DOCUMENTACIÓN PRÁCTICA DE BÚSQUEDA LOCAL

INTELIGENCIA ARTIFICIAL 2024-2025 Q1

GRADO EN INGENIERÍA INFORMÁTICA - UPC FIB

DAVID MORAIS, JOAN VILA, NURIA ENSEÑAT

ÍNDICE

1. INTRODUCCIÓN

En esta práctica de búsqueda local se nos ha propuesto un reto de optimización logística en la que la compañía ficticia *Ázamon* nos pide mejorar su método de asignación de los paquetes a las ofertas de transporte que reciben diariamente para cada ciudad. Dichas ofertas varían en cuanto a la capacidad de carga, el precio por kilogramo y el tiempo de entrega. Por otro lado, los paquetes a enviar tienen un peso y una prioridad de entrega, lo que añade más restricciones y complejidad a la hora de asignarlos de manera óptima, es decir, minimizando los costes y maximizando el beneficio.

El problema que se nos presenta tiene las siguientes características:

1. **Paquetes**

Peso: Los paquetes que se deben enviar tienen un peso múltiplo de 0.5 kg y no superan los 10 kg.
Prioridad de envío: Los clientes pueden seleccionar entre tres tipos de prioridad:

- Entrega en 1 día, con un coste de 5 euros por paquete.
- Entrega en 2-3 días, con un coste de 3 euros por paquete.
- Entrega en 4-5 días, con un coste de 1,5 euros por paquete.

Restricciones:

- Los paquetes deben ser asignados a ofertas de transporte que cumplan con las restricciones de tiempo asociadas a su prioridad.
- Todos los paquetes deben ser asignados a alguna oferta de transporte.
- Los paquetes deben llegar a su destino dentro del plazo permitido por la prioridad asignada, pero nunca después.
- Los paquetes asignados a envíos de 3-4 días se recogen un día después, y los paquetes de envíos de 5 días se recogen dos días después.

2. **Ofertas de transporte**

Capacidad máxima de transporte: varía entre 5 y 50 kg en intervalos de 5 kg.
Precio por kilogramo transportado: es variable según la oferta específica.
Tiempo de entrega: puede oscilar entre 1 y 5 días.
Restricciones: la capacidad de cada oferta no puede ser excedida por el peso de los paquetes asignados.

3. **Costes adicionales**

Un aspecto clave del problema es que los paquetes no son recogidos de manera inmediata, lo que genera un coste de almacenamiento de 0,25 euros por kilogramo/día.

4. Satisfacción de los clientes

Otro factor a tener en cuenta es la satisfacción del cliente cuando los paquetes llegan antes del plazo de entrega que han seleccionado, lo que se traduce en más ventas futuras. Por este motivo consideraremos un componente llamado “felicidad” que se medirá según los días de adelanto en la entrega.

La solución del problema será una asignación de paquetes a ofertas de transporte. Al haber múltiples combinaciones posibles, requerimos de técnicas de optimización para explorar eficazmente. El problema tiene dos criterios principales para evaluar si una solución es óptima, o no: buscamos minimizar los costes de transporte y almacenamiento a la vez que maximizar la felicidad del cliente sin olvidarnos de cumplir con las restricciones que nos presenta el problema en cuanto a la capacidad y el tiempo de entrega.

El problema plantea un escenario donde soluciones factibles deben ser encontradas en un tiempo óptimo entre muchas posibles soluciones que no cumplen con los requisitos, lo que se ajusta bien a las técnicas de búsqueda local. Por estos motivos concluimos que se crea un paisaje de soluciones que puede ser recorrido por algoritmos de búsqueda local.

2. ELEMENTOS DEL PROBLEMA

2.1. IMPLEMENTACIÓN DEL ESTADO

En nuestra solución hemos comprobado que la mejor manera de definir un estado era asignando paquetes a las distintas ofertas de transporte, para eso hemos definido una clase llamada “AzamonBoard” que contiene los siguientes elementos:

- **nPackages:** Entero el cual contiene el número de paquetes que se tienen que transportar.

- **nOffers:** Entero que contiene el número de ofertas que existen en el problema. Esto es constante (una vez inicializado no puede cambiar)

- **packages:** Objeto de la clase Paquetes que contiene todos los paquetes que deben de ser asignados a las ofertas de transporte.

- **transport:** Objeto de la clase Transporte que contiene todos las ofertas de transportes que cada local a las que se debe asignar los paquetes

- **packageAssignment:** Vector de enteros donde cada posición hace referencia al paquete con esa misma posición en la lista de ofertas (“transport”).

- **weightLeft:** Vector de doubles que contiene el peso restante disponible para cada oferta de transporte.

- **totalPrice:** El coste total de la asignación de actual de paquetes a ofertas

- **happinessPrice:** Contiene la medida de felicidad en cada

- **happinessConstant:** Una constante que determina la importancia de la felicidad al calcular el heurístico.

JUSTIFICACIÓN

Hemos elegido esta implementación del estado ya que para asignar un Paquete a una Oferta tan solo basta con modificar la posición del vector *packageAssignament* el cual tiene un coste muy reducido.
El tener el coste en el vector weight nos da una ventaja al no tener que calcular todo el rato el peso restante. Además el cálculo del coste y felicidad también tiene un coste reducido sin necesidad de recalcular.

2.2. OPERADORES

Los diferentes operadores que hemos elegido son:

movePaquete: Mueve un paquete de una oferta a otra realizando las comprobaciones necesarias para poder realizar este movimiento:

- El paquete ya está asignado a la oferta que se quiere mover.
- Observamos si cabe el paquete en la oferta y si se entrega en un plazo compatible.

swapPaquetes: Intercambia dos paquetes de solución, es decir dados dos paquetes p1 y p2, p2 se cambia a la posición del paquete p1 y viceversa. También abarca todas las comprobaciones necesarias:

- Los dos paquetes no son el mismo
- Los dos paquetes no están asignados a la misma oferta
- Cada paquete se entrega en su plazo de prioridad y caben en la oferta en las que se les van a asignar.

JUSTIFICACIÓN

Utilizamos estos operadores ya que con ellos abarcamos todo el espacio de soluciones del problema. Los operadores “quitar” o “añadir” no se requieren ya que todos los paquetes están asignados a una oferta desde el primer momento.

Ambos son cuadráticos por nuestra implementación, el operador de move tiene un coste de $O(n*m)$, siendo n el número de paquetes y m el número de ofertas; por otro lado, el operador de swap tiene un coste de $O(n^2)$

2.3. ESTADO INICIAL

Hemos definido dos estados iniciales:

- *“ini = 0”*: Este método funciona de tal manera que ordena la lista los paquetes en función de su prioridad, obteniendo los paquetes de mayor prioridad al principio de la lista

- *“ini = 1”*: Este método funciona de tal manera que ordena la lista de los paquetes en función de su peso, obteniendo los paquetes de mayor peso al principio de la lista y ordena la lista de ofertas en función de ratio = Precio/Capacidad.

JUSTIFICACIÓN

Utilizamos estos estados iniciales ya que con ambos podemos observar resultados de dos casuísticas distintas.

2.4. FUNCIONES HEURÍSTICAS

Hemos definido dos heurísticas basándonos en los dos criterios para evaluar la calidad de una solución que nos establece el enunciado:

- **Amazon Heuristic Function:** Esta función únicamente se basa en calcular el coste actual generado por el precio de la asignación del paquete a la oferta, es decir, el coste del transporte y el almacenamiento, devolviendo este valor.

- **Amazon Heuristic Function Happiness** Esta función únicamente se basa en calcular el coste actual teniendo en cuenta el factor de felicidad acumulada hasta el momento multiplicado por una constante.

Para tener en cuenta el factor de la felicidad hemos decidido multiplicarla al coste para poder regular la importancia que le damos y estudiar en los experimentos correspondientes qué valor le damos para que sea una buena heurística.

Podemos ver que la felicidad está restando al resto del coste, esto se debe a que la librería AIMA minimiza el heurístico, pero como nosotros buscamos maximizar la felicidad tenemos que restarla al coste.

<u>JUSTIFICACIÓN</u>
Utilizamos estas heurísticas ya que con estos valores conseguimos darle la misma importancia a cada uno de los casos

3. EXPERIMENTACIÓN

3.1. Experimento 1

Observación: El conjunto de operadores que elegimos al generar los sucesores puede influir en la solución final que obtenga el Hill Climbing.

Planteamiento: Usaremos 3 conjuntos de operadores, el primero será solo usar el swap de paquetes, el segundo solo el move de un paquete y finalmente los dos combinados. Observaremos el coste y los nodos expandidos del estado final encontrado.

Hipótesis: Usar la combinación de los dos operadores nos permitirá llegar a una solución más óptima, a cambio de tener más tiempo de cómputo.

Método:

Generamos 10 semillas aleatorias.

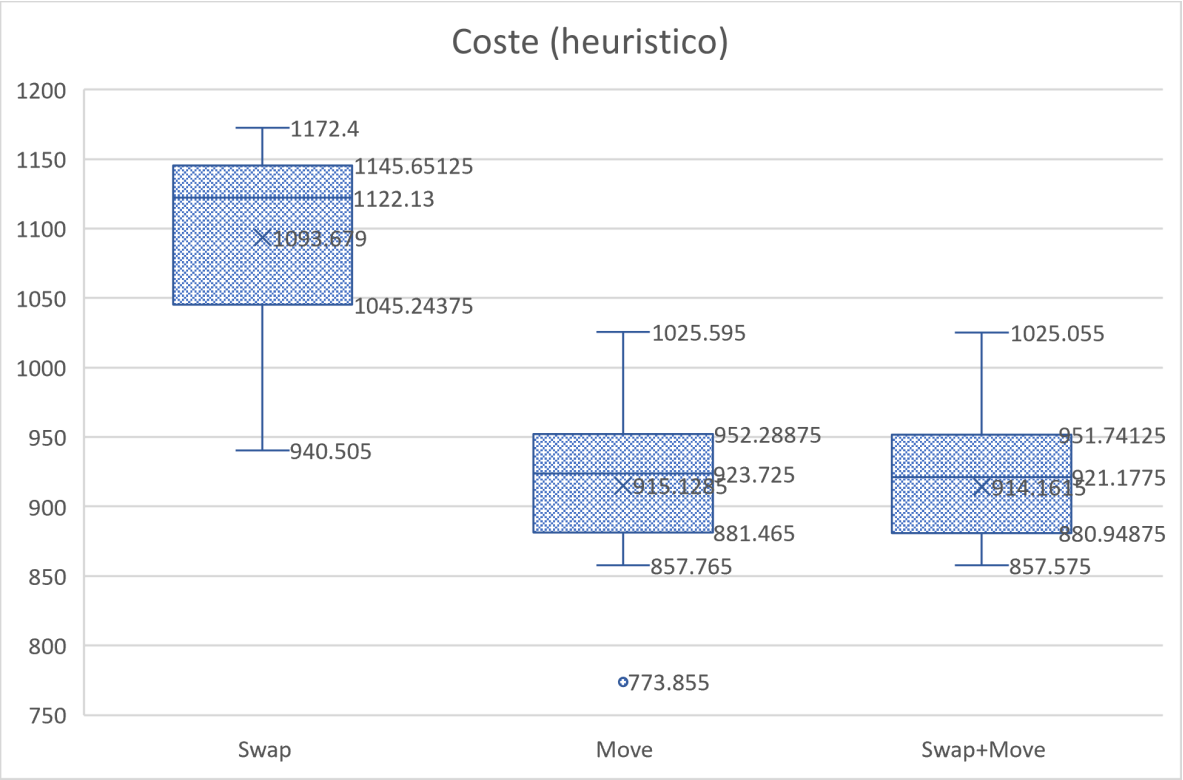
Ejecutaremos 3 experimentos para cada semilla, uno por conjunto de operadores.

