booleano que será cierto si este cambio se puede realizar correctamente y falso en caso contrario. Se podrá mover el paquete siempre y cuando la oferta o tenga una capacidad libre mayor que el peso del paquete, para así evitar sobrepasar el peso máximo de la oferta y los días de entrega de la oferta de transporte estén dentro del límite de la prioridad de entrega correspondiente al paquete. Si estas condiciones se cumplen, entonces se asigna el paquete a la nueva oferta, se actualiza la capacidad de las dos ofertas (origen y destino) y se calcula el coste y la felicidad del estado una vez aplicados estos cambios.

El operador mover paquete permite que, desde un estado, podamos mover cualquier paquete a cualquier oferta, con los que hay, como máximo, tantos estados vecinos cómo número de paquetes multiplicado por número de ofertas. Por lo tanto, este operador tiene un factor de ramificación de nPaquetes * nOfertas.

El otro operador que hemos incluido es *intercambiarPaquetes(p1, p2)* donde tanto p1 como p2 hacen referencia al entero que identifica los paquetes a intercambiar. La estructura del método es la misma que la del operador anterior. En este caso, las condiciones de aceptar dicho intercambio consisten en asegurar que un paquete quepa en la oferta a la que pertenece el otro restando el peso de este segundo paquete, dado que si se realiza el intercambio no seguirá en la oferta. Si ambos paquetes caben en la oferta del otro paquete, pasamos a mirar las prioridades de los paquetes para ver si éstas permiten ser asignados a la oferta a la que pertenece el otro paquete.

En este caso, nuestra implementación de intercambiar paquete permite realizar un intercambio con cualquier paquete. Es por ello que el factor de ramificación de de (nPaquetes * (nPaquetes - 1))/2.

Análisis de las funciones heurísticas

Para poder priorizar diferentes atributos dependiendo del interés de cada usuario, hemos implementado dos funciones heurísticas:

- *FuncionHeuristica1*: el valor que utiliza como heurístico es el coste de almacenamiento y transporte de los paquetes, el cual se obtiene de la función *getPrecio*(), implementada en la clase Estado. Es por eso que implementar este heurísitico nos interesa para cuando el usuario solo quiera priorizar la obtención de un precio lo más mínimo posible en la solución final.
- *FuncionHeuristica2*: el valor que utiliza como heurístico es el coste de almacenamiento y transporte de los paquetes reducido con la ponderación de la felicidad de los usuarios, la cual se obtiene de la función *getFelicidad*(), implementada en la clase Estado. Dado que en este heurístico nos interesa tanto priorizar el precio como la felicidad, debíamos encontrar un valor adecuado para la ponderación en el que la felicidad se maximice, pero sin que afecte demasiado al precio. Después de realizar experimentos y observar los resultados para encontrar el mejor valor para la ponderación hemos decidido utilizar un valor diferente para la ponderación según el algoritmo usado:
    - Hill Climbing: precio - 11.0 * felicidad
    - Simulated Annealing: precio - 10.0 * felicidad

Elección y generación del estado inicial

Para la ejecución de los algoritmos elaboramos dos funciones generadoras de soluciones iniciales usando diferentes estrategias. Para considerarse una solución válida hay que tener en cuenta que debe cumplirse que todos los paquetes estén asignados a una oferta, que la suma de los pesos de los paquetes asignados a la oferta no supere la capacidad máxima de esta, y que el límite de días de entrega especificado en la oferta sea inferior o igual a los días de entrega exigidos por la prioridad de los paquetes asignado a esta. Para justificar nuestras elecciones, hemos decidido centrar cada una de las dos estrategias generadoras de soluciones iniciales en uno de los dos conceptos que se mencionan para juzgar la solución obtenida del problema: la felicidad y el coste, respectivamente. Las dos estrategias utilizadas son las siguientes:

- **Solución inicial 1:** esta primera estrategia se basa en la idea de maximizar la felicidad de los clientes, y se da la máxima prioridad posible a los paquetes, en referencia a las ofertas que se generan, siempre cumpliendo las restricciones. En primer lugar, se ordenan los paquetes por prioridad, de más alta a más baja. Por otro lado, se ordenan las ofertas por los días de entrega, de menos días a más. Cuando tenemos ambas listas ordenadas, se recorre la de los paquetes, y para cada uno de estos, se recorre la lista de ofertas ordenada, y se van asignando los paquetes a las ofertas donde quede suficiente capacidad disponible. De esta manera, se cumplen todas las restricciones y en general, los paquetes quedan asignados a ofertas con su misma o mayor prioridad. Esto provoca que aumente el coste debido a que las ofertas utilizadas son las más caras, pero se reducen costes de almacenamiento, y se genera una gran cantidad de felicidad por parte de los clientes. A continuación, el algoritmo ya se encarga de reducir este precio final, siguiendo la función heurística que se le indique.

- **Solución inicial 2:** esta segunda estrategia se centra en minimizar el coste de transporte, por lo que en un primer momento siempre se asignan los paquetes a ofertas con su misma prioridad. Para obtener una primera solución inicial de una manera rápida pero eficiente, simplemente se recorre la lista de paquetes y se asigna aleatoriamente a una de las ofertas generadas que tenga la misma prioridad que este, y que tenga suficiente capacidad. Podríamos haber determinado cuáles son las ofertas de menor precio dentro de una misma prioridad y asignado los paquetes con referencia a esto, pero decidimos que implicaría un aumento en el tiempo de ejecución, y era innecesario ya que el algoritmo ya se encarga de hacer ciertas modificaciones posteriormente. Con esta estrategia conseguimos minimizar el coste de transporte en comparación con la anterior, pero aumenta el coste debido al almacenamiento, y no obtenemos ninguna felicidad por parte de los clientes.

Experimentación

Experimento 1

Determinar qué conjunto de operadores da mejores resultados para una función heurística que optimice el primer criterio con un escenario en el que el número de paquetes a enviar es 100 y la proporción del peso transportable por las ofertas es 1, 2. Deberéis usar el algoritmo de Hill Climbing. Escoged una de las estrategias de inicialización de entre las que proponéis. A partir de estos resultados deberéis fijar los operadores para el resto de experimentos.

