LABORATORIO DE INTELIGENCIA ARTIFICIAL:

PRÁCTICA DE BÚSQUEDA LOCAL

Ibars Cubel, Albert

López Rodríguez, Iván

Santiago Corona, Daniel

Grupo 13

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Facultat d'Informàtica de Barcelona

FIB

Índice

1. Motivación

El problema presentado nos habla de una empresa que debe organizar los paquetes a enviar en una ciudad, y para ello trabajaremos con algoritmos de búsqueda local. Este tipo de algoritmos han sido presentados en las clases de teoría y por ello mismo ya estamos familiarizados con ellos.

Hemos tenido que pensar en la representación de los estados, que operadores utilizar para generar los estados sucesores, los generadores de solución inicial y las funciones heurísticas. Dado que en clases de teoría hemos resuelto ejercicios relativamente parecidos a esta práctica, nos ha ayudado para la definición de los parámetros que teníamos que implementar.

En nuestro caso hemos podido implementar dos generadores, cuatro operadores y las dos funciones heurísticas que se nos pedían. Por desgracia nos hubiera gustado disponer de más tiempo para la realización de la práctica, para así poder implementar otros parámetros, como por ejemplo nuevos generadores de solución inicial, o simplemente para profundizar más en la experimentación.

Tanto la implementación como la experimentación de la práctica nos han ayudado a acabar de comprender como funcionan los algoritmos de búsqueda local, ya que hemos podido estudiar el comportamiento de Hill Climbing y de Simulated Annealing y compararlos entre ellos.

2. Análisis del problema de estudio

El problema se nos presenta como un problema de optimización en la logística de una empresa. Dicha empresa precisa de enviar una cantidad determinada de paquetes en un día, cada uno de ellos con un peso y una prioridad, que determina el rango de días en los que se espera que llegue al destino. Para cumplir con este cometido, la empresa dispone cada día de un listado con las ofertas de las empresas de paquetería, cada una de estas ofertas tiene asociado un peso máximo que puede transportar, el número de días que tardará en entregar los paquetes contenidos en ella a su destino y el precio por kilo transportado en dicha oferta.

Una asignación se considera válida cuando todos los paquetes están asociados a alguna oferta, y las ofertas garantizan que los paquetes en ellas contenidos serán entregados en un período de tiempo igual o menor a los días máximos permitidos por la prioridad de éstos. Además, la suma de los pesos de los paquetes contenidos en las ofertas no debe superar el peso máximo transportable por cada oferta.

Los gastos de la empresa se calculan teniendo en cuenta que ésta incurre en los gastos de los envíos de los paquetes en las ofertas de las empresas de paquetería, además hay gastos de almacenamiento asociados a todos los paquetes que no tienen la prioridad mínima que deben ser tomados en cuenta para ser sumados al gasto. Aun así, el cliente paga por una parte del transporte de su paquete al pagar una cuota determinada asociada a la prioridad de su paquete, que debe ser tenida en cuenta como un gasto negativo en los gastos de la empresa, puesto que ayuda a sufragar los costes de todo el proceso.

El objetivo es encontrar una asignación válida entre todos los paquetes a enviar con ofertas disponibles que además minimizando los gastos de la empresa.

Además, el problema introduce un indicador de felicidad, una modelización de la satisfacción de los clientes con el servicio al recibir un paquete en una cantidad de días inferior al rango que él había solicitado y pagado. En varios experimentos se nos ha pedido que tengamos este indicador en cuenta, maximizándolo a la vez que se siguen minimizando los gastos de la empresa.

Pese a la adecuada contextualización del problema, lo identificamos como una instancia concreta del *"generalized assignment problem",* o el problema de asignación generalizado.

El problema de asignación generalizado es un problema matemático perteneciente a la optimización combinatoria. Dicho problema consiste en un número dado de tareas y agentes, cada uno con un tamaño distinto asignado, y requiere de la asignación de tareas a agentes consciente del coste o beneficio generado por dicha asociación. Además, cada agente tiene asociado un límite cuyo conjunto de tareas asociadas no puede superar. Se requiere encontrar la asociación que, respetando los límites de los agentes, maximice el beneficio o minimice el coste, en función de la instancia del problema.

Este problema es conocido por pertenecer al conjunto de problemas **NP-Hard**, así una generación exhaustiva de posibles valores para la asignación esta fuera de lugar para tamaños de problema más allá de la trivialidad.

Así, es lógico recurrir a las técnicas de inteligencia artificial para tratar este problema. Al analizar, sin entrar en detalles de implementación, el espacio de estados posible para este problema, podemos observar que es exponencialmente grande así que cualquier técnica alternativa a búsquedas a ciegas será bienvenida para tamaños de problema más allá de los pequeños. Dado

el tamaño que puede alcanzar el espacio de estados y la ausencia de un heurístico lo suficientemente bueno como para guiar la búsqueda por sí mismo en algoritmos de búsqueda heurística, recurriremos a los algoritmos de búsqueda local, operando en el espacio de soluciones.

Aunque nuestro objetivo en este apartado no es hablar de los detalles de implementación, con la información hasta ahora presentada puede ser fácil intuir que en la representación de estado o solución ha de existir la asignación entre paquetes y ofertas como principal elemento representativo del estado, y para este problema concreto es lógico que pueden existir soluciones que difieran entre ellas por solo una asignación. Esto genera un espacio de soluciones con vecindad entre soluciones, muy adecuado para problemas de búsqueda local.

Además, pese a que nuestra intención es minimizar al máximo posible los costes, no se nos pide el mínimo absoluto, así que consideraremos que encontrar un **buen** mínimo local está lo suficientemente bien, así que los algoritmos de búsqueda local encajan bien en nuestro objetivo.

3. Representación del problema como un problema de búsqueda local

A continuación, describiremos cómo hemos diseñado los elementos necesarios para realizar una búsqueda local adaptados al problema a resolver.

3.1. Estado del problema

La representación de un estado o solución del problema ha sido el elemento principal en la implementación del modelo de búsqueda local para resolver este problema. La implementación del estado del problema es la base sobre la que hemos podido pensar en cómo implementar los operadores, las funciones heurísticas y las generadoras.

Era crucial obtener una representación que fuera eficiente espacialmente, pues deberían crearse tantas instancias de estados como objetos exploremos. A la misma vez debíamos procurar una implementación lo suficientemente cómoda para poder trabajar en ella para crear operadores, generadores de solución inicial y que además ahorrará trabajo a la función heurística.

Así decidimos que la clase que implementaba el estado del problema debía contener:

- El objeto Paquetes, que contiene los diferentes paquetes a ser asignados a ofertas en el transcurso del problema, y el objeto Transporte, que contiene el conjunto de todas las ofertas de transporte en las que asignar los paquetes. Estos dos objetos permiten ser incluidos en la clase como atributos estáticos, evitando su copia por cada instancia de estado, ya que son los mismos en cualquier momento del problema.
- Un array de asignaciones, donde cada posición corresponde a un paquete y el contenido de cada posición corresponde a el número de oferta donde el paquete correspondiente ha sido asignado. Este vector corresponde con la solución, el objetivo es obtener este vector para el estado final que alcancemos.
- Un array de peso disponible, donde cada posición corresponde a una oferta y el contenido de cada posición es el peso disponible en la oferta correspondiente en el estado actual. Este array es de suma utilidad para la realización de cálculos en el resto del problema, por lo tanto, es un vector auxiliar.
- Un atributo que guarda el coste correspondiente a la asignación del estado actual. Mantener un atributo con el valor del coste actualizado es más eficiente que recalcularlo cada vez que se necesite.

3.2. Operadores y función generadora de sucesores

Uno de los primeros puntos a pensar fue que operadores podríamos utilizar para resolver el problema presentado.

Unos días antes de plantearnos qué conjunto de operadores utilizar, solucionamos un ejercicio en clase el cual mantiene una similitud con la práctica, por eso mismo, la idea de los primeros operadores a utilizar viene dada por ese ejercicio.