Compararemos los resultados obtenidos y determinaremos el conjunto de operadores a usar en próximos experimentos.

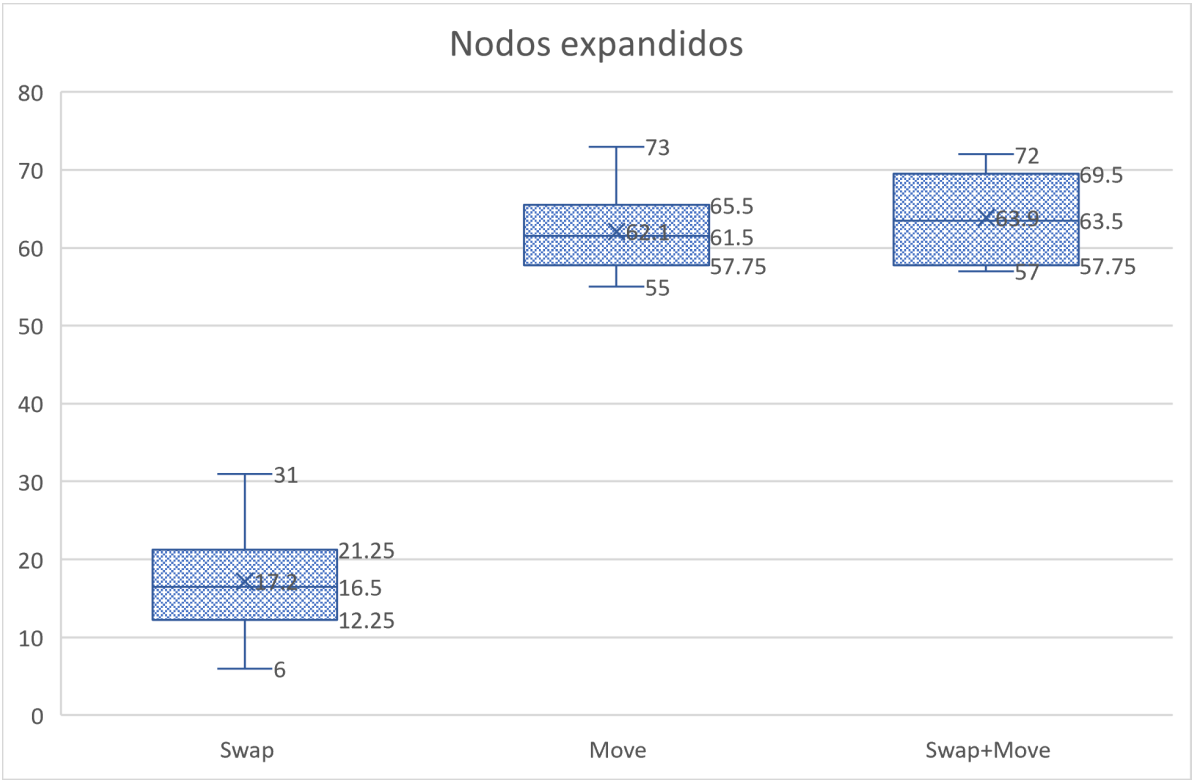
Resultados:

	Coste (heurístico)			Nodos expandidos		
Semilla	Swap	Move	Swap+Move	Swap	Move	Swap+Move
9663	1172.4	1025.595	1025.055	13	59	61
6315	1124.185	944.905	943.59	21	58	57
2899	1150.2	961.21	960.7	10	57	58
370	1144.135	943.665	939.375	17	67	71
9586	1120.075	903.785	902.98	16	65	67
1155	1047.515	901.825	901.83	31	60	61
6120	1141.43	949.315	948.755	22	64	66
920	940.505	773.855	773.015	16	63	69
3077	1057.915	889.365	888.74	20	73	72
6223	1038.43	857.765	857.575	6	55	57

Tabla con los resultados de las distintas ejecuciones con diferentes semillas



Boxplot del coste (heurístico) de todas las ejecuciones



Boxplot de los nodos expandidos de todas las ejecuciones

Con la tabla y los boxplots, podemos comprobar que, como decíamos en la hipótesis, la combinación de los dos operadores nos daría una solución más óptima. Pero lo que no esperábamos es que el operador move estuviera tan cerca de la combinación de los dos operadores. Simplemente, el move puede llegar

a una solución bastante buena y explorar casi el mismo espacio de soluciones. En cambio, el swap, nos da soluciones con un coste bastante mayor y explora muy poco el espacio de soluciones. Por estos motivos elegiremos la combinación del swap+move para los demás experimentos, ya que la combinación nos deja explorar un poco más el espacio de soluciones para obtener soluciones más óptimas (menor precio).

3.2. Experimento 2

Observación: la manera en la que generamos la solución inicial afectará directamente a la solución final.
Planteamiento: elegimos dos maneras de generar la solución y analizaremos cuál de las dos genera una mejor solución final.
Hipótesis: el método 1 genera una solución inicial un poco peor y por ende permitirá que el algoritmo de Hill Climbing pueda encontrar una solución final mejor.
Método:

1. Generamos 10 semillas aleatorias diferentes.
2. Ejecutaremos el algoritmo de Hill Climbing 10 veces con el conjunto de operadores de move y swap y las 10 semillas generadas, primero con la solución inicial 1 y luego con la solución inicial 2.
3. Extraemos el valor medio de las 10 ejecuciones para cada semilla y comparamos los resultados.

	SOLUCIÓN INICIAL 1			SOLUCIÓN INICIAL 2		
Semilla	**Coste heurístico**	**Texe (ms)**	**Nodos expandidos**	**Coste heurístico**	**Texe (ms)**	**Nodos expandidos**
1234	794,675	349	72	794,785	261	41
6120	948,75	314	66	948,76	151	19
9663	1025,06	303	61	1024,99	147	15
370	939,38	329	71	939,15	183	28
1155	901,75	421	61	901,93	262	27
2363	947,6	329	65	947,6	227	33
5421	898,02	358	76	898,12	283	41
673	882,24	345	61	882,23	196	30
20	1048,17	394	65	1048,87	186	26

En esta tabla podemos comprobar cómo afecta directamente a la solución final la manera en la que generamos la solución inicial. En nuestro caso, la solución inicial 1 ordena los paquetes por prioridad, mientras que la solución inicial 2 primero ordena los paquetes por peso y las ofertas por un ratio de precio/capacidad.

Podemos ver que ambas maneras de generar soluciones iniciales acaban en soluciones finales muy similares. En nuestra implementación hemos escogido la primera manera de generar soluciones iniciales ya que a pesar de tener un coste computacional mayor, buscamos minimizar al máximo el coste de la solución final.

3.3. Experimento 3

Observación: El algoritmo *Simulated Annealing* puede generar mejores resultados que *Hill Climbing*.
Planteamiento: Ejecutamos el algoritmo *Simulated Annealing* ajustando sus parámetros y comparando los resultados con los de *Hill Climbing*. Utilizaremos el primer método de generación de estados iniciales, el primer heurístico (que sólo tiene en cuenta el coste total) y la combinación de operadores Move y Swap ya que hemos comprobado anteriormente que obtienen mejores resultados.
Hipótesis: Existe una combinación de valores (# total de iteraciones, iteraciones por cada cambio de temperatura, k, lambda) de manera que el algoritmo *Simulated Annealing* genera mejores resultados que *Hill Climbing*.
Método:

1. Escogemos 5 semillas aleatorias: 1234, 2468, 3291, 4284 y 6294.
2. Fijamos un valor para el número total de iteraciones (1000, 10000, 50000, 100000) y las iteraciones por cada cambio de temperatura, y observamos cómo cambian los resultados al cambiar los valores de k = {1, 5, 25, 125} y λ = {1, 0.01, 0.0001}.
3. Ejecutamos el algoritmo *Simulated Annealing* 5 veces con todas las combinaciones posibles de k y λ y se hace la media del coste heurístico y el tiempo de ejecución obtenidos.

Resultados:
En la siguiente tabla podemos observar los resultados obtenidos al utilizar el algoritmo de búsqueda local Hill Climbing con diferentes semillas: 1234, 2468, 3291, 4284 y 6294. Esperamos que con el algoritmo Simulated Annealing podremos encontrar soluciones similares, o incluso mejores, al utilizar ciertos parámetros que deberemos ajustar en este experimento.

Ejecución Hill Climbing	**1234**	**2468**	**3291**	**4284**	**6294**
Coste heurístico	794,675	972,26	916,49	912,82	933,24
Tiempo ejecución (ms)	349	322	313	320	420
Nodos expandidos	72	67	72	59	72

Figura 3.3.1. Tabla con los resultados de Hill Climbing

Sabemos por composición del algoritmo que la bondad de las soluciones vendrá determinada por el número total de iteraciones, ya que si es muy pequeño el algoritmo no “tendrá suficiente tiempo” para encontrar una mejor solución; el parámetro *k*, que es parte de la función de aceptación y por ende, cuanto mayor es *k*, más tarda en bajar la temperatura; el parámetro *stiter*, que son el número de iteraciones que hace el algoritmo antes de cambiar la temperatura y por último, el parámetro *λ*, que al ser parte de la función de aceptación, determinará junto al parámetro *k* qué tan rápido se baja la temperatura.

