Práctica 1 de Inteligencia Artificial [Grau - IA]

Estudio y optimización de los algoritmos de búsqueda para la empresa *Azamon*

1r cuatrimestre - Curso 2016-2017
Ismael Julià - Ana Mestre - Gorka Piñol

Índice.

Descripción del problema y justificaciones de su solución.

Descripción y justificación de la implementación del estado

El estado queda representado con los siguientes atributos:

- `paq`: objeto de la clase Paquetes que contiene todos los paquetes (objetos de la clase Paquete) que deben asignarse a alguna oferta para ser entregados.
- `trans`: objeto de la clase Transporte que contiene todas las ofertas de transporte (objetos de la clase Oferta) que las compañías tienen disponibles.
- `asignacion`: vector de enteros que relaciona cada paquete con la oferta que va a transportarlo. `asignacion[i]`, donde `i` es el índice del paquete en `paq`, toma por valor -1 si el paquete `i` no tiene ninguna oferta asignada y, si sí se le ha asignado una, el índice de ésta en `trans`.
- `capActual`: vector de reales que contiene el peso de todos los paquetes asignados a una oferta. `capActual[i]` toma por valor la suma del peso de todos los paquetes asignados a la oferta de índice `i` en `trans`.
- `precio:` real que contiene el coste de almacenamiento y transporte para la asignación en el estado actual.
- `felicidad`: entero que contiene la felicidad total observada por los clientes con la asignación actual, definida como la suma de las diferencias entre los días mínimos de llegada del paquete para su prioridad y el tiempo real que ha tardado en llegar, si esta diferencia es positiva.

Hemos elegido esta implementación porque cambiar de oferta un paquete se reduce a modificar el valor de la posición correspondiente del vector `asignacion`, que tiene un coste temporal muy reducido y por tanto no añadirá excesivo tiempo a la ejecución de la búsqueda local. Los atributos `capActual`, `precio` y `felicidad` permiten acceder a los datos que proporcionan sin tener que recalcularlos cada vez que los necesitemos, hecho que haría que la búsqueda tardara un tiempo prohibitivo.

Descripción y justificación de los operadores usados

Los operadores usados son:

- `moverPaquete(p, o)`: mueve el paquete `p` a la oferta `o` si la prioridad del paquete permite asignarlo a dicha oferta y si ésta no superará el peso máximo que puede transportar al añadir el paquete.
- `permutarPaquete(p1, p2)`: intercambia los paquetes `p1` y `p2` si sus respectivas prioridades permiten asignarlos a la oferta a la que pertenecía el otro paquete y si éstas no superarán el peso máximo que pueden transportar al cambiar el paquete que se les asigna.

Hemos escogido únicamente estos dos operadores porque con ellos somos capaces de recorrer todo el espacio de soluciones. En un inicio planteamos los operadores `añadirPaquete(p, o)` y `quitarPaquete(p)`, que, respectivamente, añadían un paquete `p` a la oferta `o` y quitaban el paquete `p` de la oferta a la que estuviera asignado, pero descartamos su uso dado que `quitarPaquete` generaba no-soluciones y sin él `añadirPaquete` perdía la razón de ser.

Descripción y justificación de las estrategias para hallar la solución inicial

Las dos estrategias implementadas son las siguientes:

- `generaSolInicial1`: tras ordenar los paquetes por prioridad ascendente (de 0 a 2), los asigna a ofertas que cumplan el requisito de prioridad en tres iteraciones (una para cada tipo), para asegurar que los paquetes de más prioridad tienen preferencia para ser asignados a una oferta que pueda entregarlo en el plazo indicado como mínimo; si es posible, los asignará a una oferta que los entregará antes.
- `generaSolInicial2`: tras ordenar los paquetes por prioridad descendente (de 2 a 0) y las ofertas por días también descendente (de 5 a 1), asigna secuencialmente los paquetes a las ofertas, teniendo en cuenta la prioridad que éstos tienen.

Puesto que `generaSolInicial1` coloca los paquetes de forma que estén en la oferta con menor tiempo de entrega disponible, las soluciones que genera tienen una felicidad mayor que cero, a cambio de tener un precio más elevado, ya que el coste es mayor cuanto más rápido es el transporte. `generaSolInicial2`, al ordenar las ofertas descendentemente, asigna los paquetes a ofertas más baratas, ya que el coste es menor cuanto más días tarda en entregarse; de esta forma, el precio resultante es menor que con la primera estrategia, a cambio de tener felicidad cero en la asignación.