Así que como primera idea pensamos en implementar dos operadores: *movePackage* y *swapPackages*. El objetivo del primero es: a partir de un paquete en concreto comprobamos si existe una oferta donde lo podemos mover, dicha oferta debe ser diferente a la actual que está

asignado. En cambio, el *swapPackages*, a partir de un paquete miramos si existe otro paquete con el cual podamos intercambiar la asignación de oferta entre ellos.

Obviamente en el momento de aplicar estos operadores tenemos que verificar que estos movimientos sean válidos, es decir: no sobrepasar el peso que puede cargar una oferta, el envío de un paquete debe realizarse con una oferta que cumpla la prioridad de este y finalmente actualizar estos datos para próximas aplicaciones de los operadores.

Después de implementar estos operadores, tuvimos una reunión con nuestra profesora de laboratorio y llegamos a la conclusión que sería interesante tener un operador que fusiona los operadores anteriores.

De manera que decidimos implementar dos operadores más, un *swapMove* y un *moveSwap*. El primero consiste principalmente en aplicar el swap que teníamos antes y si no hemos encontrado ningún paquete para hacer el intercambio de ofertas, aplicamos el operador *move*. Por otra parte, el segundo es la fusión de *movePackage* y *swapPackages*, es decir para un paquete en concreto aplicamos tanto el operador *move* como el swap. En el apartado de experimentación explicamos cuál es el operador que nos ha dado mejores resultados.

3.3. Función heurística

Ahora hablaremos sobre las funciones heurísticas que se han implementado:

- *AzaHeuristicFunction1:* En este heurístico se minimiza el coste de almacenamiento y transporte. Este heurístico es válido ya que guiará el algoritmo en dirección a lo que propone el enunciado.
- *AzaHeuristicFunction2:* En este heurístico entra la felicidad en el papel, e intentará maximizar la felicidad pero sin pasarse en el coste, por lo que tenemos dos criterios con diferentes unidades así que hará falta un factor de ponderación para ajustarlo. Nosotros hemos planteado una resta ponderada tal que el heurístico es: *cost - 6*happiness*. Éste 6 es el factor de ponderación y el por qué lo observaremos con más detalle en el experimento número 6.

3.4. Generación del estado inicial

Las dos estrategias implementadas para la generación del estado inicial son:

- *generateInitSolution1:* Primeramente, ordenamos los paquetes por prioridad ascendente y en caso de igualdad de prioridad los paquetes menos pesados irán delante. También se ordenan las ofertas (*carriers*) por días de forma ascendente (de 1 a 5).
  Secuencialmente, recorriendo los vectores de izquierda a derecha, se asignan los paquetes a las ofertas siempre que haya espacio suficiente, en caso de que un paquete *i* no quepa en una oferta *j*, el paquete *i* intentará asignarse a la oferta *j+1, j+2*… y así hasta encontrar hueco. De esta manera estamos colocando los paquetes de forma que estén en la oferta con menor tiempo de entrega disponible, creando así una solución con bastante felicidad, a cambio de tener un coste más elevado.
- *generateInitSolution2:* Esta generación de estado es similar a la anterior. Ordenamos los paquetes y las ofertas de la misma forma, pero esta vez los paquetes se asignan de diferente manera, ya que recorreremos los paquetes de derecha a izquierda, se aplicará la misma política de asignación, pero invirtiéndola, en caso de un paquete *i* no quepa en una oferta *j*, el paquete *i* intentará asignarse a la oferta *j-1, j-2…* Cabe destacar que un paquete no será asignado a una oferta que no cumpla la restricción de prioridades. De este modo estamos colocando los paquetes a las ofertas más baratas, lo que resultará una asignación con un coste menor que con la solución 1, a coste de tener a los clientes descontentos.

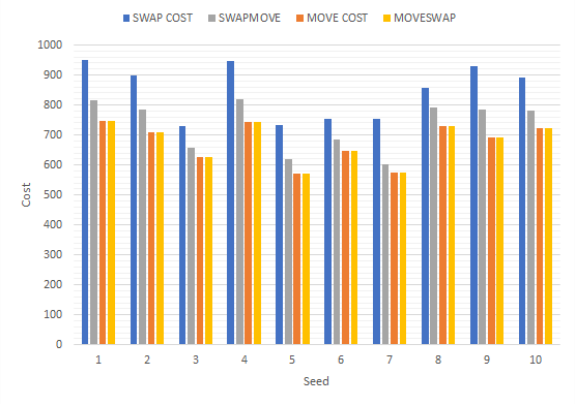
El coste de generar estas dos soluciones no es elevado; ordenar los paquetes $O(nlogn)$, ordenar las ofertas $O(nlogn)$, y luego la asignación paquete oferta que es coste lineal, que en caso peor es recorrer una vez los dos vectores.

4. Experimentación

4.1. Determinación de conjunto de operadores

Como primer experimento se nos ha pedido determinar qué operador nos da mejores resultados para el primer criterio presentado en la práctica. Las características de este experimento son:

- Observación: Puede haber operadores que obtengan mejores soluciones.
- Planteamiento: Escogemos los operadores presentados y observamos sus soluciones.
- Hipótesis: Todos los operadores son iguales (H0) o hay algunos mejores que otros.
- Método:
    - Elegiremos 10 semillas (1-10 en nuestro caso), una para cada réplica.
    - Ejecutaremos 1 experimento para cada semilla y para cada operador.
    - Experimentamos con los parámetros indicados en el enunciado: 100 paquetes y una proporción del peso igual a 1.2.
    - Usaremos el algoritmo de Hill Climbing.
    - Mediremos el coste de la solución final para realizar la comparación.



El gráfico adjunto nos muestra la información del experimento enunciado antes. Como podemos ver los operadores que nos dan un coste mínimo son *movePackage* y *moveSwap*, aún más, nos retornan el mismo coste. Por eso mismo nos tenemos que decantar por uno de estos dos. *MoveSwap* está haciendo tanto *move* como swap por cada paquete, es decir, los sucesores que nos está devolviendo, son la unión de los sucesores creados por el operador *move* y los del operador swap. Por tanto, el tiempo de computación de este es mayor que el del *movePackage*.

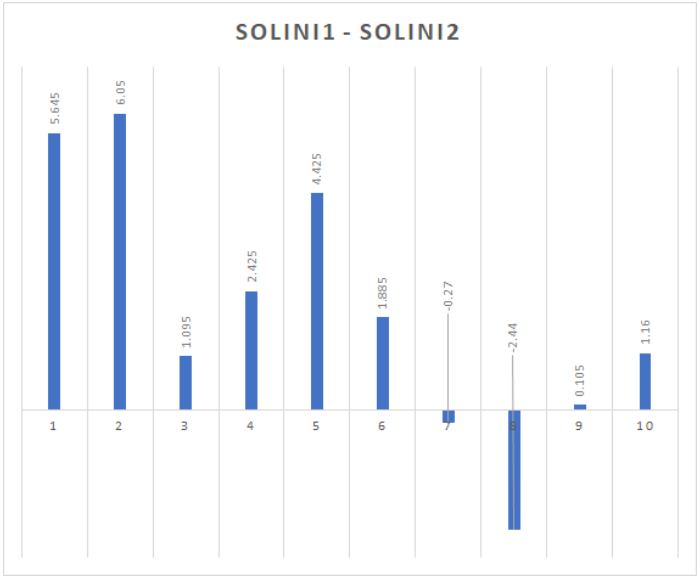
Como hemos podido comprobar, y por eso mismo nos da el mismo coste que el operador que solo aplica *move*, siempre elegimos los sucesores creados del operador m*ovePackage*. Por ese hecho nos decantamos por elegir el operador *movePackage* como el operador que da mejores resultados con los parámetros comentados anteriormente.

4.2. Determinación de generador de la solución inicial

En este segundo experimento tenemos que determinar cuál es la mejor generación de solución inicial entre las dos que hemos propuesto. Para ello vamos a hacer un experimento con estas características:

- Observación: Puede haber generadores de la solución inicial que obtengan mejores soluciones.