En las siguientes gráficas podremos observar los resultados obtenidos al fijar los parámetros *k* y *λ* e ir probando con diferentes números de iteraciones.

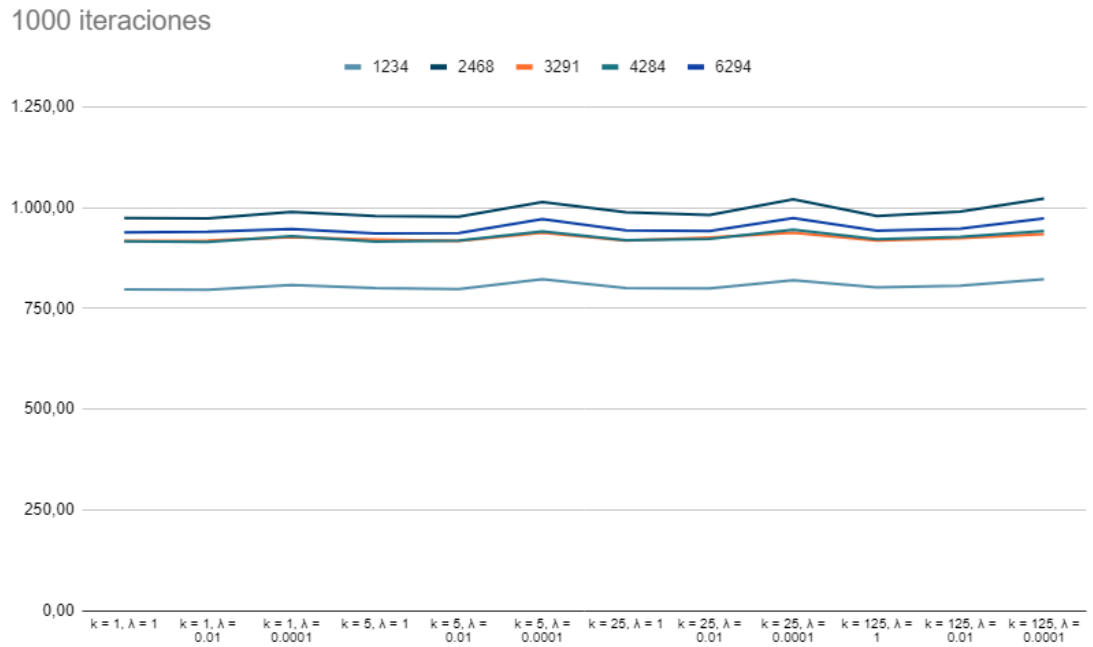


Figura 3.3.2. Coste heurístico fijando fijando 1000 iteraciones.

1000 iteraciones	Nodes expanded	Texe (ms)
λ = 1	801	1985
λ = 0.01	1001	2455
λ = 0.0001	1001	3152

Figura 3.3.3. Tabla comparativa de los nodos expandidos y el tiempo de ejecución.

Cuando fijamos las iteraciones a 1000, podemos ver cómo empeoran los resultados a medida que el valor de λ disminuye, encontrando así máximos en los puntos donde $\lambda = 0.001$. A su vez, la bondad de la solución no es tan buena como con Hill Climbing; sin embargo, este es un resultado que esperamos ya que como hemos comentado anteriormente, al fijar un número de iteraciones pequeño, el algoritmo no tendrá suficiente tiempo para explorar mejores soluciones.

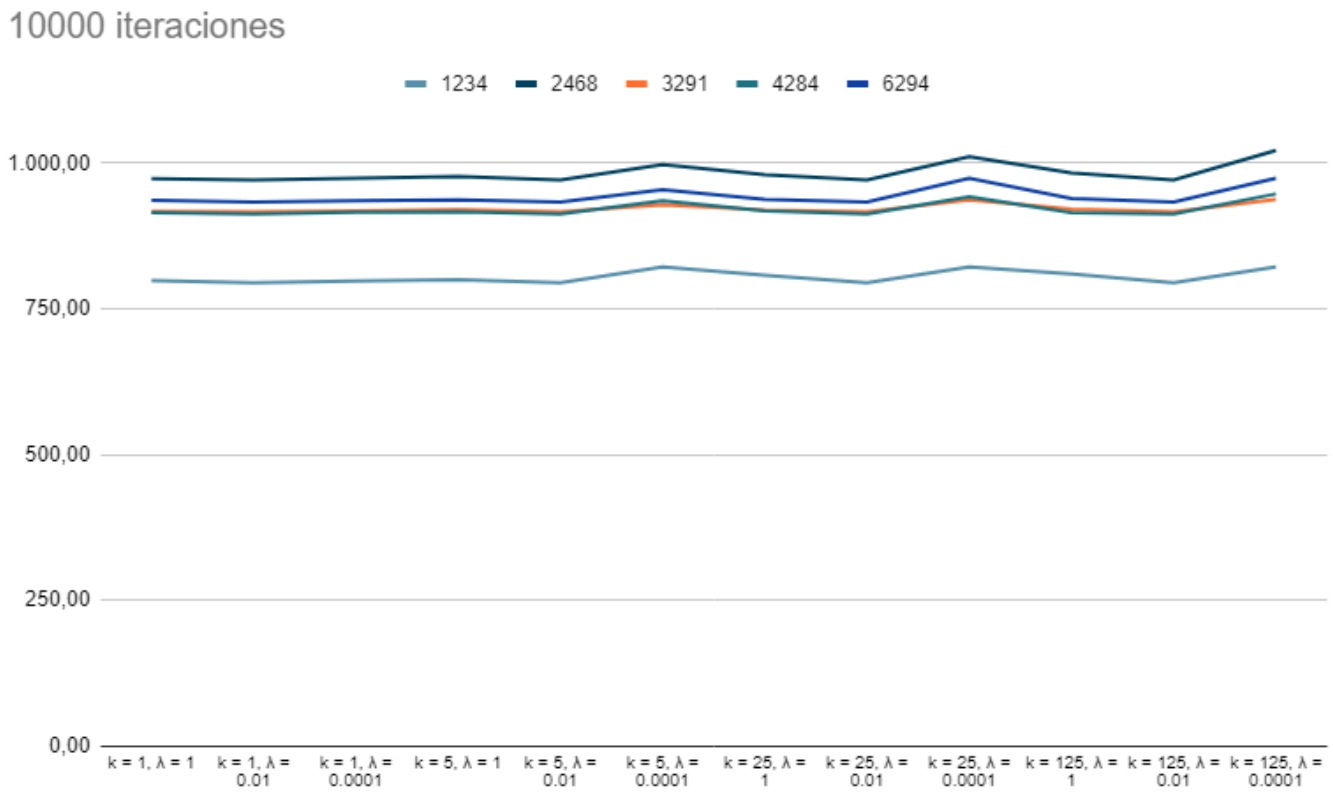


Figura 3.3.4. Coste heurístico fijando fijando 10000 iteraciones.

10000 iteraciones	Nodes expanded	Texe (ms)
λ = 1	801	1720
λ = 0.01	10001	12332
λ = 0.0001	10001	19054

Figura 3.3.5. Tabla comparativa de los nodos expandidos y el tiempo de ejecución.

Al aumentar las iteraciones a 10000 podemos comprobar que hay un cierto patrón en las soluciones obtenidas: cuando $k > 1$ y $\lambda = 0.01$ podemos ver que la gráfica presenta un mínimo local, e incluso a veces el coste obtenido es igual o más óptimo que el resultado que obtenemos con Hill Climbing. Sin embargo cuando $k = 1$, independientemente del valor de λ, los costes obtenidos son bastante similares. Y cuando $k > 5$ y $\lambda = 0.0001$ podemos ver como la calidad de la solución empeora, y por ende aparecen máximos en la gráfica, entendemos que al λ tener un valor tan pequeño, el algoritmo no puede explorar todo el espacio de soluciones y al k tomar un valor más alto, la función de sucesores acepta estados peores más fácilmente.

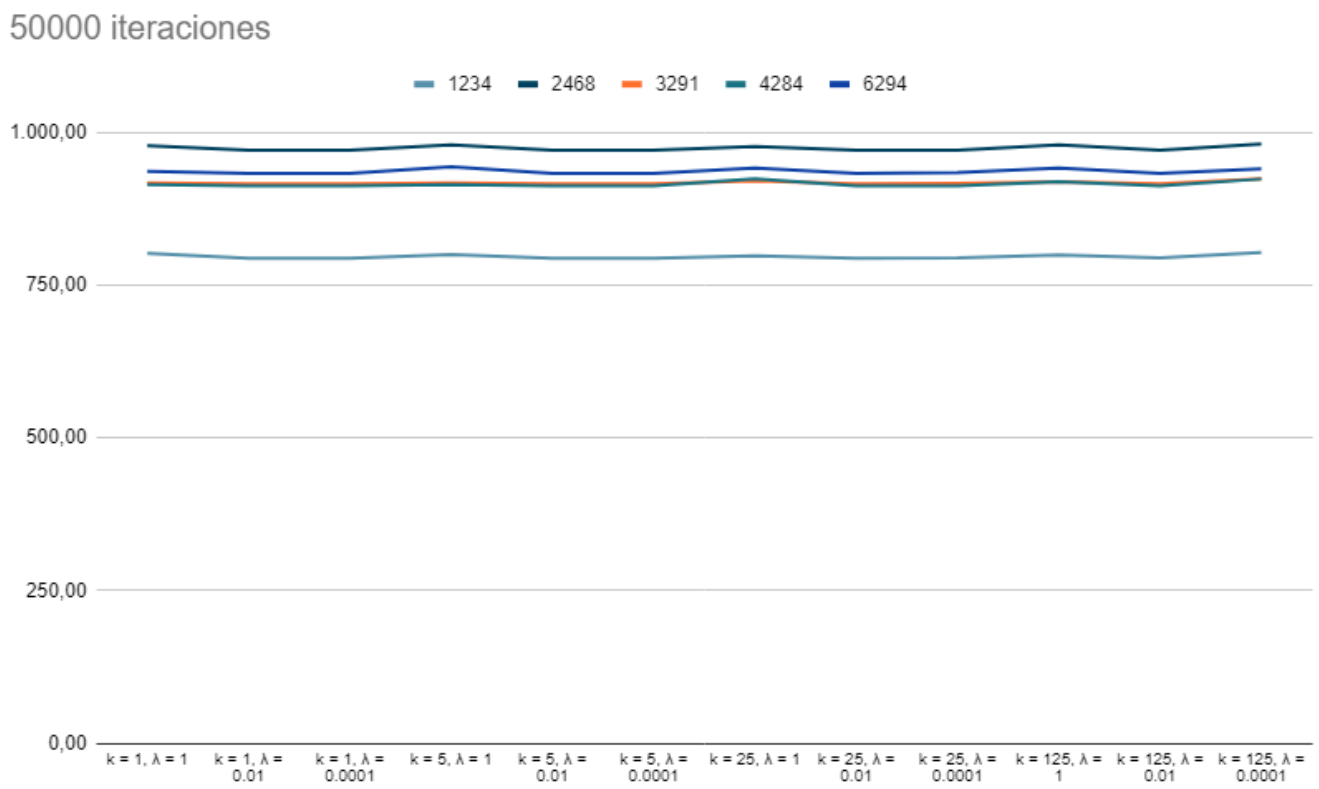


Figura 3.3.6. Coste heurístico fijando 50000 iteraciones.