Descripción y justificación de las funciones heurísticas

Las funciones heurísticas que se han implementado son:

- `HeuristicFunction1`: el valor que guía la función es el coste del almacenamiento y transporte que supone la asignación considerada.
- `HeuristicFuncion2`: Tenemos que darle un valor a la felicidad lo suficientemente grande como para que cambie el heurístico, pero sin que afecte demasiado en el coste, por lo que nosotros hemos decidido hacer una resta ponderada tal que el heurístico sea: `precio - 10*felicidad`.

Experimentos.

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: Puede haber conjuntos de operadores mejores que otros.
Planteamiento: Elegimos distintos conjuntos de operadores y observamos sus resultados.
Hipótesis: Todos los conjuntos de operadores son iguales (H0) o los hay que producen mejores resultados.
Método:

- Elegimos diez semillas aleatorias, una para cada réplica.
- Ejecutamos un experimento para cada semilla y conjunto de operadores (mover y permutar, mover).
- Experimentamos con problemas de 100 paquetes y proporción de peso paquetes/ofertas 1,2.
- Usamos el algoritmo de Hill Climbing.
- Medimos distintos parámetros para hacer la comparación.

Réplica	Solución		Tiempo de ejecución (ms)	
	Mover y permutar	Mover	Mover y permutar	Mover
1	791.17	791.62	371.95	88.44
2	858.63	860.15	132.65	57.76
3	1093.36	1093.61	124.37	56.88
4	987.69	989.44	87.63	30.15
5	987.78	989.28	78.24	46.15
6	987.84	989.33	134.68	30.15
7	776.55	777.73	106.41	32.01
8	951.11	952.72	311.77	45.88
9	978.21	980.06	89.80	33.08
10	1056.91	1057.55	138.97	46.40
Media (desv. típica)	946.9 (105.7)	948.1 (105.7)	157.6 (100.4)	46.69 (18.01)

Es obvio que, aunque el conjunto de operadores que sólo consta del operador “mover” da soluciones ligeramente peores que el conjunto formado por “mover” y “permutar”, en el tiempo de ejecución es mucho mejor (columnas tercera y cuarta de la tabla). Dado que tal ventaja en el tiempo de ejecución es más significativa que la pequeña desventaja en la solución, nos decantamos por usar el único operador “mover”.

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: Uno de los generadores de soluciones iniciales puede ser mejor que el otro.
Planteamiento: Generamos soluciones con ambos y comparamos sus resultados.
Hipótesis: Los dos generadores de soluciones son iguales (H0) o uno de ellos produce mejores resultados.
Método:

- Elegimos diez semillas aleatorias, una para cada réplica.
- Ejecutamos un experimento para cada semilla y generador de soluciones iniciales.
- Experimentamos con problemas de 100 paquetes y proporción de peso paquetes/ofertas 1,2.
- Usamos el algoritmo de Hill Climbing.
- Medimos distintos parámetros para hacer la comparación.

	Solución inicial		Solución final		Tiempo de ejecución (ms)	
	1	2	1	2	1	2
1	1211.24	1038.09	995.19	996.09	53.60	17.22
2	1337.87	1134.9	1092.49	1092.72	31.79	18.50
3	1174.65	999.57	964.49	966.93	29.02	10.37
4	1241.44	976.55	927.47	921.69	35.12	14.21
5	1146.91	997.94	942.88	943.47	30.74	16.62
6	1128.04	971.58	946.12	946.12	25.65	12.35
7	1215.90	1049.03	1009.43	1007.85	63.37	9.93
8	1399.79	1218.95	1183.65	1184.34	34.20	13.28
9	1045.06	847.67	831.93	832.97	25.33	10.33

10	1388.06	1155.56	1120.56	1120.88	27.96	7.23
Media (desv. típ.)	1229 (115.8)	1039 (107.3)	1001 (104.3)	1001 (104.6)	35.68 (12.66)	13.00 (3.65)

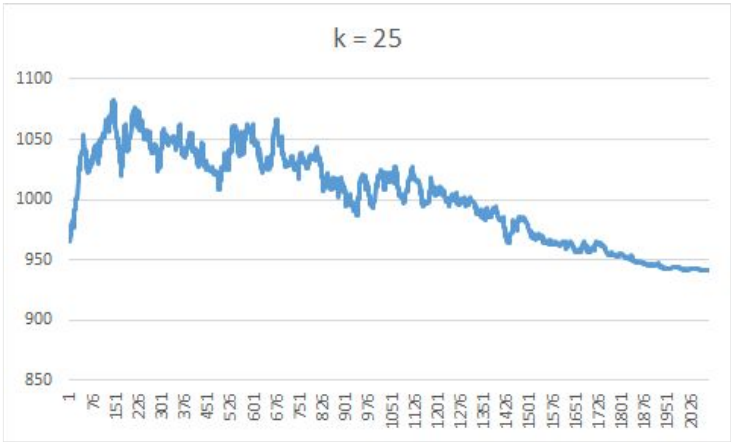
El segundo generador de soluciones iniciales, aunque nos proporciona una asignación con un precio inicial menor que el primero (corriendo así el riesgo de caer en un máximo local), como se ve en la primera y segunda columnas, nos proporciona soluciones muy similares a las del primer generador, con la ventaja de tardar menos tiempo en ejecutar la generación y búsqueda (columnas cinco y seis de la tabla). Por esto, nos decantamos por usar la segunda función de generación de soluciones.