- Planteamiento: Escogemos los generadores de solución inicial presentados y observamos sus soluciones.
- Hipótesis: Todos los generadores de solución inicial son iguales (H0) o hay algunos mejores que otros.
- Método:
    - Elegiremos 10 semillas (1-10 en nuestro caso), una para cada réplica.
    - Ejecutaremos 1 experimento para cada semilla y para cada generador de solución inicial.
    - Experimentamos con los parámetros indicados en el enunciado: 100 paquetes y una proporción del peso igual a 1.2.
    - Usaremos el algoritmo de Hill Climbing.
    - Mediremos el coste de la solución final para realizar la comparación.



El gráfico que tenemos encima se ha calculado restando el coste obtenido con el primer generador de solución inicial y el coste obtenido con el segundo generador de solución inicial. Como podemos observar prácticamente en todos los casos la resta muestra un valor positivo, lo que quiere decir que el coste obtenido con el primer generador es mayor que el obtenido con el segundo.
A partir de estos resultados, hemos decidido quedarnos con el segundo generador de solución inicial para los próximos experimentos.

4.3. Determinación de parámetros de Simulated Annealing

Para la elección de los parámetros del algoritmo Simulated Annealing utilizaremos una estrategia algo diferente a la del resto de experimentos, aunque sigue siendo un proceso totalmente experimental. Como punto de partida para los parámetros de Simulated Annealing, hemos consultado el enunciado de la práctica, así como la implementación de la demostración del problema TSP en los ficheros fuente de AIMA.

Para el diseño de los experimentos se ha utilizado la misma metodología para los cuatro parámetros, sobre 5 semillas aleatorias se ha ejecutado el algoritmo con diferentes valores del parámetro de estudio mientras el resto de parámetros se han mantenido como variables fijas.

4.3.1. Parámetro k

El parámetro k en Simulated Annealing determina el tiempo (o proporción de iteraciones sobre las totales) que tarda el algoritmo en empezar a rechazar movimientos o soluciones que no son mejores que la actual. Cuanto mayor es k, más tiempo transcurre hasta que empieza a descartar algunos de estos movimientos.

Para determinar un valor de k beneficioso para nuestro experimento probaremos diversos valores de k mientras fijamos el número de iteraciones en 10.000 (suficientemente grande para observar los efectos), el número de iteraciones por cambio de temperatura en 1000 (un total de 100 cambios de temperatura) y un valor de λ = 0.001 que hemos obtenido de la implementación de la demo del AIMA como referencia.

Los resultados de la experimentación no encontraron diferencias relevantes en los valores promedio del coste para las diferentes semillas probando diferentes valores de k. Así parece que, en nuestro experimento, no es un factor importante el tiempo que pasamos en las regiones más calientes del algoritmo, con la mayor probabilidad de aceptar estados peores.

K	MEAN COST
1	672,473
5	672,2566
25	673,265
125	672,437

Dado que apenas hay movimiento para diferentes valores de k, como mostramos en la tabla de los resultados, nos quedaremos con el valor inicial k = 5, que además es el que da un mejor resultado, aunque por un margen mínimo

4.3.2. Parámetro λ

El parámetro λ en Simulated Annealing determina la rapidez con la que disminuye la probabilidad de movimientos o soluciones peores a la actual, una vez la probabilidad ya ha empezado a caer (comportamiento que determina el parámetro k).

Para la experimentación hemos mantenido los valores del número de iteraciones (10.000) y del número de iteraciones por cambio de temperatura (100) utilizados en el apartado anterior, y fijamos ahora k=5 por los resultados del experimento anterior.

Los resultados de la experimentación, si bien son más notables los efectos de diferentes valores que en el caso anterior, no muestran un efecto muy grande en los costes obtenidos.

λ	MEAN COST
0,0001	692,711
0,001	672,2566
0,01	674,459
1	678,95

Como podemos observar en la tabla, el valor λ = 0.001 presenta los mejores resultados, sin obtener beneficio en los resultados el utilizar valores extremos para λ.

4.3.3. Número de iteraciones

El número de iteraciones es el número fijo de iteraciones que realizará el algoritmo antes de retornar un resultado o solución final. Simulated Annealing, a diferencia de Hill Climbing, realiza un número fijo de iteraciones.

Es intuitivo pensar que un número de iteraciones demasiado pequeño no nos permitirá que el coste converja en un mínimo local, mientras que una cantidad demasiado grande de iteraciones realizará trabajo redundante al no mejorar la solución respecto al número óptimo de iteraciones.

Manteniendo los valores de los parámetros k y λ conseguidos en los anteriores experimentos, y siguiendo utilizando 100 iteraciones como el nivel de cambio de temperatura, hemos probado sucesivos números de iteraciones (todos múltiplos de 100) y hemos conseguido los siguientes resultados que además son consistentes con nuestra intuición inicial de cómo harían variar el coste.

Observamos que el mejor coste medio se ha conseguido con 10.000 iteraciones. con 400 iteraciones el coste es bastante peor ya que debemos estar en el punto en el que el coste no tiene iteraciones suficientes para converger en el mínimo local. Mientras en 50.000 iteraciones apenas observamos diferencia en el coste de 10.000 iteraciones, indicando que todas esas iteraciones de más son redundantes, ya que sirvieron 10.000 para converger.

NUM_IT	MEAN COST
400	708,44
2000	683,903
10000	672,2566
50000	672,259

4.3.4. Número de iteraciones por cambio de temperatura

Una vez fijados los valores apropiados para k, λ y el número de iteraciones fijaremos el número de iteraciones por cada cambio en el nivel de temperatura. El nivel del cambio de temperatura es un valor que junto al parámetro λ determina la caída de la probabilidad de aceptar soluciones peores que la actual. Un valor muy grande de iteraciones por nivel comportaría una discretización muy acusada del comportamiento del algoritmo al generar pocos niveles con comportamiento distinto. Por otro lado, cuantas menos iteraciones por nivel haya más, el comportamiento de los noveles será más continuo y progresivo, aunque un número muy elevado de niveles puede conllevar demasiadas pocas iteraciones por nivel y muy pocas iteraciones pueden ser muestras poco representativas.

STEP_ITER	MEAN COST
1	672,667
50	672,757
100	672,2566
200	672,406

En la práctica hemos encontrado que para el rango de valores escogido ha habido muy poco efecto en el coste.

Escogemos el valor de 100 iteraciones por nivel de temperatura, ya que es el de menor coste medio, aunque en este caso los resultados obtenidos son muy parecidos para todos los casos.

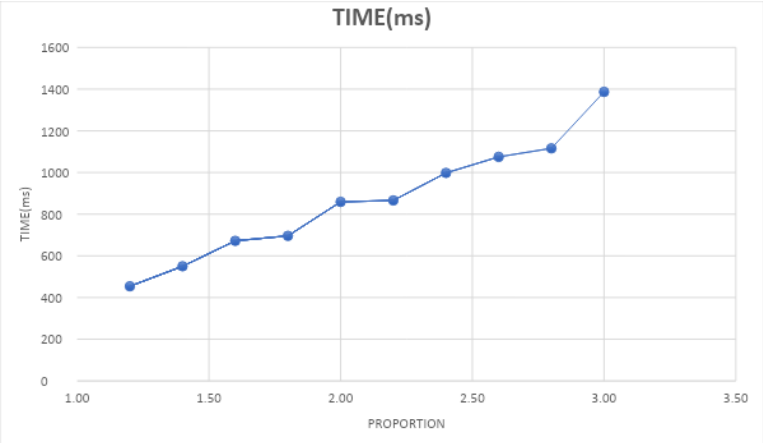
4.4. Evolución del tiempo de ejecución

Estudiaremos cómo evoluciona el tiempo de ejecución manteniendo el número de paquetes e incrementando la proporción del peso transportable, y cómo se comporta si mantenemos la proporción e incrementamos el número de paquetes.