50000 iteraciones	Nodes expanded	Texe (ms)
λ = 1	801	1708
λ = 0.01	50001	62706
λ = 0.0001	50001	136906

Figura 3.3.7. Tabla comparativa de los nodos expandidos y el tiempo de ejecución.

Similarmente al caso anterior, cuando fijamos el número de iteraciones a 50000 podemos volver a observar que los valores obtenidos siguen cierto patrón: cuando $k \geq 5$ y $\lambda \geq 0.01$ obtenemos soluciones iguales, o incluso un poco más óptimas, que con Hill Climbing. En general hay bastante estabilidad en las soluciones, encontrando soluciones ligeramente peores cuando $\lambda = 1$.

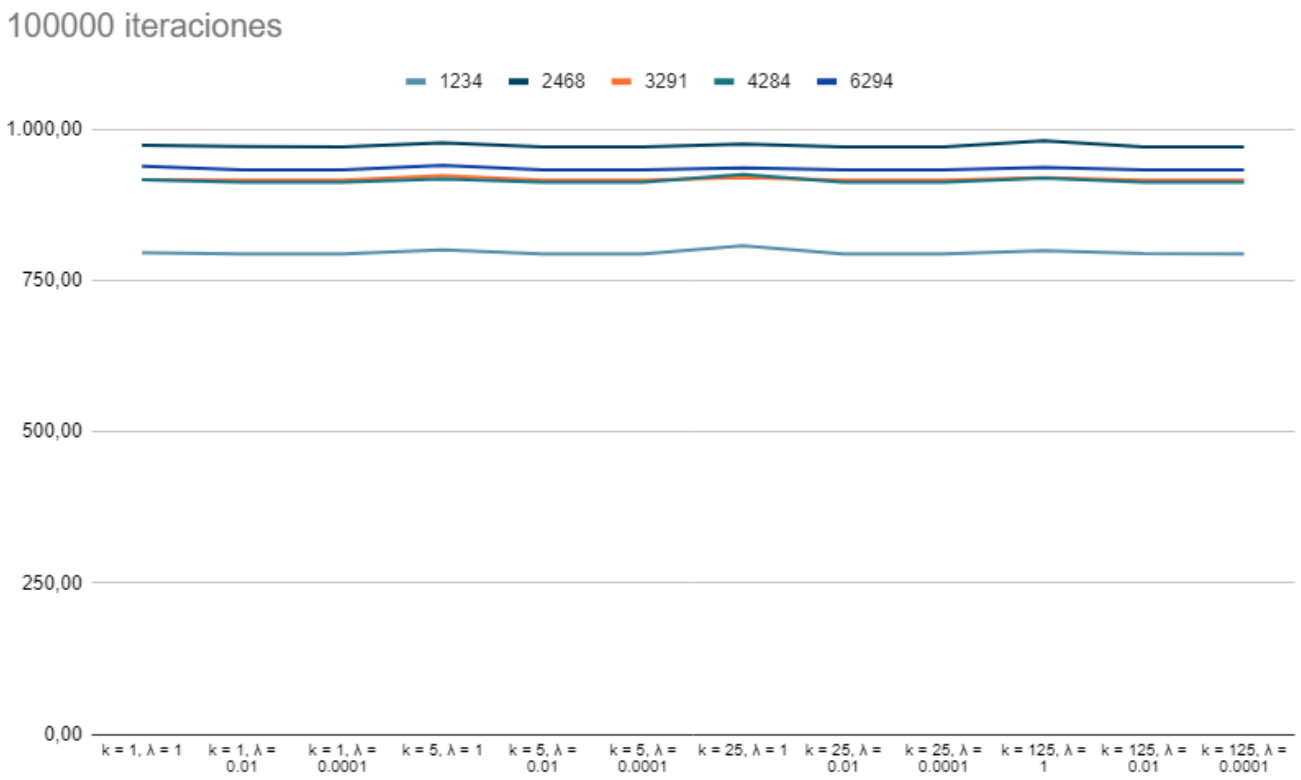


Figura 3.3.8. Coste heurístico fijando fijando 100000 iteraciones.

100000 iteraciones	Nodes expanded	Texe (ms)
λ = 1	801	2155
λ = 0.01	74601	156596
λ = 0.0001	100001	283528

Figura 3.3.9. Tabla comparativa de los nodos expandidos y el tiempo de ejecución.

Al aumentar el número de iteraciones a 100000 podemos comprobar que, cuando $k \geq 5$ y $\lambda \geq 0.01$ obtenemos soluciones similares que con Hill Climbing. Más exactamente, cuando $k = 125$ y $\lambda = 0.0001$, las soluciones obtenidas son mejores que las de Hill Climbing. Al igual que en la situación anterior, donde las iteraciones estaban fijadas a la mitad, vemos que cuando $\lambda = 0.0001$, independientemente del valor de k, es cuando encontramos los mínimos.

En resumen, podemos observar que a partir de 10000 iteraciones, la bondad de las soluciones es prácticamente igual siempre y cuando $\lambda = 0.01$ e independientemente del valor de k. Por este motivo, concluimos que los mejores parámetros para el algoritmo de *Simulated Annealing* serán 10000 iteraciones, $k = 1$ y $\lambda = 0.01$ ya que de esta manera el algoritmo habrá alcanzado una estabilidad en las soluciones aceptadas y no aceptará resultados peores a los de Hill Climbing, logrando así un equilibrio entre la exploración de las soluciones y la eficiencia computacional.

3.4. Experimento 4

Observación: Al experimentar con diferentes números de paquetes y la proporción del peso transportable, el tiempo empleado para calcular el resultado será diferente y queremos averiguar cómo varía el tiempo respecto a estos valores.

Planteamiento:

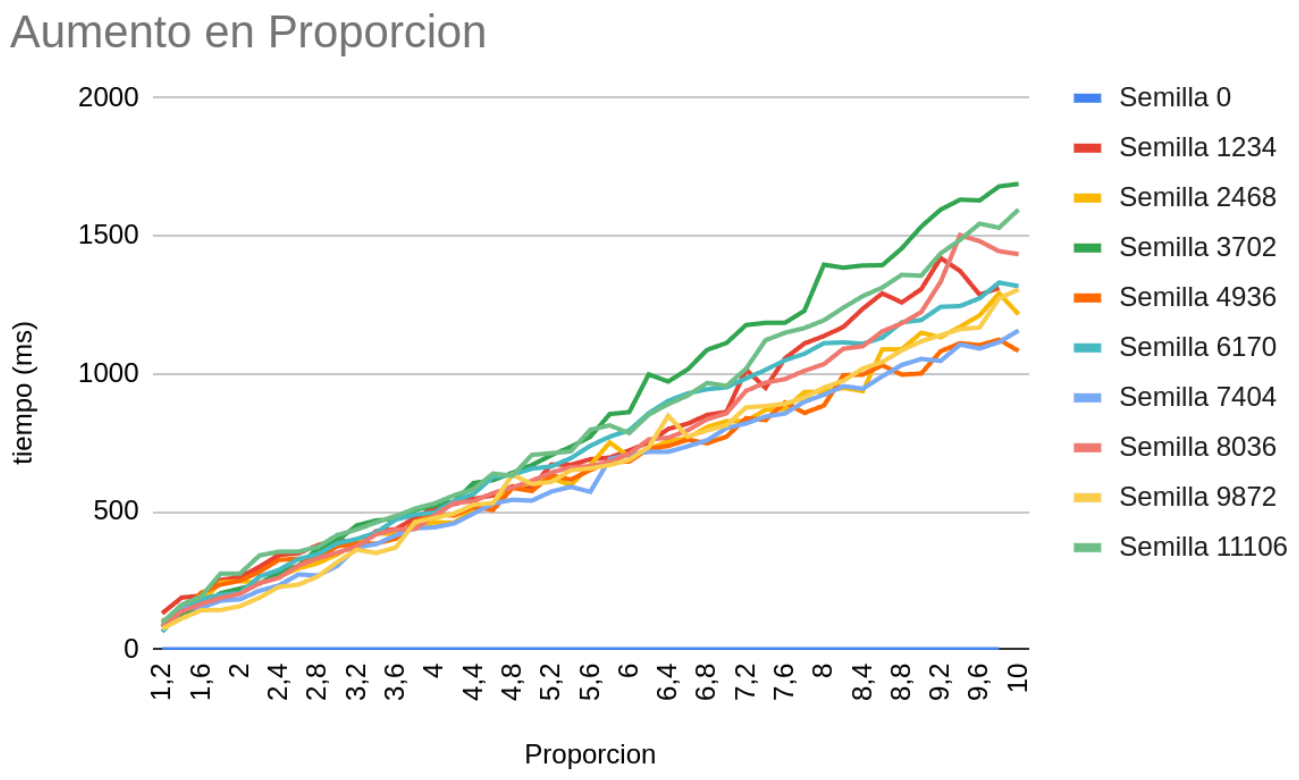
1. Fijamos número de paquetes y realizamos pruebas con las proporciones del peso transportable desde 1,2 en 0,2 hasta ver la tendencia (10)
2. Fijamos la proporción del peso transportable y realizamos pruebas con los números de paquete desde 100 de 50 en 50 hasta ver la tendencia (500).