Observación	Pueden haber conjuntos de operadores que obtienen mejores soluciones.
Planteamiento	Escogemos diferentes conjuntos de operadores y observamos sus soluciones para optimizar el primer criterio.
Hipótesis	Ho: Todos los conjuntos de operadores son igual de óptimos. H1: Hay algunos conjuntos de operadores mejores que otros.

Método	- Elegimos como sujetos los siguientes conjuntos de operadores:
	  - Mover paquete
	  - Intercambiar paquete
	  - Mover paquete e intercambiar paquete
	- Elegimos 10 semillas aleatorias, una para cada réplica.
	- Ejecutamos 3 experimentos por semilla, 1 para cada sujeto.
	- Experimentamos con problemas de 100 paquetes y una proporción del peso transportable por las ofertas es 1,2.
	- Usamos la estrategia de inicialización 2.
	- Usamos la función heurística 1, que minimiza el precio.
	- Usamos el algoritmo de Hill Climbing.
	- Medimos el coste de transporte y almacenamiento de cada conjunto de operadores y el tiempo de ejecución del algoritmo para realizar la comparación.

Resultados:

	Coste (€)			Tiempo (ms)		
Número réplica	Mover	Intercambiar	Mov + Int	Mover	Intercambiar	Mov + Int
1	1.067,34	1087,04	1.093,425	149	85	22
2	1103,02	1107,61	1115,5	19	29	140
3	868,16	868,45	861,75	11	190	41
4	1.035,70	1036,78	1034,74	11	15	128
5	961,93	978,2	964,75	82	76	36
6	815,96	830,41	824,58	23	52	56
7	872,41	887,35	878,2	13	14	100
8	951,91	954,18	950,56	2	5	15
9	909,75	918,37	912,35	18	8	14
10	999,39	1023,82	1010,44	3	3	123
Media resultados	958,56	969,221	964,630	33,1	47,7	67,5

Tabla 1: Coste y tiempo de ejecución para los diferentes operadores

Conclusiones:

Con los resultados obtenidos, hemos podido observar que utilizar solamente el operador Mover paquete nos permite obtener el mínimo coste, y además, también obtenemos la ejecución más rápida. Es por eso que hemos decidido fijar para el resto de ejecuciones la configuración de los operadores solamente con el de Mover paquete.

Experimento 2

Determinar qué estrategia de generación de la solución inicial da mejores resultados para la función heurística usada en el apartado anterior, con el escenario del apartado anterior y usando el algoritmo de Hill Climbing. A partir de estos resultados deberéis fijar también la estrategia de generación de la solución inicial para el resto de experimentos.

Observación	Las dos soluciones iniciales pueden no ser igual de buenas.
Planteamiento	Probamos las dos soluciones iniciales y observamos los resultados con cada una.
Hipótesis	Ho: Las dos soluciones son igual de óptimas. H1: Hay una solución inicial que es mejor que la otra.

Método	- Elegimos como sujetos las siguientes soluciones iniciales:
	  - Solución inicial 1 (prioriza la felicidad)
	  - Solución inicial 2 (prioriza el precio)
	- Elegimos 10 semillas aleatorias, una para cada réplica.
	- Ejecutamos 2 experimentos por semilla, 1 para cada sujeto.
	- Experimentamos con problemas de 100 paquetes y una proporción del peso transportable por las ofertas de 1,2.
	- Usamos la función heurística 1, que minimiza el precio.
	- Usamos solamente el operador Mover paquete.
	- Usamos el algoritmo de Hill Climbing.
	- Medimos el coste de transporte y almacenamiento de cada sujeto y el tiempo de ejecución del algoritmo para realizar la comparación.

Resultados:

	Coste (€)		Tiempo (ms)	
Número réplica	Solución inicial 1	Solución inicial 2	Solución inicial 1	Solución inicial 2
1	1.000,78	1.015,73	182	82
2	876,05	874,1	18	18
3	975,83	945,55	35	83
4	1.148,17	1.144,02	25	46
5	980,75	970,54	13	17
6	960,17	959,73	57	15
7	1.117,23	1.116,87	20	29
8	987,93	952,08	31	45
9	976,80	963,12	22	27
10	1.084,51	1.086,79	8	14
Media resultados	1.010,82	1.002,85	41,1	37,6

Tabla 2: Coste y tiempo de ejecución para las diferentes soluciones iniciales

Conclusiones:

Después de realizar el experimento, con los resultados obtenidos podemos ver que las dos soluciones son muy similares en este caso que hemos usado la función heurística que solo tiene en cuenta el precio. Sin embargo, teniendo en cuenta que los resultados son ligeramente mejores usando la solución inicial 2 hemos decidido optar por esta para la realización de los siguientes experimentos.

Experimento 3

Determinar los parámetros que dan mejor resultado para el Simulated Annealing con el mismo escenario, usando la misma función heurística y los operadores y la estrategia de generación de la solución inicial escogidos en los experimentos anteriores.

Observación	Tiene que haber una combinación de parámetros para el algoritmo Simulated Annealing que de unos resultados más buenos en comparación a otras.
Planteamiento	Probamos diferentes combinaciones de parámetros y observamos los resultados con cada uno.
Hipótesis	Ho: Hay una combinación con la que Simulated Annealing tiene mejores resultados que Hill Climbing. H1: Hay una combinación con la que Simulated Annealing tiene los mismos resultados que Hill Climbing.

Método	- Configuramos el estado y la función heurística con los datos fijados en los experimentos anteriores (100 paquetes, proporción de 1,2, operador mover paquete, solución inicial 2 y función heurística 1).
	- Elegimos 10 semillas aleatorias, una para cada réplica.
	- Asignamos 50000 al número de iteraciones (steps) para asegurarnos de que llegamos a converger.
	- Ejecutamos el algoritmo con los siguientes valores para cada réplica:
	  - steps: 50000.
	  - stiter: 500.
	  - k: 1, 5, 25, 125.
	  - λ : 1.0, 0.01, 0.0001.
	- Tras realizar la media de los valores k y λ para cada réplica, generamos varias gráficas y elegimos los parámetros que muestran los mejores resultados.
	- Con los valores de k y λ asignados, elegimos 10 semillas y pasaremos a acotar el valor del parámetro número de iteraciones tras probar con los siguientes valores:
	  - steps 5000 - stiter 100
	  - steps 10000 - stiter 100
	  - steps 50000 - stiter 500
	- Generamos varias gráficas y elegimos el número de iteraciones que muestre los mejores resultados.