4.4.1. En función de la proporción del peso trasportable

Como ya hemos comentado antes, en el primer experimento que vamos a realizar mantenemos el número de paquetes a 100. La proporción del peso transportable empezará en 1,2 e irá aumentando de 0,2 en 0,2. Las características del experimento son las siguientes:

- Observación: La proporción de peso transportable puede afectar al tiempo total de ejecución.
- Planteamiento: Encontraremos soluciones para diferentes valores de la proporción, manteniendo estables el resto de parámetros y estudiaremos el tiempo de ejecución.
- Hipótesis: La proporción de peso transportable no afecta al tiempo de ejecución (H0) o sí lo hace.
- Método:
    - Elegiremos 10 semillas (1-10 en nuestro caso), una para cada réplica.
    - Ejecutaremos 1 experimento para cada iteración sobre las 10 semillas y por cada valor de la proporción.
    - Experimentamos con los parámetros indicados en el enunciado: 100 paquetes y una proporción del peso entre 1.2 y 3.0.
    - Usaremos el algoritmo de Hill Climbing.
    - Mediremos para cada valor de la proporción la suma de los tiempos de ejecución de las 10 semillas para realizar la comparación.

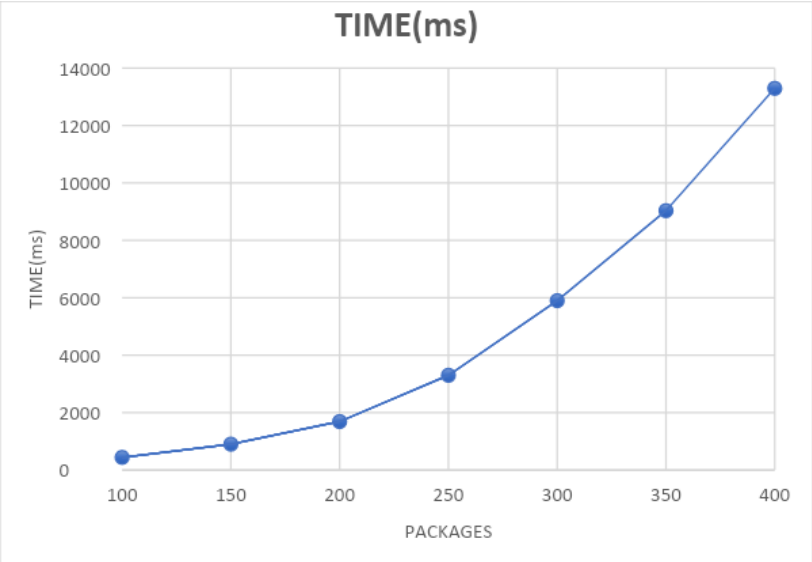


Podemos observar claramente como el hecho de aumentar la proporción del peso transportable afecta directamente al tiempo de ejecución. Esto viene dado a que disponer de más peso transportable, se traduce a tener un mayor número de ofertas. Como el operador Move comprueba todas las asignaciones posibles, supondrá un incremento en el tiempo de ejecución, además, por cómo funciona Hill Climbing, tendrá que mirar entre todos los estados generados, cual devuelve un menor coste, lo que también se traduce en un aumento del tiempo de ejecución.

4.4.2. En función del número de paquetes

El segundo experimento de este apartado estudiará la evolución del tiempo de ejecución cuando mantenemos la proporción del peso transportable a 1,2, comenzaremos con 100 paquetes y lo iremos incrementando de 50 en 50. Definimos el experimento como:

- Observación: El número de paquetes puede afectar al tiempo total de ejecución.
- Planteamiento: Encontraremos soluciones para diferente número de paquetes, manteniendo estables el resto de parámetros y estudiaremos el tiempo de ejecución.
- Hipótesis: El número de paquetes no afecta al tiempo de ejecución (H0) o sí lo hace.
- Método:
    - Elegiremos 10 semillas (1-10 en nuestro caso), una para cada réplica.
    - Ejecutaremos 1 experimento para cada iteración sobre las 10 semillas y por cada valor del número de paquetes.
    - Experimentamos con los parámetros indicados en el enunciado: entre 100 y 400 paquetes y una proporción del peso igual a 1.2.
    - Usaremos el algoritmo de Hill Climbing.
    - Mediremos para cada valor del número de paquetes los tiempos de ejecución de las 10 semillas para realizar la comparación.



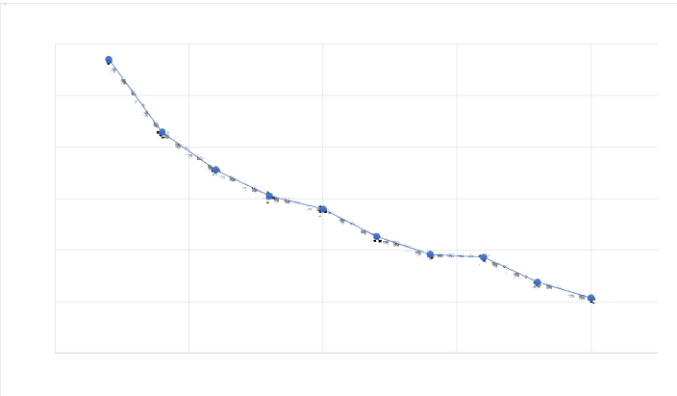
Pensábamos que el hecho de aumentar el número de paquetes afectaría al tiempo total de ejecución, pero no de una manera tan brusca. En los primeros aumentos, el tiempo se estaba doblando por cada iteración. Cuando llegamos a 300 paquetes, el tiempo obtenido no es el doble, pero, aunque no lo sea, cabe destacar que no deja de aumentar de forma drástica.

Por cómo trabaja el operador Move, el hecho de incrementar el número de paquetes, supone un mayor trabajo tanto para el operador, como para el algoritmo de búsqueda local (Hill Climbing en este caso). El operador iterará sobre un mayor número de paquetes y podrá crear más estados sucesores. Hill Climbing recibirá un mayor número de sucesores, de entre los cuales, tendrá que comprobar cuál es el mejor para quedarse con ese.

4.5. Evolución del coste en función de la proporción del peso transportable

La proporción de peso transportable es la fracción que representa el peso total transportable por todas las ofertas entre el peso de los paquetes a transportar. Sabemos que esta proporción siempre será superior a 1 por construcción, y cuanto mayor sea mayor será la holgura con la que los paquetes pueden ser asignados a las ofertas.

- Observación: La proporción de peso transportable podría afectar al coste final obtenido.
- Planteamiento: Encontraremos soluciones para diferentes valores de la proporción manteniendo estables el resto de parámetros y estudiaremos la evolución del coste.
- Hipótesis: La proporción de peso transportable no afecta al coste final obtenido (H0) o sí lo hace.
- Método:
    - Elegiremos 10 semillas (1-10 en nuestro caso), una para cada réplica.
    - Ejecutaremos 1 experimento para cada iteración sobre las 10 semillas y por cada valor de la proporción.
    - Experimentamos con los parámetros indicados en el enunciado: 100 paquetes y una proporción del peso igual entre 1.2 y 3.00.
    - Usaremos el algoritmo de Hill Climbing.
    - Mediremos para cada valor de la proporción la suma de los costes de las soluciones finales de las 10 semillas para realizar la comparación.



Los resultados del experimento indican de manera clara que la hipótesis H0 es falsa ya que la proporción de peso está afectando al coste total de la solución. Concretamente observamos que el coste disminuye cuando aumenta la proporción de peso transportable. De hecho, esto es intuitivo ya que como comentamos al principio una mayor proporción de peso transportable aporta una mayor holgura a la hora de colocar ofertas.

Desde este punto de vista parece interesante seguir aumentando la proporción de peso, sin embargo, tal y como observamos en el anterior experimento un aumento de la proporción también resulta en un aumento del tiempo de ejecución necesario, que puede llegar a ser bastante alto para valores altos. Además, es discutible si es realista en el mundo real que exista una proporción tan grande entre los paquetes que se pueden transportar en un día y los que realmente se han de transportar.