Hipótesis: Aumentar estos valores hace que el tiempo de ejecución del programa aumente proporcionalmente, o existe un valor clave donde el tiempo cambiará radicalmente (exponencialmente)

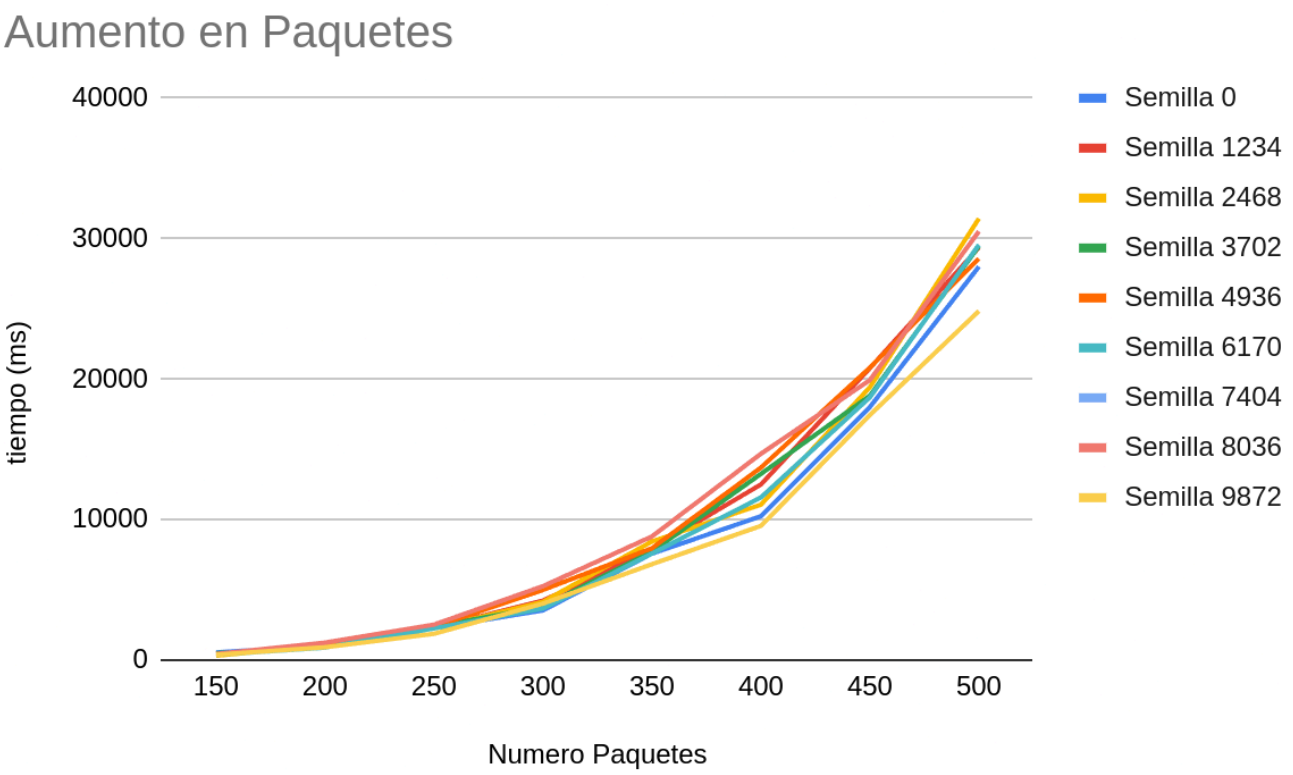
Método:

1. 10 semillas de 1234 en 1234
2. Realizamos cada uno de los dos experimentos por cada semilla con la misma heurística (precio)
3. Experimentos de 100 paquetes y subiendo proporcionalmente de 0,2 en 0,2 de 1,2 hasta 10 (hasta ver tendencia
4. Experimentos de 1,2 proporción y subir de 50 en paquetes de 100 hasta 500 hasta ver tendencia
5. Usamos Hill Climbing como se menciona en el enunciado

Resultados:



tiempo de ejecución en milisegundos en función de la proporción



tiempo de ejecución en milisegundos en función de los paquetes

Conclusiones:

Del primero vemos una tendencia donde el coste es de forma lineal debido a que la proporción solo afecta al número de ofertas y este solo implica a aumentar el uso de operadores. Por ello dada la gráfica representada anteriormente O(n*m) donde n paquetes y m número de ofertas cada vez usa un operador más por lo que aumenta en 1.

Del segundo vemos una tendencia donde el coste es de forma cuadrática debido a que el número de paquetes afecta también al número de ofertas. Por ello dada la gráfica representada anteriormente O(n*m) donde n paquetes y m número de ofertas un paquete puede implicar una oferta más así que n*m tendería a n^2.

3.5. Experimento 5

Observación: Nos preguntamos en unos de los experimentos anteriores (primero) de qué manera variar el coste de transporte

Planteamiento: 1. Fijamos número de paquetes y realizamos pruebas con las proporciones del peso transportable desde 1,2 en 0,2 hasta ver la tendencia (10)

Hipótesis: aumentar la proporción hace que varíe el coste o no

Método: 10 semillas de 1234 en 1234

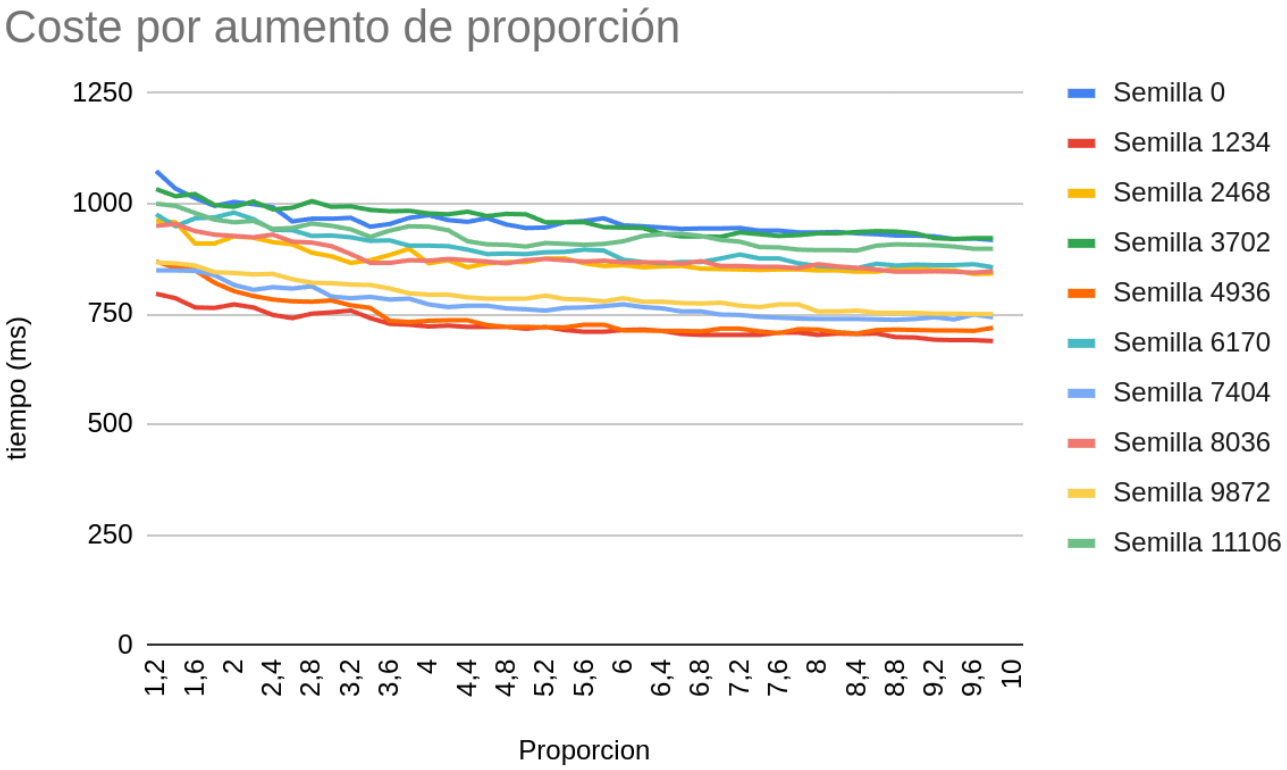
1- Realizamos cada uno de los dos experimentos por cada semilla con la misma heurística (precio)

2- Experimentos de 100 paquetes y subiendo proporcionalmente de 0,2 en 0,2 de 1,2 hasta 10 (hasta ver tendencia

3- Experimentos de 1,2 proporción y subir de 50 en paquetes de 100 hasta 500 hasta ver tendencia

4- Usamos Hill Climbing como se menciona en el enunciado

Resultados:



Coste total en función de la proporción de las ofertas

Conclusiones:

Como se observa en el gráfico la variación es muy poca y solo sucede al principio, llegando a un punto veremos que se mantendría constante por que no merece la pena aumentar el número de ofertas para disminuir el coste.

Esto ocurre ocurre con una proporción acotada, los paquetes deben distribuirse en un número límite de ofertas, pero si aumentamos la proporción llegará un punto que las pocas ofertas más baratas, dejando solo unas ofertas vacías se podrá contener todo el rango de paquetes del problema.

3.6. Experimento 6

Observación: Variando la importancia que tiene la felicidad en el cálculo del heurístico, obtendremos distintas soluciones y tiempos de ejecución para Hill Climbing.

Planteamiento: Modificaremos el valor de la constante que multiplica la felicidad de los clientes y observaremos cómo afecta en el tiempo de ejecución y la solución del Hill Climbing

Hipótesis: Habrá cierto punto donde dará igual si seguimos aumentando el valor de la constante, ya que el precio tendrá tan poca importancia en comparación a la felicidad que las soluciones serán muy parecidas o iguales.