Resultados:

Primera parte: encontrar mejores valores para k y λ.

Figura 1: Coste para las diferentes parámetros k y λ para Simulated Annealing

Segunda parte: encontrar los mejores valores para steps y stiter.

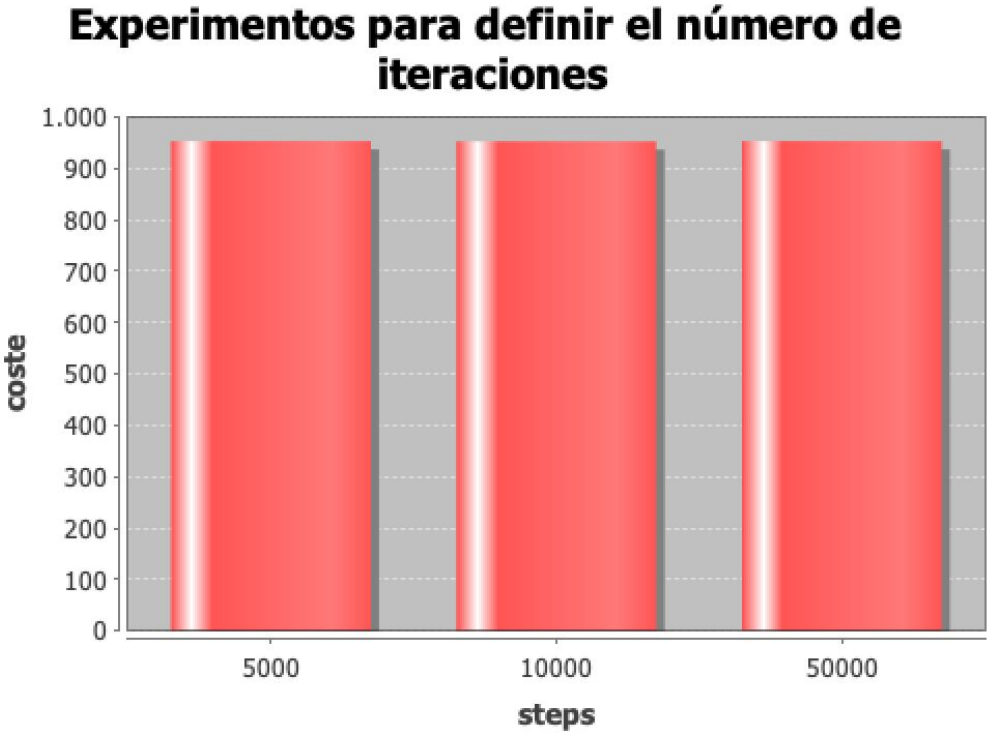

Figura 2: Coste para las diferentes parámetros steps y stiter para Simulated Annealing

Conclusiones:

Después de realizar el experimento, con los resultados obtenidos podemos ver que los costes son muy similares. Aun así, la combinación que tiene un coste más bajo es k = 25 y λ = 0,0001. Este valor para k es bastante adecuado, dado que un valor más alto supondría un aumento del número de iteraciones en las que podemos aceptar un estado peor, cosa que queremos evitar. Por otro lado, asignar λ = 0,0001 es una buena elección porque conseguimos una solución buena sin aumentar excesivamente el número de iteraciones que hacen falta para llegar a probabilidad 0, cosa que afecta directamente al tiempo de ejecución.

Tras fijar estos dos parámetros y poner a prueba el número de iteraciones, hemos visto que los costes son muy similares, pero hemos decidido asignar steps = 10000 y stiter = 100 ya que de esta manera ejecutamos un número de iteraciones razonable.

Experimento 4

Dado el escenario de los apartados anteriores, estudiad como evoluciona el tiempo de ejecución para hallar la solución en función del número paquetes y la proporción de peso transportable.
Usad el algoritmo de Hill Climbing y la misma heurística.

Experimento 4.1

- Fijad el número de paquetes en 100 e id aumentando la proporción del peso transportable desde 1, 2 de 0, 2 en 0, 2 hasta ver la tendencia.

Observación	El tiempo de ejecución puede ser más elevado para una mayor proporción de peso.
Planteamiento	Para un número de paquetes, vamos aumentando la proporción del peso transportable y observamos los resultados.
Hipótesis	Ho: El tiempo de ejecución aumenta con la proporción de peso. H1: El tiempo de ejecución se mantiene igual independientemente de la proporción de peso.
Método	- Como variable independiente tenemos la proporción de peso y como variable dependiente su respectivo tiempo de ejecución. - Elegimos 10 semillas aleatorias, una para cada réplica. - Fijamos el límite de proporción en 5 y ejecutamos el experimento para cada valor de proporción (desde 1,2 hasta 5, aumentando 0,2 en cada iteración). - Configuramos lo que queda del estado y la función heurística con los datos fijados en los experimentos anteriores (100 paquetes, operador mover paquete, solución inicial 2 y función heurística 1). - Usamos el algoritmo de Hill Climbing. - Registramos el tiempo de ejecución del algoritmo y analizamos los resultados.

Resultados:

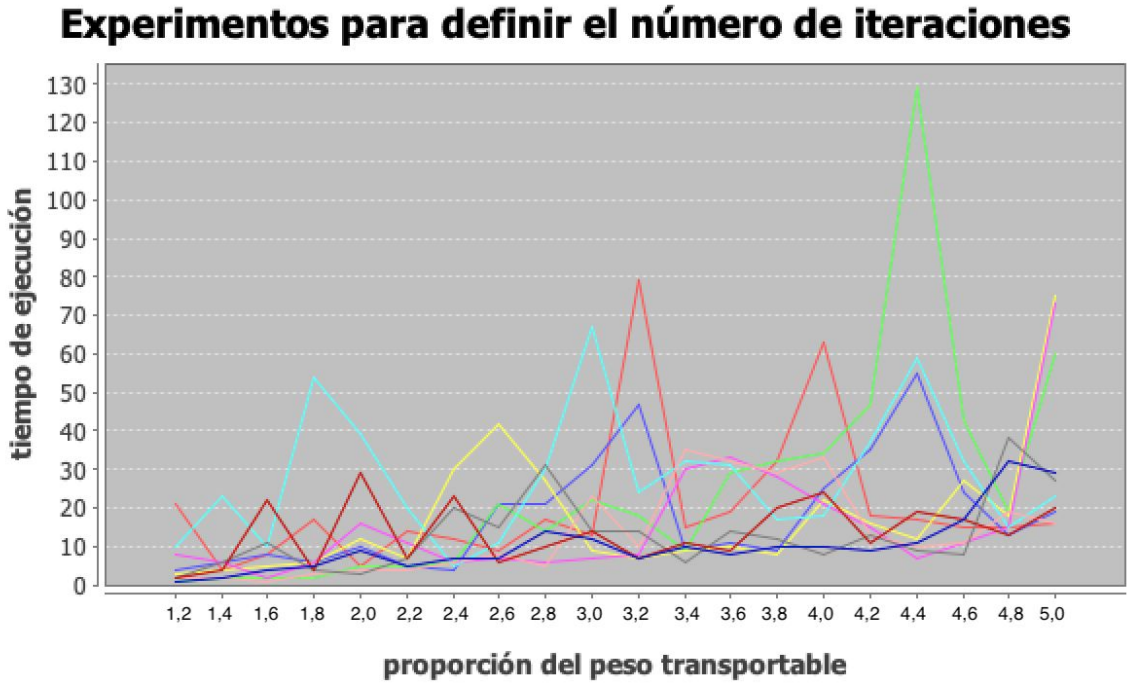

Figura 3: Tiempo de ejecución para diferentes valores de proporción de peso transportable

Conclusiones:

Aunque en la figura X.X podemos ver como hay ejecuciones en las que ciertos valores quedan muy lejos de los demás, si nos fijamos en el conjunto podemos ver cómo, a medida que aumenta la proporción del peso transportable, también lo hace el tiempo de ejecución siguiendo una distribución lineal. Esto es debido a que cuando aumentamos la proporción, la función generadora de estados sucesores puede mover cada paquete a más ofertas, en comparación con una proporción menor.

Experimento 4.2

- Fijad la proporción de peso en 1, 2 e id aumentando el número de paquetes desde 100 de 50 en 50 hasta ver la tendencia.

Observación	El tiempo de ejecución puede ser más elevado para un mayor número de paquetes a transportar.
Planteamiento	Para una única proporción, vamos aumentando el número de paquetes y observamos los resultados.
Hipótesis	Ho: El tiempo de ejecución aumenta con el número de paquetes. H1: El tiempo de ejecución se mantiene igual independientemente del número de paquetes.

Método	- Como variable independiente tenemos el número de paquetes y como variable dependiente su respectivo tiempo de ejecución. - Elegimos 10 semillas aleatorias, una para cada réplica. - Fijamos el límite del número de paquetes en 1000 y ejecutamos el experimento para cada valor de la cantidad de paquetes (desde 100 hasta 500, aumentando 50 en cada iteración). - Configuramos lo que queda del estado y la función heurística con los datos fijados en los experimentos anteriores (proporción de paquetes 1,2, operador mover paquete, solución inicial 2 y función heurística 1). - Usamos el algoritmo de Hill Climbing. - Registramos el tiempo de ejecución del algoritmo y analizamos los resultados.

Resultados:

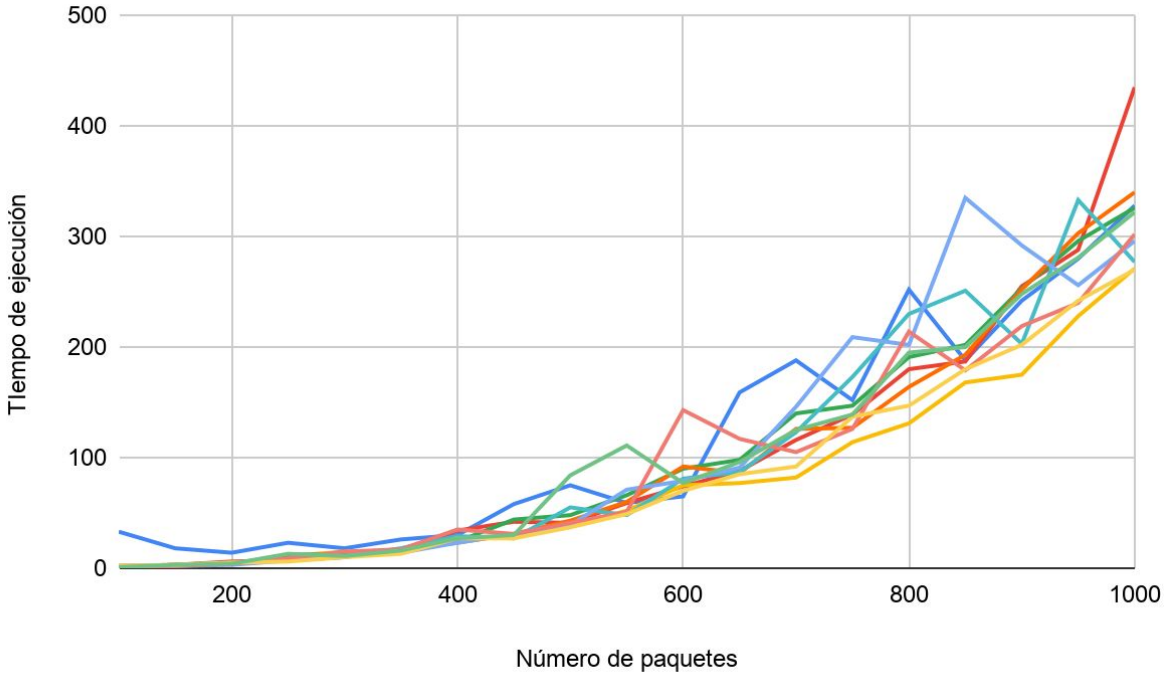

Figura 4: Tiempo de ejecución para diferentes valores de número de paquetes

Conclusiones:

En la figura X.X se ve claramente que, al aumentar el número de paquetes, el tiempo de ejecución necesario para encontrar una solución aumenta de manera exponencial. Esto se debe a que cuantos más paquetes tenemos, más posibles estados sucesores se pueden generar.

Experimento 5

Analizando los resultados del experimento en el que se aumenta la proporción de las ofertas ¿cual es el comportamiento del coste de transporte y almacenamiento? ¿merece la pena ir aumentando el número de ofertas?