4.6. Evolución del coste de transporte y almacenamiento en función de la felicidad

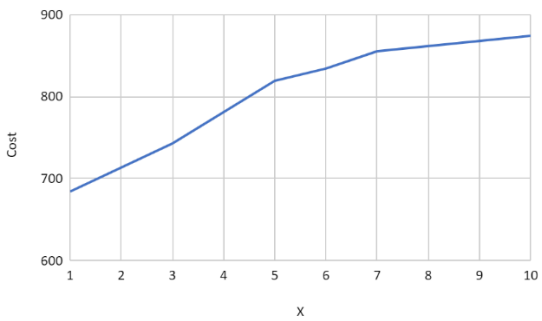
En este experimento queremos estudiar cómo afecta la felicidad al coste de transporte y almacenamiento. Como dice el enunciado, el escenario será como el del primer apartado, pero con este concepto de felicidad en la función heurística. Adelantándonos, esta función será del tipo *coste – X*felicidad*.

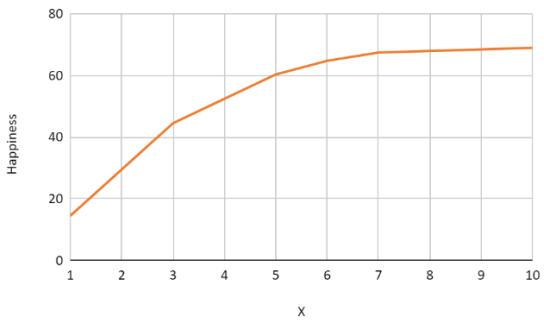
- Observación: Al querer entregar los paquetes antes del plazo mínimo para tener a los clientes "felices", provocará un aumento en el coste de entrega.
- Planteamiento: Generamos diversas soluciones aplicando esta nueva heurística con distinto "peso" de felicidad y observamos los resultados.
- Hipótesis: Cuanto más peso tenga la felicidad más aumentará el coste (H0).
- Método:
    - Elegiremos 10 semillas (1-10 en nuestro caso), una para cada réplica.
    - Experimentamos con los parámetros del experimento 1: 100 paquetes y una proporción del peso igual a 1.2.
    - Usaremos el algoritmo de Hill Climbing, el operador mover y el generador solución inicial 2.
    - Función heurística 2: coste – X*felicidad (*nos dedicaremos a estudiar cuánto vale esa X*).

Para cada valor de X (*X es el factor de ponderación de la felicidad en la heurística*) obtendremos su respectivo coste y felicidad, se obtendrá haciendo la media de las diez semillas.

Estas son las medias obtenidas para diferente valor X.

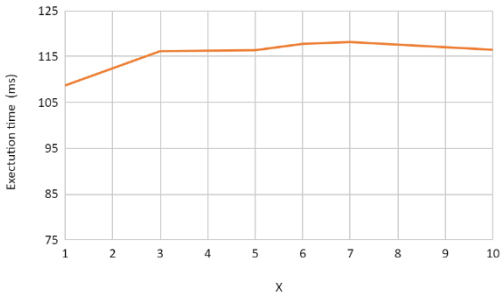
X	COST HC	HAPPINESS HC
1	684,172	14,5
3	742,83	44,5
5	819,31	60,3
6	834,12	64,7
7	855,23	67,4
10	874,161	68,9





Como podemos observar a medida que X es más grande sube el precio de la entrega, y esto es a causa de que estamos aumentando el peso de la felicidad. Fijándonos en la gráfica de felicidad,

una ponderación baja de felicidad reduce el coste, pero tiene a menor gente contenta, los valores X anteriores a 5 se observa un crecimiento rápido, entre 5 y 7 se tranquiliza un poco más este crecimiento y ya a partir de 7 el crecimiento se aplana. Así pues, asignar el valor X entre 5 y 7 nos parece muy acertado, decidiendo ya cada uno si prefiere pagar un poco más para tener a los clientes más felices o a la inversa.



En la izquierda encontramos el tiempo de ejecución (*ms*) para los diferentes valores de X. El tiempo se mantiene constante a partir de 3, pero entre 1 y 3 ha habido un pequeño incremento de tiempo y esto es debido a que como la solución inicial estaba decantada para reducir el precio y la felicidad tiene muy poco peso, no ha tenido que recorrer mucho para encontrar una solución.

Tabla con los resultados obtenidos con nuestro valor escogido X = 6 para Hill Climbing.

SEED	INIT. COST	COST	HAPPINESS
1	820,01	942	64
2	741,99	885,73	65
3	656.84	724,82	48
4	795.33	890,49	73
5	620.90	751,07	78
6	676,13	759,9	59
7	624,26	767,21	88
8	774,99	881,3	54
9	733,53	888,53	64
10	767,09	850,64	54
Average	734	834,12	64,7
Deviation	65,36	75,77	12

4.7. Experimentación con Simulated Annealing

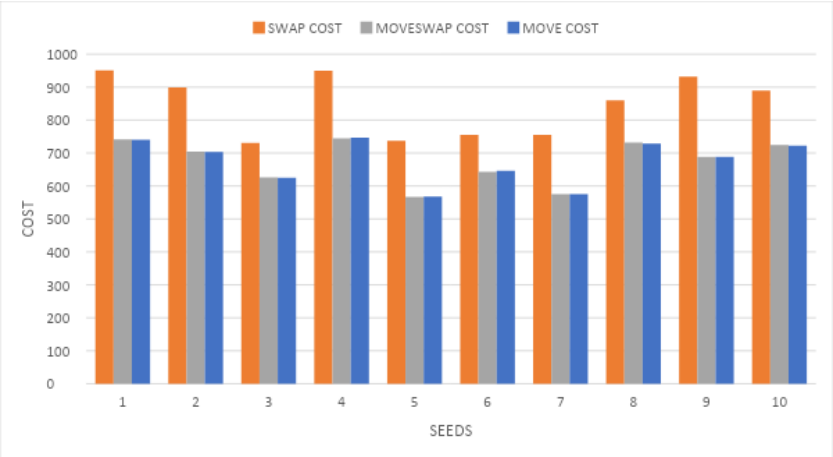
En este apartado vamos a hacer de nuevo los anteriores experimentos, pero ahora aplicando el algoritmo de Simulated Annealing.

4.7.1. Determinación de conjunto de operadores

Vamos a hacer de nuevo el primer experimento, pero en este caso aplicaremos el algoritmo de Simulated Annealing. Cuando implementábamos el operador *SwapMove*, nos dimos cuenta que no tiene sentido aplicarlo para este algoritmo, ya que no había forma de aplicarle la aleatoriedad. Dicho esto, experimentaremos con los tres operadores restantes:

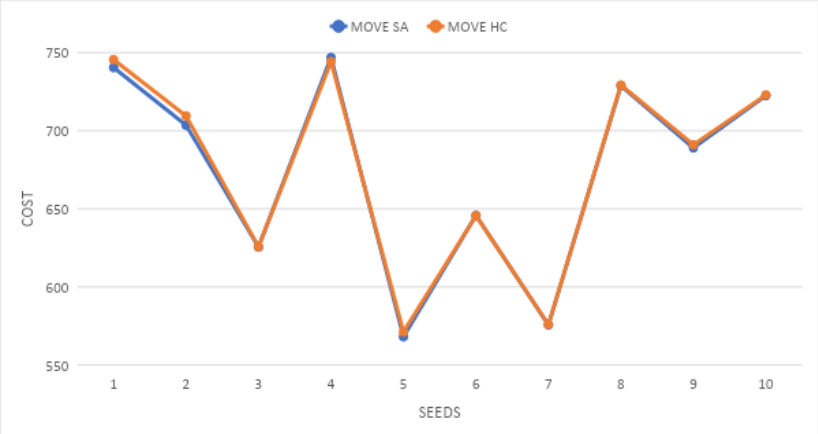
- Observación: Puede haber operadores que obtengan mejores soluciones.
- Planteamiento: Escogemos los operadores presentados y observamos sus soluciones.
- Hipótesis: Todos los operadores son iguales (H0) o hay algunos mejores que otros.
- Método:
    - Elegiremos 10 semillas (1-10 en nuestro caso), una para cada réplica.
    - Ejecutaremos 1 experimento para cada semilla y para cada operador.

- Experimentamos con los parámetros indicados en el enunciado: 100 paquetes y una proporción del peso igual a 1.2.
- Usaremos el algoritmo de Simulated Annealing.
- Mediremos un parámetro para realizar la comparación.