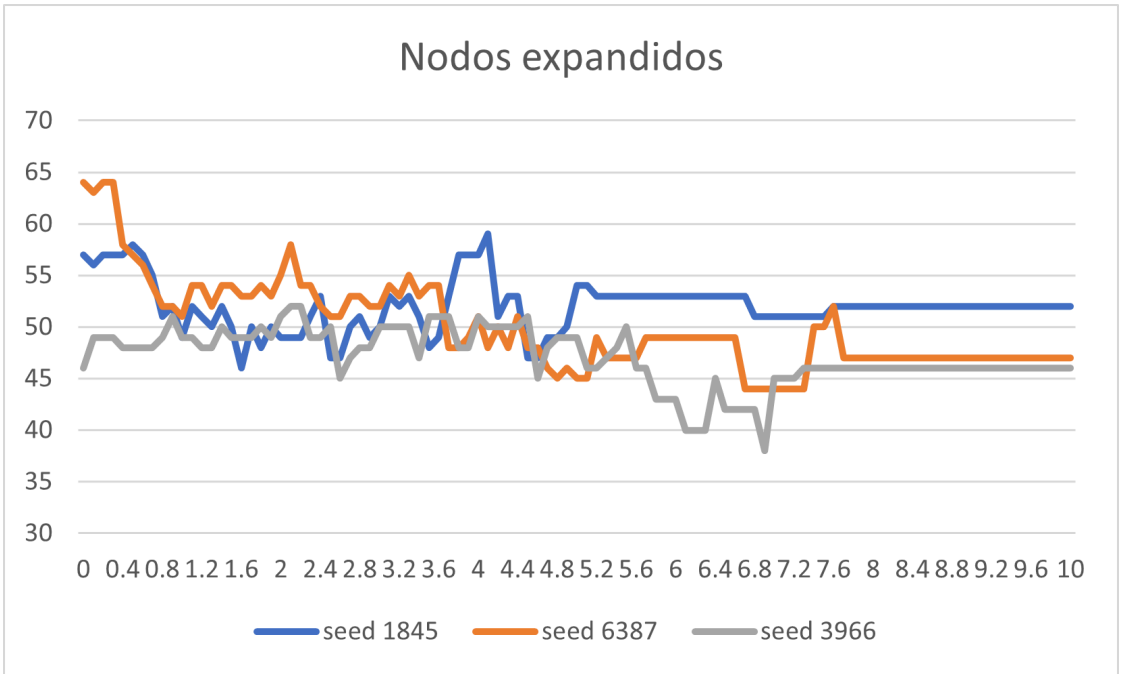
Método:

Generamos 5 semillas aleatorias

-100 experimentos se ejecutarán por cada semilla en el rango 0-10 para su respectivo promedio (tiempo y coste)

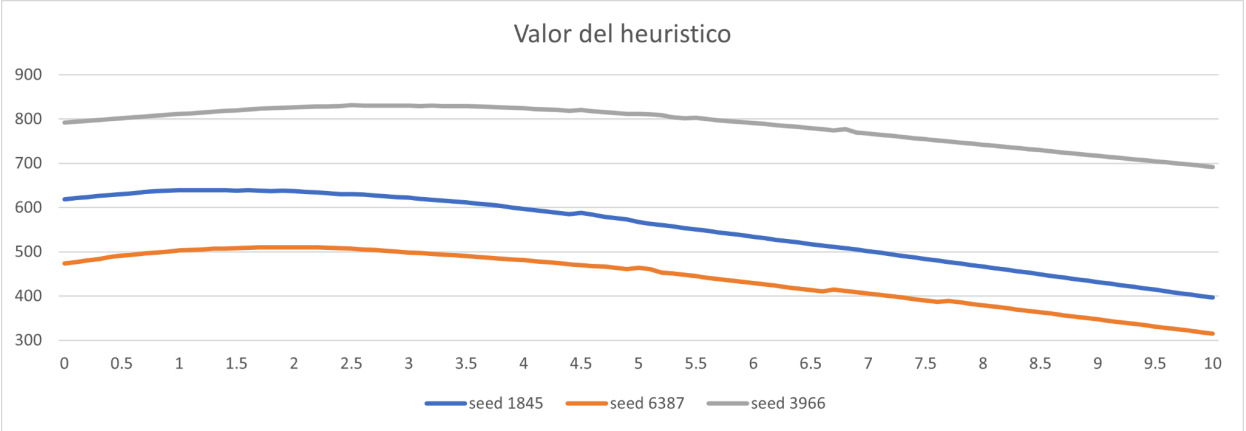
-Observaremos y analizaremos los respectivos resultados

Para introducir la ponderación al cálculo del heurístico hemos pensado en que podríamos tener una constante (happinessConstant) que puede ser la que multiplicando por el valor de la felicidad. Como la felicidad es un valor pequeño, así podemos incrementar su peso en el cálculo del heurístico.



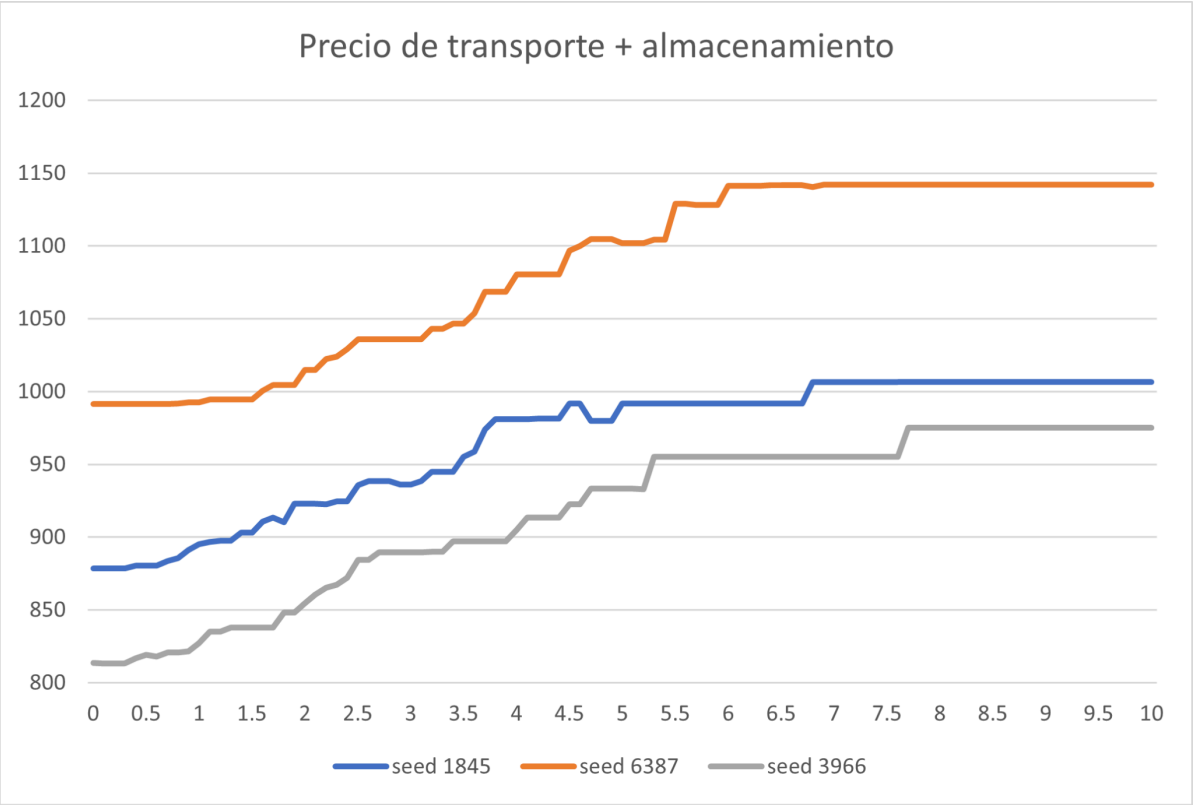
Nodos expandidos en función de la constante de felicidad (happinessConstant)

Como se puede observar en la gráfica, más o menos la cantidad de nodos que exploramos se mantiene entre los 40-60 nodos, menos en el caso cuando la constante tiene valor <1 que en ese caso es casi como correr hill climbing sin la felicidad.



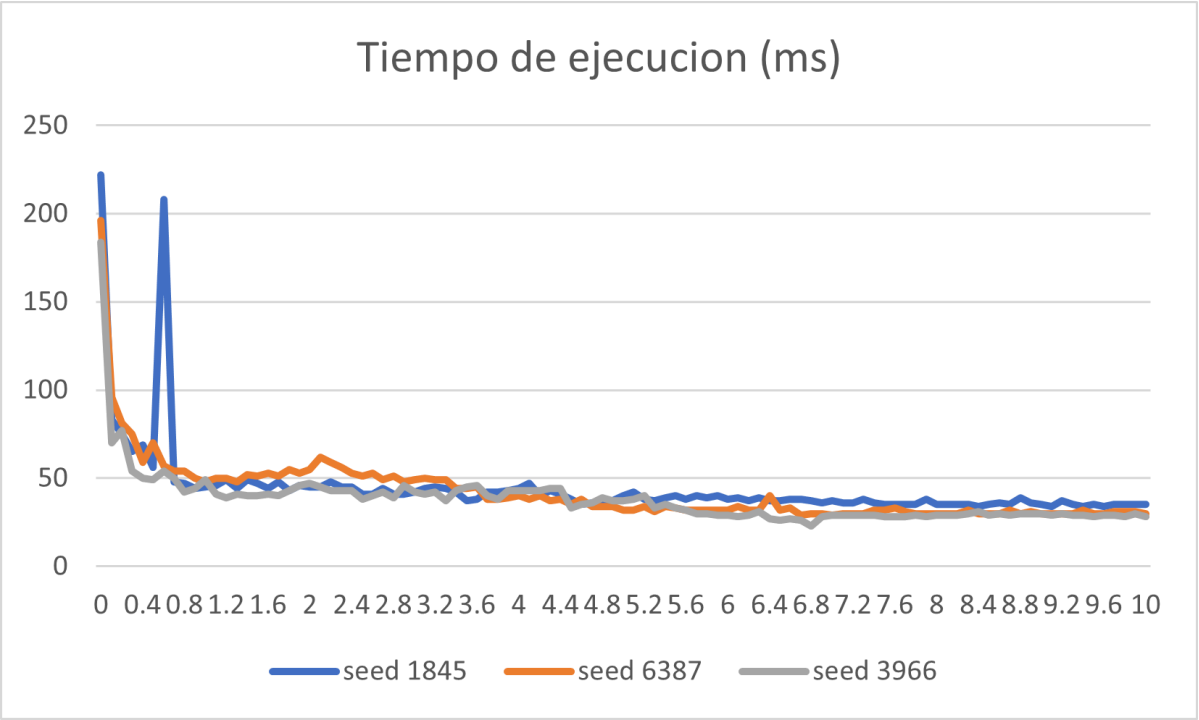
Precio total de la solución en función de la constante de felicidad

En el valor del heurístico podemos ver como hay un pico sobre el 1,5, pero luego este se va reduciendo. Esto es causado a que calculamos el heurístico, restando la felicidad al precio de transporte + almacenamiento, entonces puede que aumente el heurístico para primero cambiar los paquetes de ofertas que lleguen antes para aumentar la felicidad, entonces como lo le damos mucha importancia resta un valor pequeño, pero cuando lo vamos aumentando este valor pasa a ser mayor entonces es cuando se empieza a disminuir. En cuanto a las diferentes semillas, todas tienen un comportamiento similar.