Observación	El coste puede variar según la proporción de las ofertas.
Planteamiento	Vamos aumentando la proporción y observamos los resultados del coste.
Hipótesis	Ho: El coste aumenta con la proporción del peso transportable. H1: El coste se mantiene igual independientemente de la proporción de las ofertas de transporte.
Método	- Como variable independiente tenemos la proporción del peso transportable y como variable dependiente su respectivo coste. - Elegimos 10 semillas aleatorias, una para cada réplica. - Fijamos el límite de proporción en 5 y ejecutamos el experimento para cada valor de proporción (desde 1,2 hasta 5, aumentando 0,2 en cada iteración). - Configuramos lo que queda del estado y la función heurística con los datos fijados en los experimentos anteriores (número de paquetes 100, operador mover paquete, solución inicial 2 y función heurística 1). - Usamos el algoritmo de Hill Climbing. - Registramos el coste y analizamos los resultados.

Resultados:

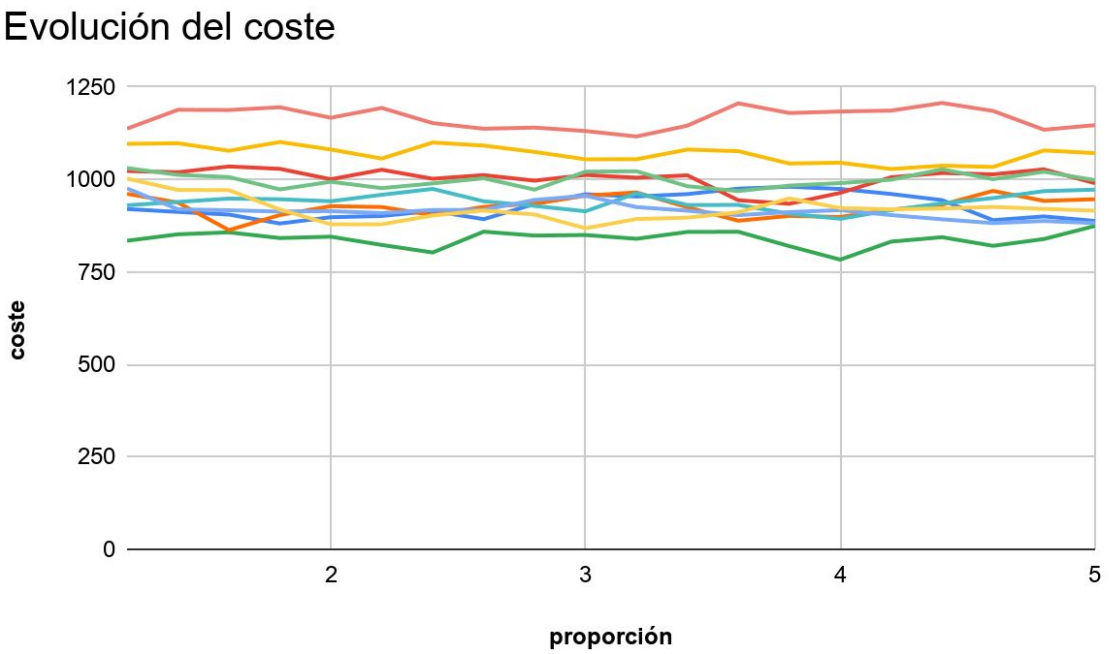

Figura 5: Coste para diferentes valores de proporción de peso transportable

Conclusiones:

Como se puede ver en la gráfica obtenida, el precio va variando cuando se varía el número de ofertas, pero no se puede apreciar que siga ninguna distribución, es decir, que mejore ni empeore el precio final en referencia a la proporción de ofertas respecto al número de paquetes. Es por esto que hemos concluido que no es necesario aumentar esta proporción ya que, como hemos apreciado anteriormente, solo aumenta inútilmente el tiempo de ejecución, sin mejorar el resultado de precio obtenido.

Experimento 6

Ahora queremos estudiar cómo afecta la felicidad de los usuarios al coste de transporte y almacenamiento. Asumiremos que esta felicidad se calcula como la suma de las diferencias entre el tiempo mínimo de llegada del paquete indicado por su prioridad y el número de días que realmente ha tardado en llegar cuando esta cantidad es positiva. Por ejemplo, si un paquete tiene prioridad de 4-5 días y llega en 3 días, cuenta como un punto de felicidad, pero si llega en 5 días cuenta como 0 puntos de felicidad.

Dado el escenario del primer apartado, estimad como varían los costes de transporte y almacenamiento y el tiempo de ejecución para hallar la solución con el Hill Climbing cambiando la ponderación que se da a este concepto en la función heurística. Deberéis pensar y justificar como introducís este elemento en la función heurística y como hacéis la exploración de su ponderación en la función. No hace falta que la exploración sea exhaustiva, solo hace falta que veáis tendencias.

Observación	Pueden haber diferentes ponderaciones de la felicidad en la función heurística que obtienen mejores soluciones en tiempo y coste.
Planteamiento	Escogemos diferentes valores para la ponderación de la función heurística y observamos sus soluciones.
Hipótesis	Ho: Todos los valores para ponderar son igual de óptimos. H1: Hay algún valor para ponderar mejor que otro.
Método	- Elegimos 10 semillas aleatorias, una para cada réplica. - Experimentamos con problemas de 100 paquetes y una proporción del peso transportable por las ofertas es 1,2. - Usamos el operador mover paquete. - Usamos la estrategia de inicialización 2. - Usamos la función heurística 2, que maximiza la felicidad. - Usamos el algoritmo de Hill Climbing. - Medimos el coste de transporte y almacenamiento y el tiempo de ejecución del algoritmo para realizar la comparación.