En el gráfico adjunto se puede observar como el operador Swap devuelve soluciones con mayores costes, mientras tanto, Move y MoveSwap nos devuelven el mismo coste. Por lo tanto, nos encontramos con el mismo caso que en el experimento de Hill Climbing. Como ya comentamos, MoveSwap calcula la unión de sucesores creados por los operadores Move y Swap, dado que nos devuelve el mismo coste que Move, quiere decir que nunca escoge un sucesor creado por el operador Swap. Teniendo en cuenta el trabajo computacional extra que hace el MoveSwap, volvemos a escoger el operador Move para próximos experimentos.

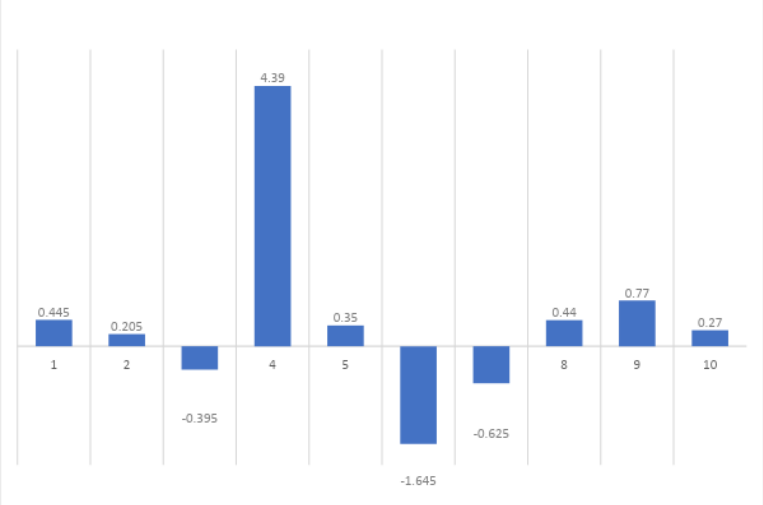
Ahora vamos a comparar los valores obtenidos con los diferentes algoritmos. Como se puede observar en el gráfico que tenemos justo debajo, el operador Move está obteniendo prácticamente los mismos estados finales para los dos algoritmos. Aunque en este experimento no mostramos los tiempos de ejecución (hay experimentos que hablaran de ello), cabe destacar que el Simulated Annealing, aunque devuelva soluciones parecidas a las del Hill Climbing, su ejecución es mucho más rápida que la del Hill Climbing.



4.7.2. Determinación de generador de la solución inicial

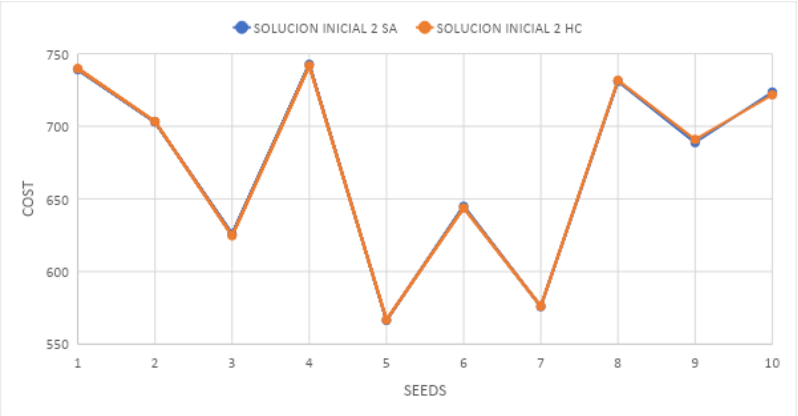
A continuación, ejecutaremos de nuevo el segundo experimento, pero en este caso aplicando el algoritmo de Simulated Annealing:

- Observación: Puede haber generadores de la solución inicial que obtengan mejores soluciones.
- Planteamiento: Escogemos los generadores de solución inicial presentados y observamos sus soluciones.
- Hipótesis: Todos los generadores de solución inicial son iguales (H0) o hay algunos mejores que otros
- Método:
    - Elegiremos 10 semillas (1-10 en nuestro caso), una para cada réplica.
    - Ejecutaremos 1 experimento para cada semilla y para cada generador de solución inicial.
    - Experimentamos con los parámetros indicados en el enunciado: 100 paquetes y una proporción del peso igual a 1.2.
    - Usaremos el algoritmo de Simulated Annealing.
    - Mediremos un parámetro para realizar la comparación.



Nos encontramos de nuevo con un resultado similar al obtenido cuando aplicamos Hill Climbing. El gráfico muestra la diferencia de los costes obtenidos del primer generador con los del segundo. Se puede observar que para la mayoría de las semillas obtenemos un valor positivo, lo que quiere decir que a partir del primer generador llegamos a estados finales con mayores costes. Así que, de la misma forma que en el experimento del Hill Climbing, a partir de lo que hemos comentado previamente, decidimos quedarnos con el segundo generador de soluciones.

Vamos a comparar la aplicación del segundo generador entre los dos algoritmos.



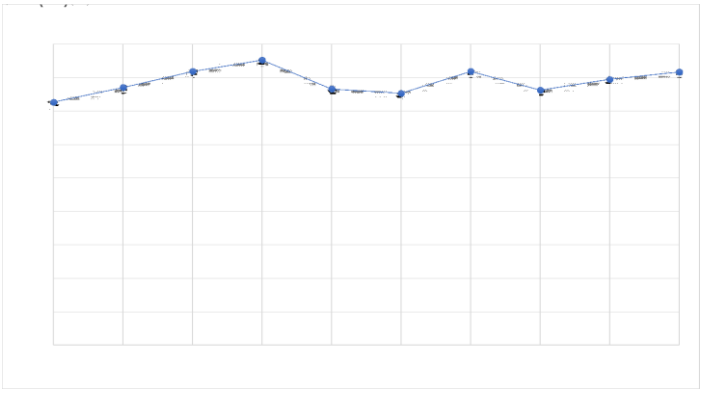
Observando los resultados obtenidos se ve perfectamente como el segundo generador funciona prácticamente de la misma manera para ambos algoritmos. Como comentamos en el anterior experimento, aunque estén funcionando de manera prácticamente idéntica, hay que tener en cuenta que Simulated Annealing tiene un menor tiempo de ejecución.

4.7.3. Evolución del tiempo de ejecución

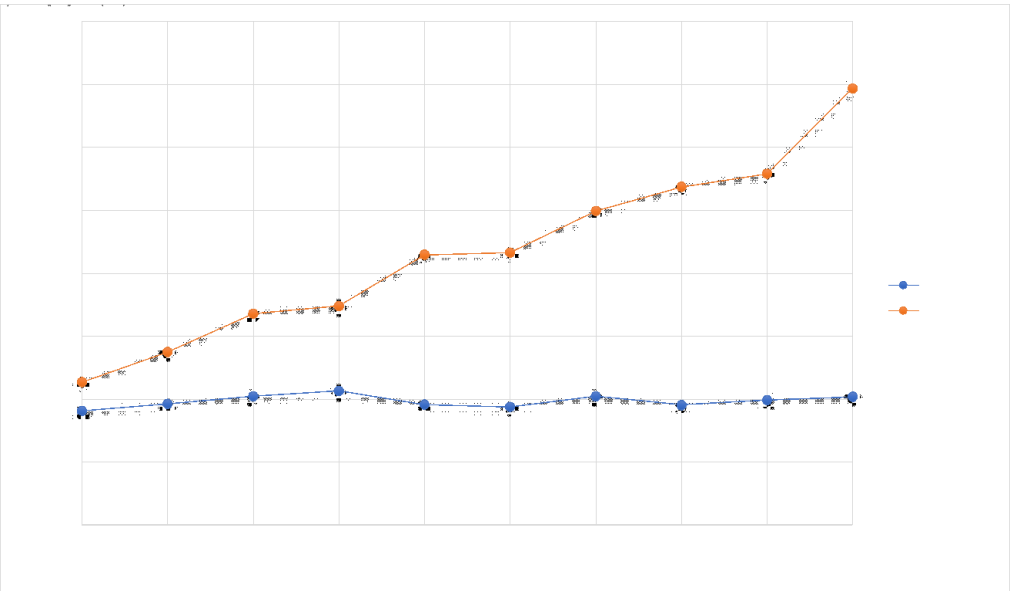
4.7.3.1. En función de la proporción de peso transportable

Este experimento tiene como objetivos estudiar cómo afecta la proporción de peso transportable al tiempo de ejecución.