Precio de transporte + almacenamiento en función de la constante de felicidad

Para poder tener más información del heurístico y de cómo afecta a la solución, hemos hecho también una gráfica con el precio total (transporte + almacenamiento) para poder ver si realmente la solución cambia a cada valor de la constante o llega a un punto que la solución permanece la misma. Como podemos confirmar llega cierto valor de la constante de felicidad (alrededor del 7-8) donde las soluciones son las mismas, eso quiere decir que ya habíamos llegado al máximo de felicidad posible (los paquetes llegan lo antes posible).



Tiempo de ejecución en función de la constante de felicidad

Finalmente, para el tiempo, al analizar la gráfica vemos que es bastante constante, sin embargo, hay unos picos sobre los primeros valores de la constante. Esto puede ser causado por cómo se han obtenido los valores (por terminal IntelliJ).

En conclusión podemos validar nuestra hipótesis del experimento, como hemos podido comprobar, llega un punto donde la constante de felicidad simplemente reduce el valor del heurístico porque los paquetes no pueden llegar antes (ya están todos asignados a las ofertas con menor tiempo de entrega). Para el tiempo de ejecución y los nodos podemos decir que son bastante constantes, por lo que la constante no afecta directamente.

3.7. Experimento 7

Observación: Variando la importancia de la felicidad en el cálculo heurístico, obtendremos distintos resultados y tiempos de ejecución para el algoritmo de *Simulated Annealing* que mejorarán los resultados obtenidos con *Hill Climbing*.

Planteamiento: Ejecutamos el algoritmo *Simulated Annealing* ajustando el valor de la constante "felicidad" de los clientes y observaremos cómo afecta a la bondad de las soluciones y el tiempo de ejecución obtenidos con el algoritmo de *Simulated Annealing* comparándolo con *Hill Climbing*.

Hipótesis: El algoritmo de *Simulated Annealing* se comportará igual que *Hill Climbing* al añadir al coste heurístico la importancia de la felicidad e incluso obtendrá mejores resultados.

Método:

1. Escogemos 5 semillas aleatorias: 1434, 246, 1691, 4884 y 8326.
2. Escogeremos 100 paquetes con una proporción de 1.2
3. Ejecutaremos 5 veces el algoritmo de *Simulated Annealing* con los siguientes parámetros: 10000 iteraciones, $k = 1$ y $\lambda = 0.01$ con cada semilla aumentando el valor de la felicidad progresivamente de 0 a 10.
4. Compararemos los valores obtenidos con los de *Hill Climbing*.

Primero ejecutaremos el algoritmo de Hill Climbing con las semillas escogidas para después poder comparar los resultados de ambos algoritmos:

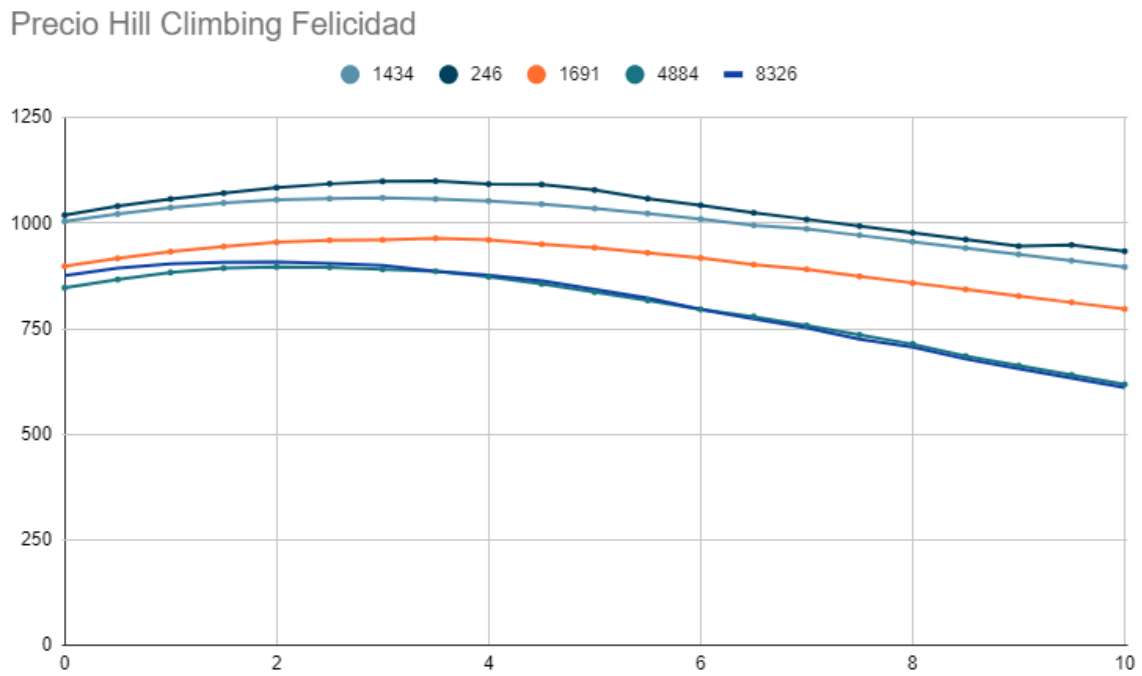


Figura 3.7.1 Coste heurístico en función de la felicidad en Hill Climbing

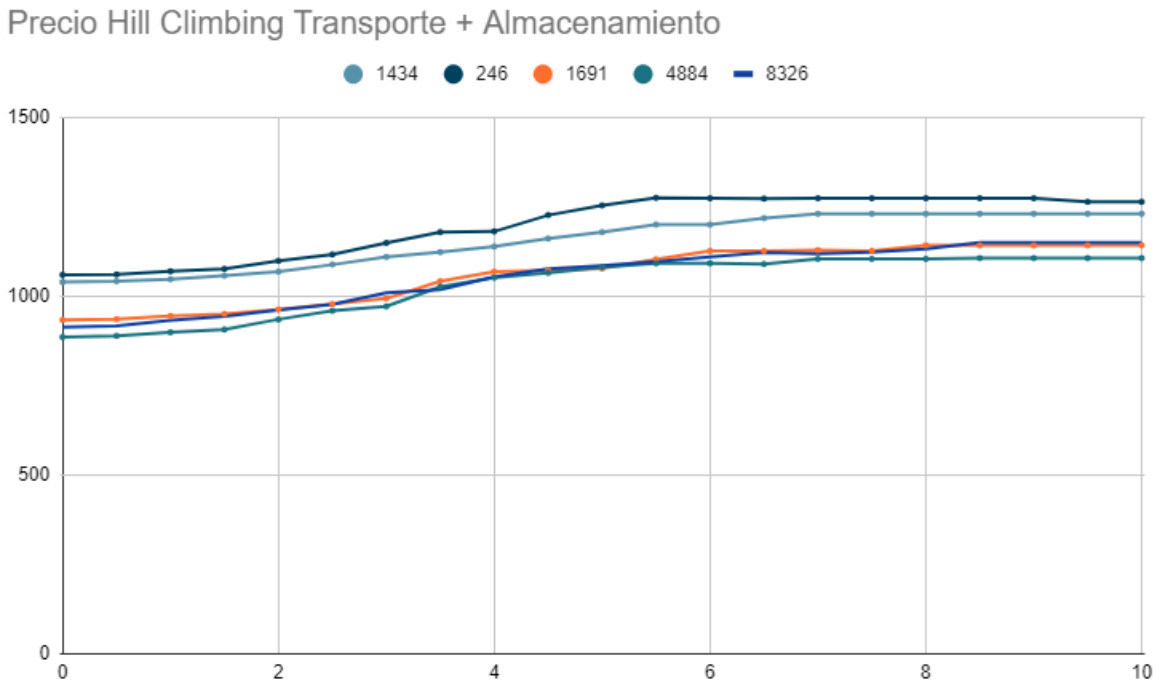


Figura 3.7.2 Coste del transporte y almacenamiento en función de la felicidad en Hill Climbing

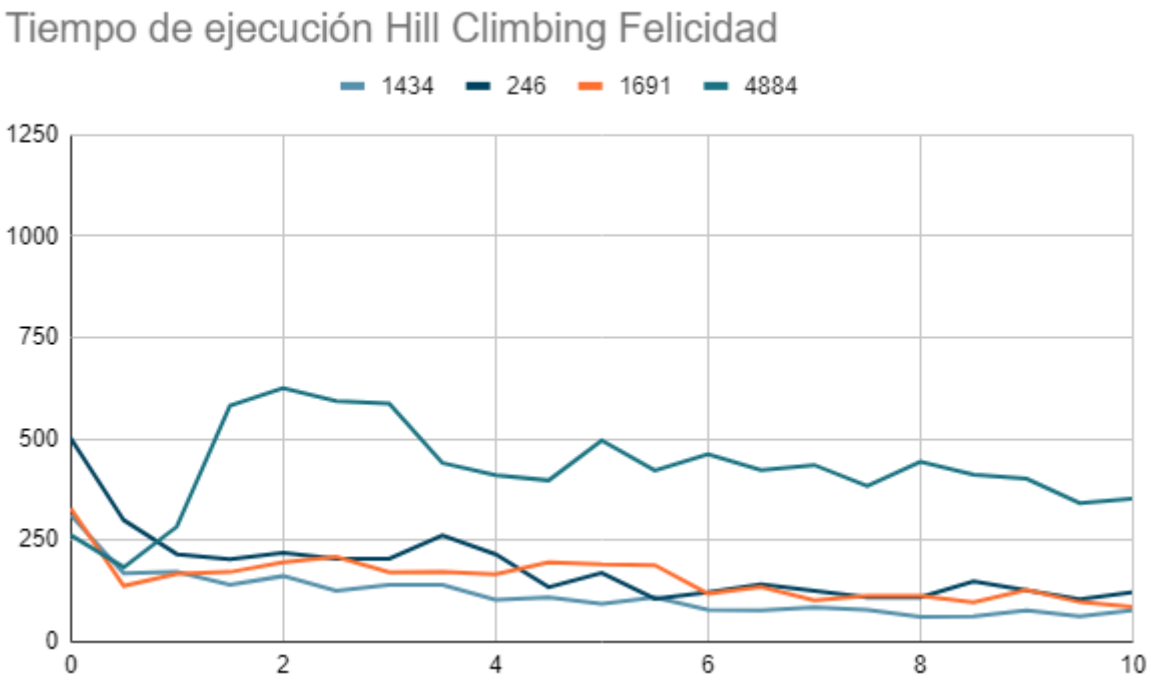


Figura 3.7.3 Tiempo de ejecución de Hill Climbing en función de la felicidad

Podemos comprobar como los resultados obtenidos cumplen con las observaciones del experimento 6, aunque en este caso vemos que a partir de 3.5 aproximadamente es cuando el valor del heurístico se va reduciendo.