Resultados:

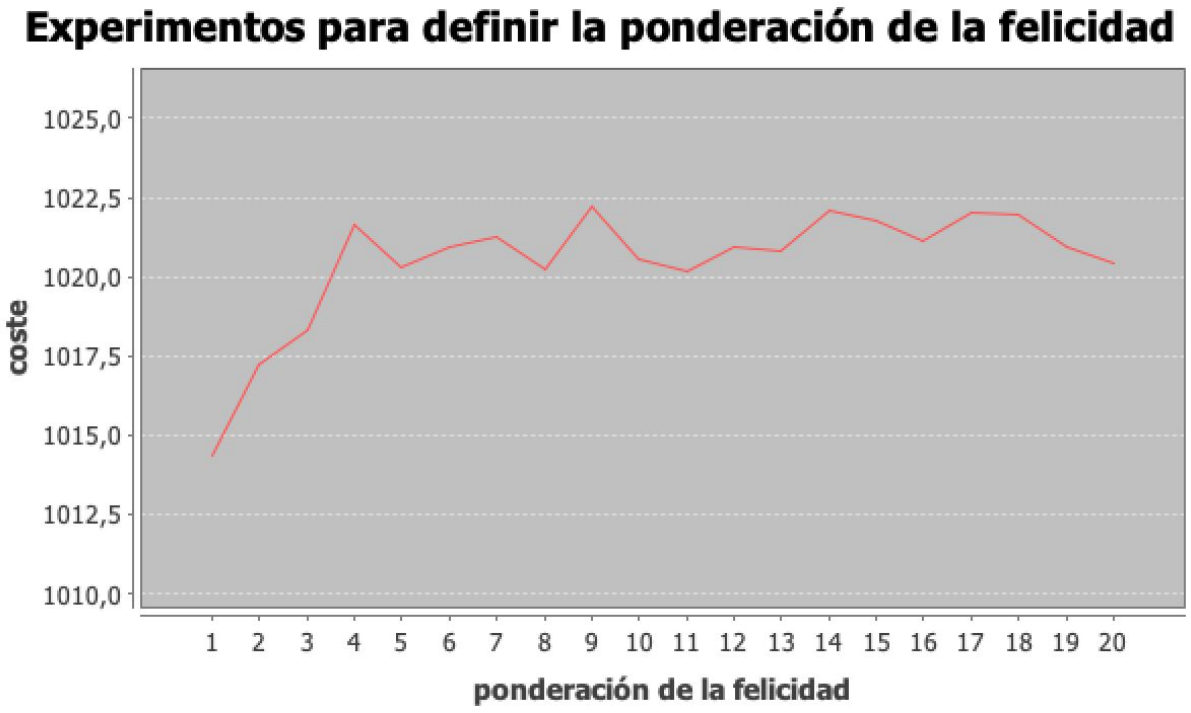

Figura 6: Coste para diferentes valores de la ponderación de la felicidad en la función heurística para Hill Climbing

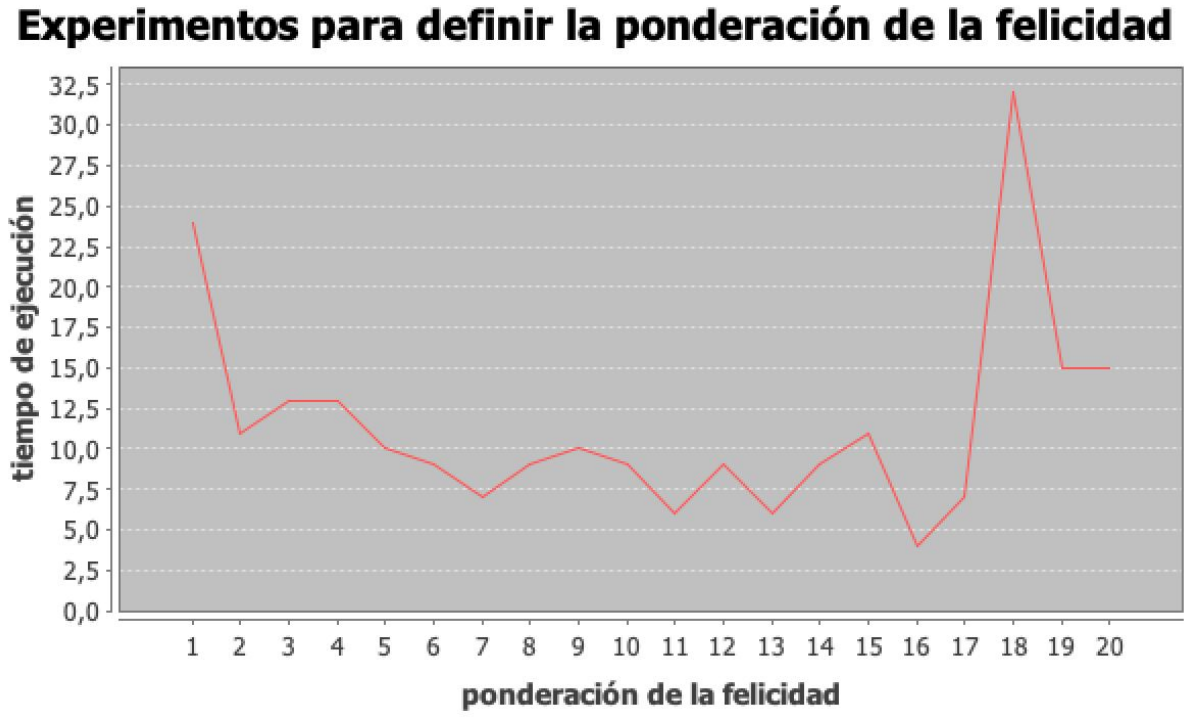

Figura 7: Tiempo de ejecución para diferentes valores de la ponderación de la felicidad en la función heurística para Hill Climbing