- Observación: La proporción de peso transportable puede afectar al tiempo total de ejecución.
- Planteamiento: Encontraremos soluciones para diferentes valores de la proporción, manteniendo estables el resto de parámetros y estudiaremos el tiempo de ejecución.
- Hipótesis: La proporción de peso transportable no afecta al tiempo de ejecución (H0) o sí lo hace.
- Método:
    - Elegiremos 10 semillas (1-10 en nuestro caso), una para cada réplica.
    - Ejecutaremos 1 experimento para cada iteración sobre las 10 semillas y por cada valor de la proporción.
    - Experimentamos con los parámetros indicados en el enunciado: 100 paquetes y una proporción del peso entre 1.2 y 3.00.
    - Usaremos el algoritmo de Simulated Annealing.
    - Mediremos para cada valor de la proporción la suma de los tiempos de ejecución de las 10 semillas para realizar la comparación.



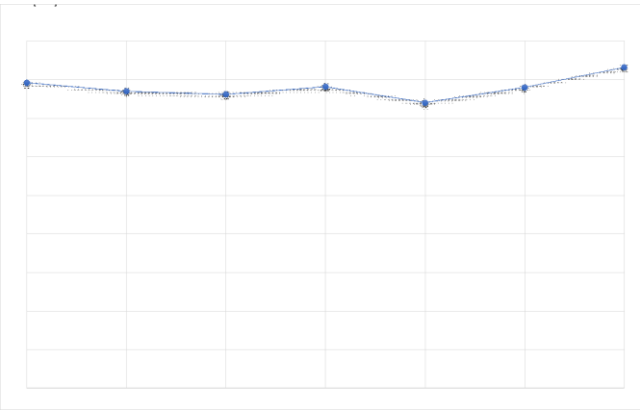
A la vista de los resultados obtenidos no podemos encontrar una relación entre aumentar la proporción de peso transportable y una variación del tiempo de ejecución. Esto es remarcable, especialmente en comparación con los resultados obtenidos para Hill Climbing, donde de manera clara encontramos un aumento drástico del tiempo de ejecución conforme aumentaba la proporción de peso transportable. En caso de que las soluciones obtenidas por ambos algoritmos sean igual de buenas, cosa que estudiaremos en el siguiente experimento, esto supondría un gran punto de ventaja para Simulated Annealing.



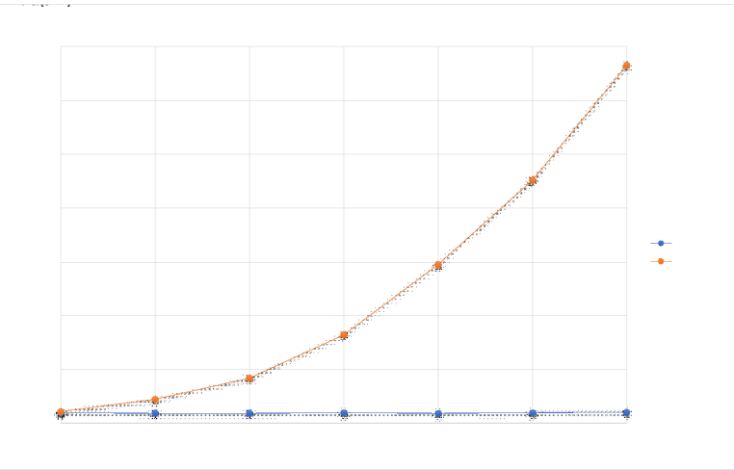
4.7.3.2. En función del número de paquetes

Este experimento tiene como objetivos estudiar cómo el número de paquetes transportado al tiempo de ejecución.

- Observación: El número de paquetes transportado puede afectar al tiempo total de ejecución.
- Planteamiento: Encontraremos soluciones para diferentes valores del número de paquetes transportado, manteniendo estables el resto de parámetros y estudiaremos el tiempo de ejecución.
- Hipótesis: El número de paquetes transportado no afecta al tiempo de ejecución (H0) o sí lo hace.
- Método:
    - Elegiremos 10 semillas (1-10 en nuestro caso), una para cada réplica.
    - Ejecutaremos 1 experimento para cada iteración sobre las 10 semillas y por cada valor de la proporción.
    - Experimentamos con los parámetros indicados en el enunciado: un número de paquetes transportados entre 100 y 400 y una proporción del peso de 1.2.
    - Usaremos el algoritmo de Simulated Annealing.
    - Mediremos para cada valor de la proporción la suma de los tiempos de ejecución de las 10 semillas para realizar la comparación.



Como para el experimento anterior, no encontramos una correlación entre el número de paquetes y el tiempo de ejecución. Este hecho es verdaderamente sorprendente, ya que Simulated Annealing no está aumentando su tiempo de ejecución pese a estar tratando con un problema claramente mayor. Otra vez, si en posteriores experimentos comprobamos que las soluciones son igual de buenas para ambos, el tiempo de ejecución de Simulated Annealing será una gran ventaja sobre su rival.

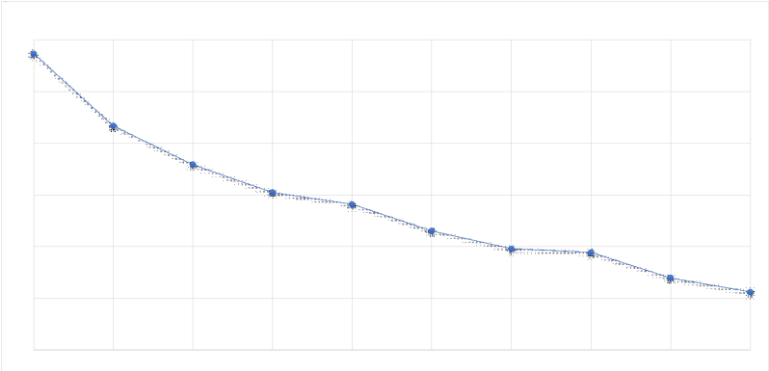


4.7.4. Evolución del coste en función de la proporción del peso transportable

La proporción de peso transportable es la fracción que representa el peso total transportable por todas las ofertas entre el peso de los paquetes a transportar. Sabemos que esta proporción siempre será superior a 1 por construcción, y cuanto mayor sea mayor será la holgura con la que los paquetes pueden ser asignados a las ofertas.

- Observación: La proporción de peso transportable podría afectar al coste final obtenido.
- Planteamiento: Encontraremos soluciones para diferentes valores de la proporción manteniendo estables el resto de parámetros y estudiaremos la evolución del coste.
- Hipótesis: La proporción de peso transportable no afecta al coste final obtenido (H0) o sí lo hace.
- Método:
    - Elegiremos 10 semillas (1-10 en nuestro caso), una para cada réplica.
    - Ejecutaremos 1 experimento para cada iteración sobre las 10 semillas y por cada valor de la proporción.
    - Experimentamos con los parámetros indicados en el enunciado: 100 paquetes y una proporción del peso entre 1.2 y 3.00.

- Usaremos el algoritmo de Simulated Annealing.
- Mediremos para cada valor de la proporción la suma de los costes de las soluciones finales de las 10 semillas para realizar la comparación.



Los resultados del experimento indican de manera clara que la hipótesis H0 es falsa ya que la proporción de peso está afectando al coste total de la solución. Estos resultados coinciden con los obtenidos en el experimento equivalente para Hill Climbing en el descenso del coste conforme crece la proporción de coste transportable, y todos los comentarios que realizamos en aquél experimento también son aplicables para éste.