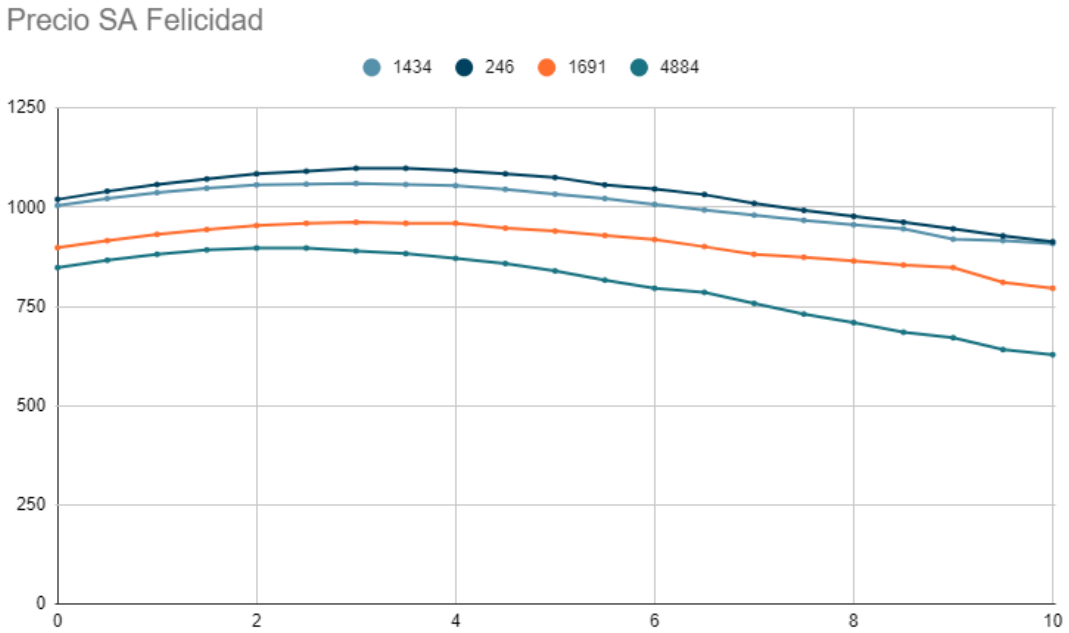


Figura 3.7.4 Coste heurístico en función de la felicidad en Simulated Annealing

Observamos que la hipótesis esperada se cumple, el comportamiento de Simulated Annealing es bastante similar, por no decir idéntico, al de Hill Climbing; además, como hemos comprobado en el experimento 3, obtenemos resultados mejores que con Hill Climbing al haber encontrado los parámetros óptimos.

Como en nuestra implementación calculamos el coste heurístico restando la felicidad al precio de transporte y almacenamiento, observamos que en los valores iniciales de la importancia a la felicidad el valor a restar es mínimo, pero al ir aumentando la importancia de recibir los paquetes antes, llega un momento en el que el valor de la felicidad disminuye el coste heurístico total considerablemente.

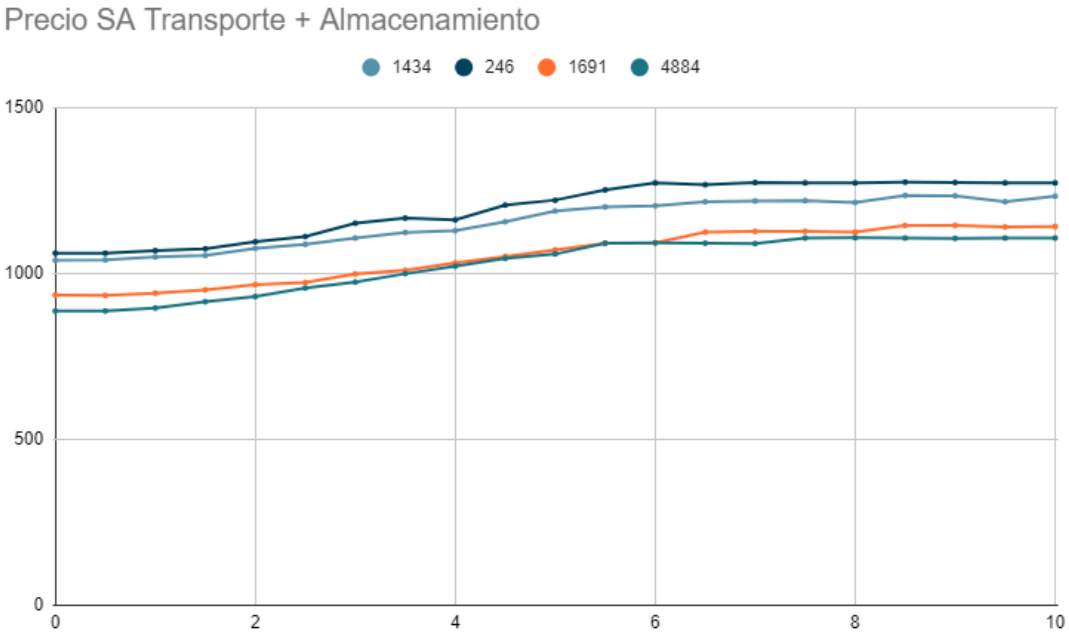


Figura 3.7.5 Coste del transporte y almacenamiento en función de la felicidad en Simulated Annealing

En esta gráfica podemos ver cómo afecta la felicidad al precio del transporte y almacenamiento. Por una parte, el precio del almacenamiento disminuirá ya que buscamos minimizar los días de entrega de los paquetes, y así aumentar la felicidad, pero por otro lado el coste del transporte aumentará, ya que las ofertas más rápidas son más caras. Como el coste de transporte es mayor al ahorro en almacenamiento, la suma de ambos resultará en una gráfica que irá aumentando el coste final progresivamente.

Podemos deducir que al igual que en Hill Climbing, el punto en el que se estabiliza el coste del transporte y almacenamiento llega un punto en el que se estabiliza ya que cuando la importancia que le damos a la felicidad coge un valor mayor a 6, los paquetes no pueden llegar antes, y por ende la felicidad no puede mejorar más.

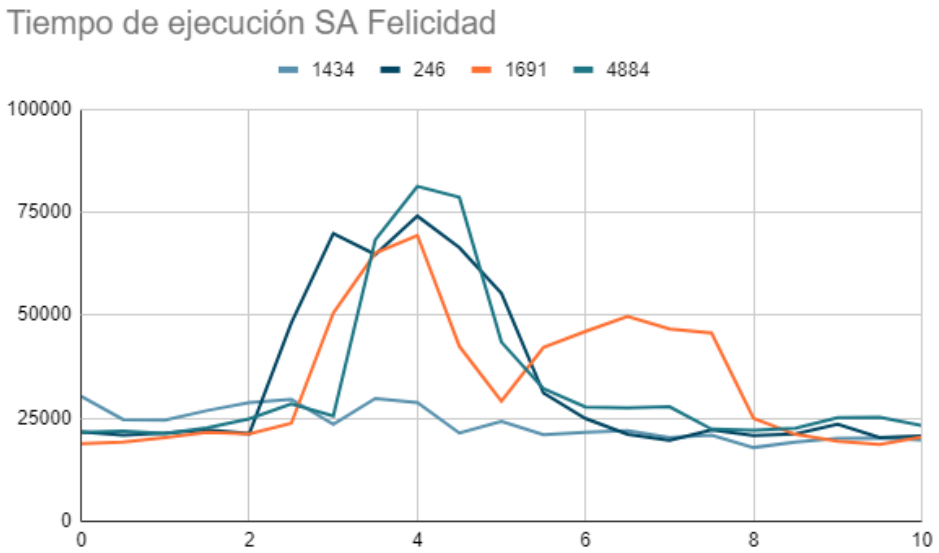


Figura 3.7.6 Tiempo de ejecución de Simulated Annealing en función de la felicidad

Observamos que hay un pico del tiempo de ejecución entre 2.5 y 4 de aproximadamente 80000 ms, que coinciden con los máximos de los costes que podemos observar en las respectivas gráficas (*Figura 3.7.4* y *Figura 3.7.5*) este gran aumento en el tiempo de ejecución se debe a que el algoritmo explora nuevas posibles soluciones priorizando la entrega de los paquetes intentando asignarlos a ofertas mejores.

3.8. Experimento 8

Si disminuimos el costo de almacenamiento diario, algunos paquetes que antes se enviaban en modalidad estándar podrían ahora almacenarse por más tiempo, ya que el ahorro en los costos de almacenamiento superaría el costo adicional de una entrega más rápida. Esto podría resultar en una reducción del costo total de envío.

Por otro lado, si aumentamos el costo de almacenamiento diario, algunos paquetes podrían migrar a modalidades de envío más rápidas, incluso si tienen un costo por kilo mayor, ya que el costo adicional de almacenar el paquete superaría el incremento en el costo de envío. Esto podría aumentar la satisfacción del cliente, al recibir sus pedidos en plazos más cortos.