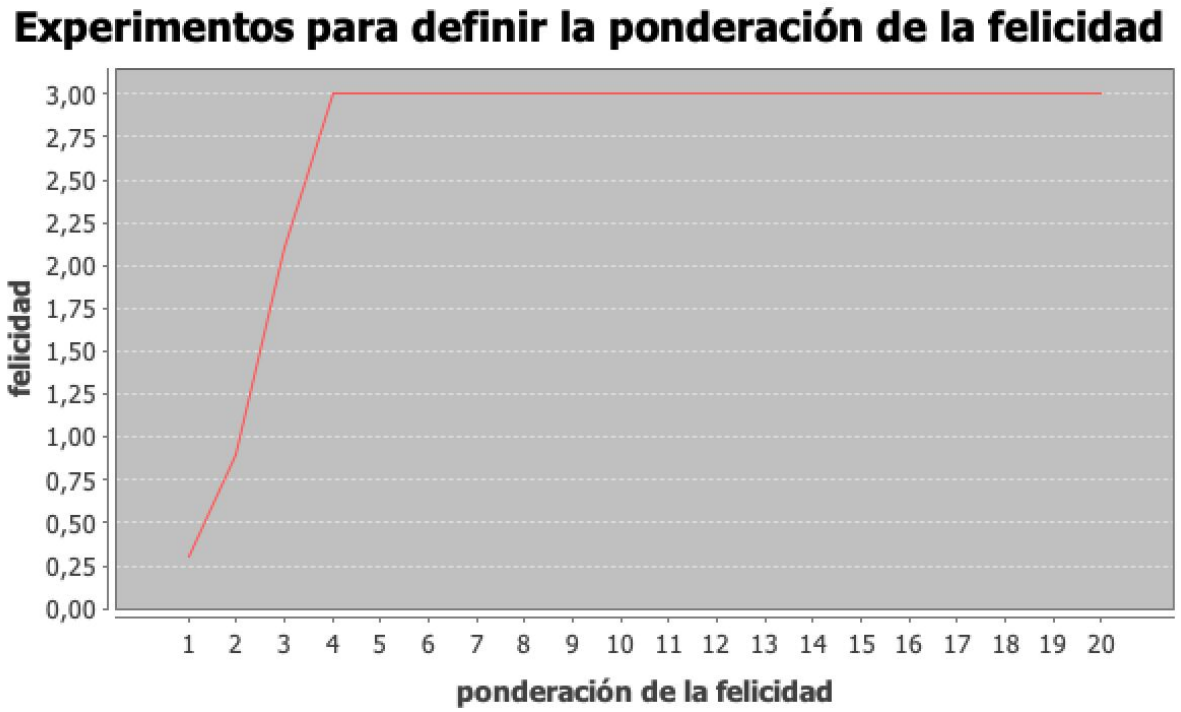

Figura 8: Valores de la felicidad para diferentes valores de la ponderación de la felicidad en la función heurística para Hill Climbing

Conclusiones:

Aunque los resultados obtenidos no siguen una tendencia clara, se puede apreciar que si no se tiene en cuenta la felicidad se obtiene un coste más reducido, y a medida que aumentamos la ponderación, el coste va aumentando y llega a un punto donde su valor se mantiene estable. Es por esto que hemos decidido elegir la ponderación que da los mejores resultados en los 3 casos, por lo que la mejor ponderación es de 11.

Experimento 7

Repetid los experimentos con Simulated Annealing y comparad los resultados. ¿Hay diferencias en las tendencias que observáis entre los dos algoritmos?

Observación	Pueden haber diferentes ponderaciones de la felicidad en la función heurística que obtienen mejores soluciones en tiempo y coste.
Planteamiento	Escogemos diferentes valores para la ponderación de la función heurística y observamos sus soluciones.
Hipótesis	Ho: Todos los valores para ponderar son igual de óptimos. H1: Hay algún valor para ponderar mejor que otro.
Método	- Elegimos 10 semillas aleatorias, una para cada réplica. - Experimentamos con problemas de 100 paquetes y una proporción del peso transportable por las ofertas es 1,2. - Usamos el operador mover paquete. - Usamos la estrategia de inicialización 2. - Usamos la función heurística 2, que maximiza la felicidad. - Usamos el algoritmo de Simulated Annealing. - Medimos el coste de transporte y almacenamiento y el tiempo de ejecución del algoritmo para realizar la comparación.

Resultados:

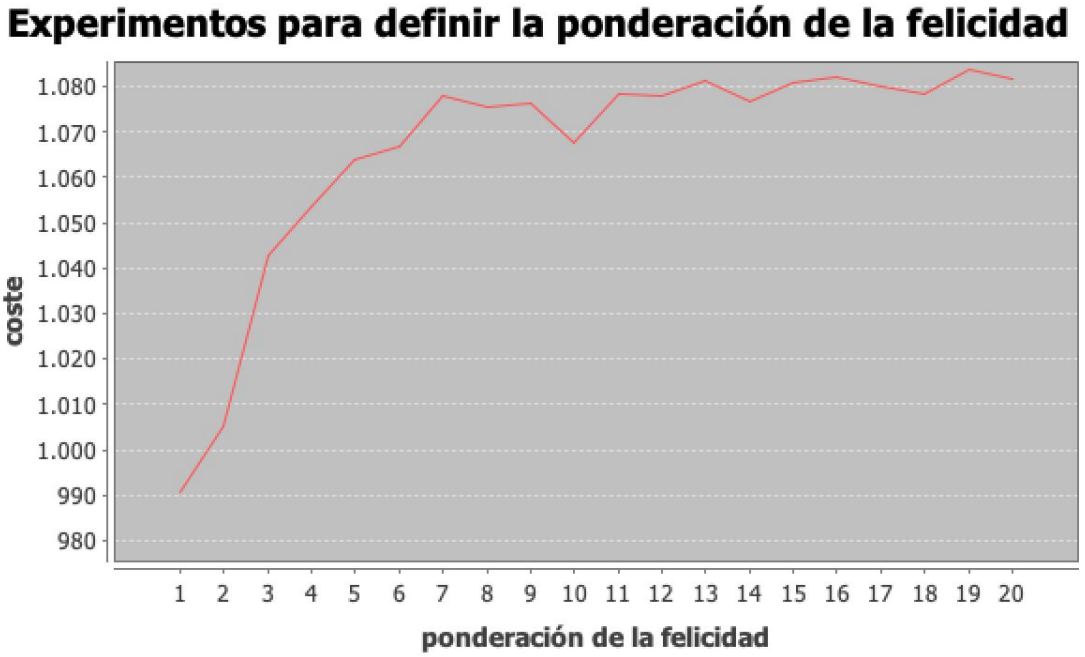

Figura 9: Coste para diferentes valores de la ponderación de la felicidad en la función heurística para Simulated Annealing