De hecho, al intentar superponer las dos gráficas con los resultados para cada algoritmo, los resultados se superponen casi completamente por lo parecido de los costes obtenidos. Este hecho sigue demostrando que los resultados obtenidos por Simulated Annealing son tan buenos como los obtenidos por Hill Climbing, pero tal y como vimos en el anterior experimento, los tiempos de ejecución de Simulated Annealing pueden llegar a ser varias órdenes de magnitud menores, siendo esto una clara ventaja para este algoritmo.

4.7.5. Evolución del coste de transporte y almacenamiento en función de la felicidad

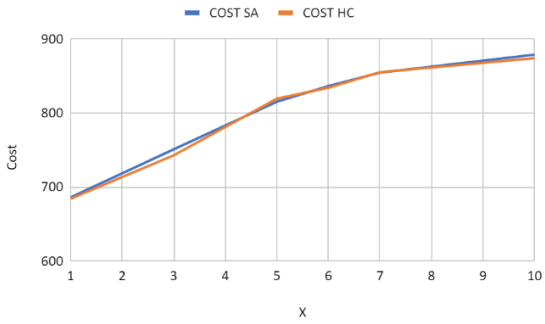
En este experimento queremos estudiar cómo afecta la felicidad al coste de transporte y almacenamiento. Este experimento será exactamente igual que el 6 pero en Simulated Annealing. Estamos tratando con el concepto de felicidad en la función heurística (coste - X*felicidad).

- Observación: Al querer entregar los paquetes antes del plazo mínimo para tener a los clientes “felices”, provocará un aumento en el coste de entrega.
- Planteamiento: Generamos diversas soluciones aplicando esta nueva heurística con distinto “peso” de felicidad y observamos los resultados.
- Hipótesis: Cuanto más peso tenga la felicidad más aumentará el coste (H0).
- Método:
    - Elegiremos 10 semillas (1-10 en nuestro caso), una para cada réplica.
    - Experimentamos con los parámetros del experimento 1: 100 paquetes y una proporción del peso igual a 1.2.
    - Usaremos el algoritmo de Simulated Annealing, el operador mover y el generador solución inicial 2.
    - Función heurística 2: coste – X*felicidad (*nos dedicaremos a estudiar cuánto vale esa X*).

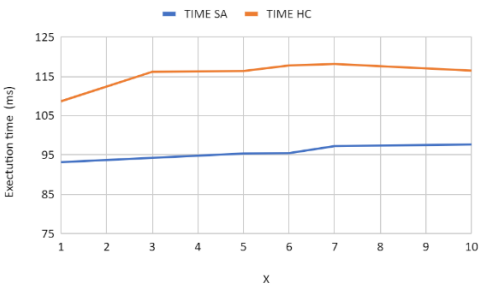
Para cada valor de X (*X es el factor de ponderación de la felicidad en la heurística*) obtendremos su respectivo coste y felicidad, se obtendrá haciendo la media de las diez semillas.

Estas son las medias obtenidas para diferente valor de X.

X	COST SA	HAPPINESS SA
1	686,13	15,9
3	751,00	46,5
5	815,34	62
6	836,38	65,7
7	854,57	67,8
10	878,87	69,6



En estas gráficas se puede observar los diferentes resultados obtenidos con las dos búsquedas locales. La interpretación de los gráficos es igual a la del experimento 6, así que nos dedicaremos a observar las diferencias entre estos dos algoritmos. Estos dos algoritmos obtienen resultados muy a la par, pero si observamos bien Simulated Annealing casi siempre está por encima, es decir, con este algoritmo acabamos teniendo una felicidad sutilmente mayor que Hill Climbing, con un coste muy similar. Esta diferencia es tan ligera que podemos concluir que estos dos algoritmos trabajan en las mismas tendencias.



Por lo que el tiempo respecta, podemos observar en la gráfica de la izquierda que Simulated Annealing consigue mejores tiempos que Hill Climbing. Pero la diferencia no es mucha ya que estamos hablando en este caso de ms. Los dos tiempos se mantiene constantes.

Tabla con los resultados obtenidos con nuestro valor escogido X = 6 para Simulated Annealing.

SEED	INIT. COST	COST	HAPPINESS
1	820,01	927	63
2	741,99	879,11	65
3	656,84	728,67	51
4	795,33	909,1	74
5	620,905	747,67	81
6	676,13	760,11	58

7	624,26	771,14	88
8	774,99	884,92	55
9	733,53	904,51	66
10	767,09	851,43	56
Average	721,11	836,38	65,7
Deviation	71,86	76,10	12

4.8. Efecto del coste de almacenamiento

La respuesta rápida y no del todo correcta, sería, si subimos el coste de almacenamiento, el coste de los estados finales incrementará, mientras que, si disminuye, el coste final disminuirá. Esta respuesta es precipitada, ya que si disponemos de un gran abanico de ofertas, y el precio de almacenar un paquete es mayor que el envio de una oferta donde no tenemos almacenar el paquete, nos puede interesar el hecho de enviar el paquete lo antes posible por tal de no gastarnos más dinero almacenándolo. Entonces, puede que si el precio de almacenar es muy elevado, prefiramos enviar un paquete lo antes posible, evitando así estos costes. Cabe destacar que solo haríamos estos movimientos si obtenemos un beneficio mayor. Si llegamos a este punto, el hecho de enviar antes los paquetes, lo veríamos reflejado en una mayor felicidad por parte de los clientes.

Por otra parte, si el coste de almacenamiento disminuye, tanto si trabajamos con la primera función heurística como con la segunda, trataremos de enviar los paquetes lo más tarde posible. En el caso de trabajar con la primera heurística, conseguiremos mayores beneficios ya que no tenemos en cuenta la felicidad de los clientes. Por tanto, si mantenemos los paquetes en el almacén el mayor tiempo posible, dado que el coste será menor que muchas ofertas, preferiremos enviarlos en las ofertas con envíos más tardíos.

Resumiendo, si aumenta el coste de almacenamiento y tenemos ofertas de envío que nos producen mayores beneficios que almacenar, optaremos por enviar los paquetes antes, si no disponemos de ofertas que cumplan esas condiciones, tendremos que almacenar los paquetes. Probablemente en ambas situaciones obtendremos costes finales mayores. Para el caso de tener costes de almacenamiento más bajos, preferiremos tener los paquetes el mayor tiempo posible en el almacén en vez de enviarlos antes, ya que probablemente disminuiremos el coste final total, así que enviaremos los paquetes lo más tarde posible.

5. Conclusión

Los experimentos llevados a cabo en esta práctica nos han servido para asentar de una manera práctica los conceptos vistos en teoría, tanto la toma de decisiones en la elección de las representaciones de los estados como en los conceptos propios de la búsqueda local.

Durante el proceso de implementación se nos ocurrieron diversas implementaciones alternativas de los diferentes elementos del problema, que debido a limitaciones en el tiempo disponible no pudimos llevar a cabo. Por ejemplo, la calidad de las soluciones iniciales ha sido relativamente alta para ambos generadores de solución inicial utilizados en nuestra práctica, creemos que hubiera sido interesante generar una solución inicial menos guiada hacia una solución eficiente y más aleatoria.

Sí decidimos implementar hasta cuatro operadores distintos, combinaciones de move y swap, que resultaron interesantes para entender el funcionamiento del algoritmo, por qué algunos operadores funcionaban mejor que otros, pero quedó claro de manera rápida que el operador move sería el operador escogido; swap no podía competir con la cantidad de estados posibles que podía generar move y el resto no generaban nunca mejores soluciones que move, pese a tener costes computacionales más altos.

De todos los resultados obtenidos, que han sido comentados conforme los representábamos, hemos de destacar la enorme ventaja que representa el uso de Simulated Annealing sobre Hill Climbing en este problema.

La elección aleatoria de movimiento de Simulated Annealing permite evitar la costosa generación exhaustiva de todos los descendientes de un estado en Hill Climbing, y este hecho ha sido expuesto por la evidencia experimental aquí presentada. En concreto, la estabilidad temporal de Simulated Annealing durante la escalabilidad del tamaño del problema lo hace un firme ganador, ya que la calidad de las soluciones obtenidas ha sido consistentemente muy parecida en los dos algoritmos.