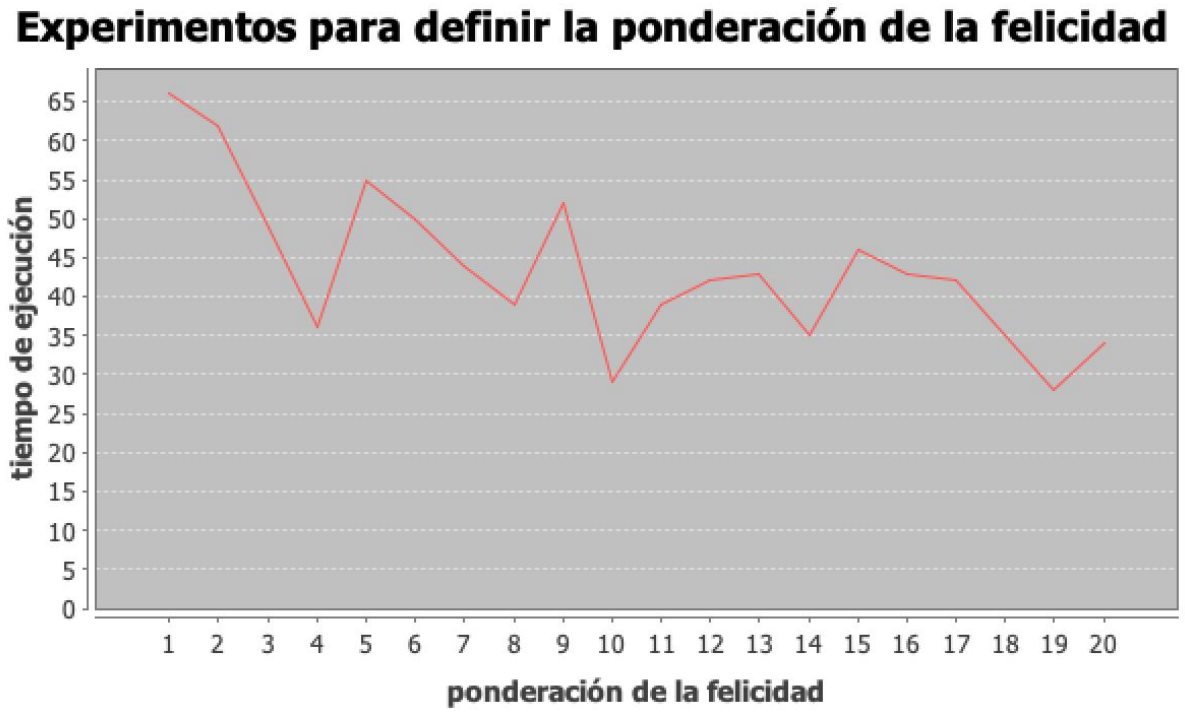

Figura 10: Tiempo de ejecución para diferentes valores de la ponderación de la felicidad en la función heurística para Simulated Annealing

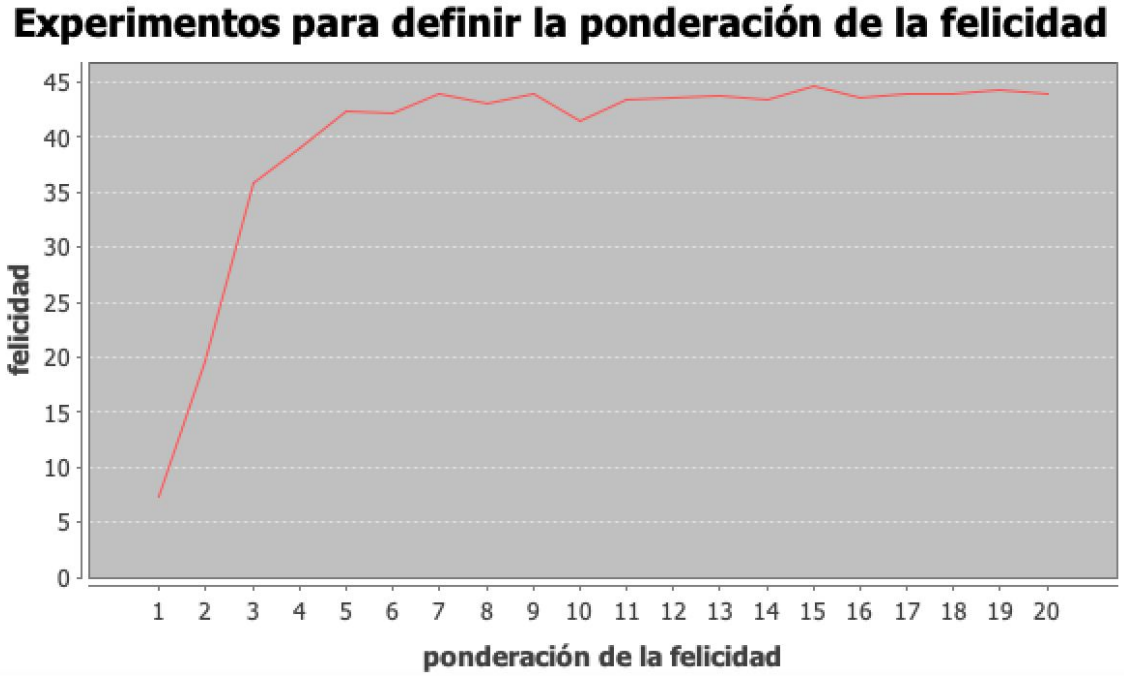

Figura 11: Valores de la felicidad para diferentes valores de la ponderación de la felicidad en la función heurística para Simulated Annealing

Conclusiones:

En este experimento, obtenemos un caso similar al anterior en referencia al coste, es decir, a partir del valor 7 el coste se estabiliza. Teniendo en cuenta todos los factores, hemos decidido elegir la ponderación que da los mejores resultados, por lo que la mejor ponderación es de 10. Aunque la felicidad obtenida sea ligeramente inferior a otros posibles resultados, el tiempo y el coste que obtenemos son muy buenos. La diferencia principal con el experimento anterior se encuentra en el valor de la felicidad, ya que en este caso, como no acabamos la ejecución del algoritmo al encontrar el primer mínimo local, llegamos a estados donde ésta se maximiza mucho más.

Experimento 8

Hemos asumido un coste de almacenamiento fijo diario de 0,25 euros por kilo. Sin hacer ningún experimento ¿cómo cambiarían las soluciones si variamos este precio al alza o a la baja?

Si variamos el precio de almacenamiento al alza, las mejores soluciones se enfocarían más en encontrar ofertas con que cumplan la prioridad y que además se entreguen lo antes posible. Gracias a esta variación también se conseguiría mejorar considerablemente el índice de felicidad de los clientes.

Por otro lado, si el precio es a la baja, no se daría importancia a este coste de almacenamiento y la solución se centraría en encontrar ofertas que cumplan la prioridad y que su precio sea el mínimo.