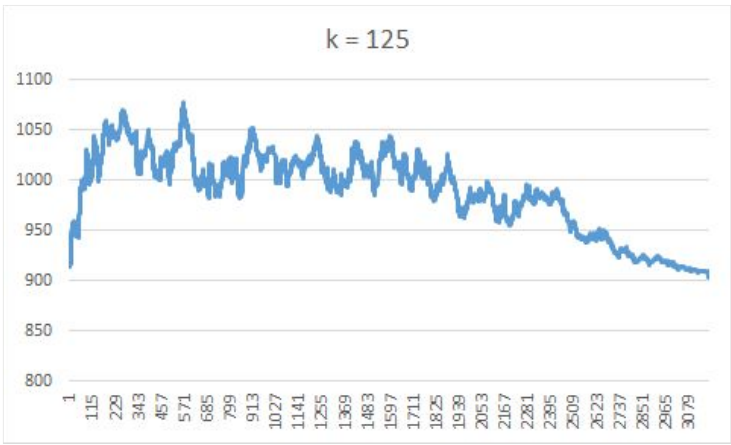
3. Determinar los parámetros que dan mejor resultado para el Simulated Annealing con el mismo escenario, usando la misma función heurística y los operadores y la estrategia de generación de la solución inicial escogidos en los experimentos anteriores.

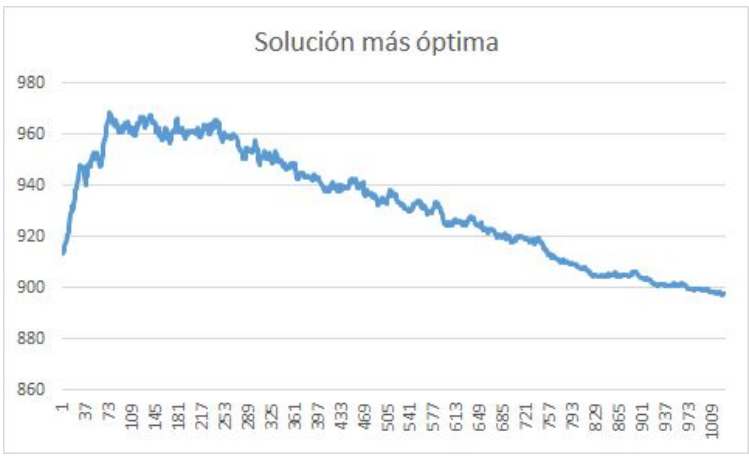
Observación: Tiene que haber una combinación de los parámetros de la función que calcula el Simulated Annealing que de mejores resultados que otra combinación.
Planteamiento: Generamos soluciones con diversos parámetros para ir poco a poco acotando sus valores.
Hipótesis: Podemos encontrar una combinación con la que el Simulated Annealing nos muestre los mismos resultados que el Hill Climbing (H0) o mejores.
Método:
Para poder determinar los valores que tendríamos que asignarles a cada parámetro del Simulated Annealing hemos hecho pruebas con diferentes números.
El proceso es el siguiente, estableciendo los valores K =1, 5, 10, 25, 125, Lambda = 1.0, 0.1, 0.01, 0.001, 0.0001, escogiendo un número lo bastante grande de iteraciones y también probando diversos valores para los pasos que hará cada iteración, se hacen cálculos del Simulated Annealing para 10 semillas diferentes. Luego se calcula la media para cada K, Lambda. Con estas medias se realizan varias gráficas, se comparan y se escogen los resultados que sean mejores, en nuestro caso, aquellos resultados que proporcionen un mejor precio.
Para hacer las pruebas hemos usado las siguientes funciones, que hemos supuesto a partir de los dos experimentos anteriores que serían las que nos proporcionarían un mejor resultado:

- El segundo generador de soluciones iniciales.
- La función heurística 1: función que solo tiene en cuenta el coste.
- La successor function 2: función que tiene como único operador el de mover.

De todas las gráficas que se han tenido que representar para estudiar mejor los resultados, éstas son las más relevantes para explicar el porqué de los parámetros escogidos. Tomando Lambda = 0,001 y 10.000 como el número de iteraciones, estos son los resultados para K = 25, K = 125 y K = 5 respectivamente.



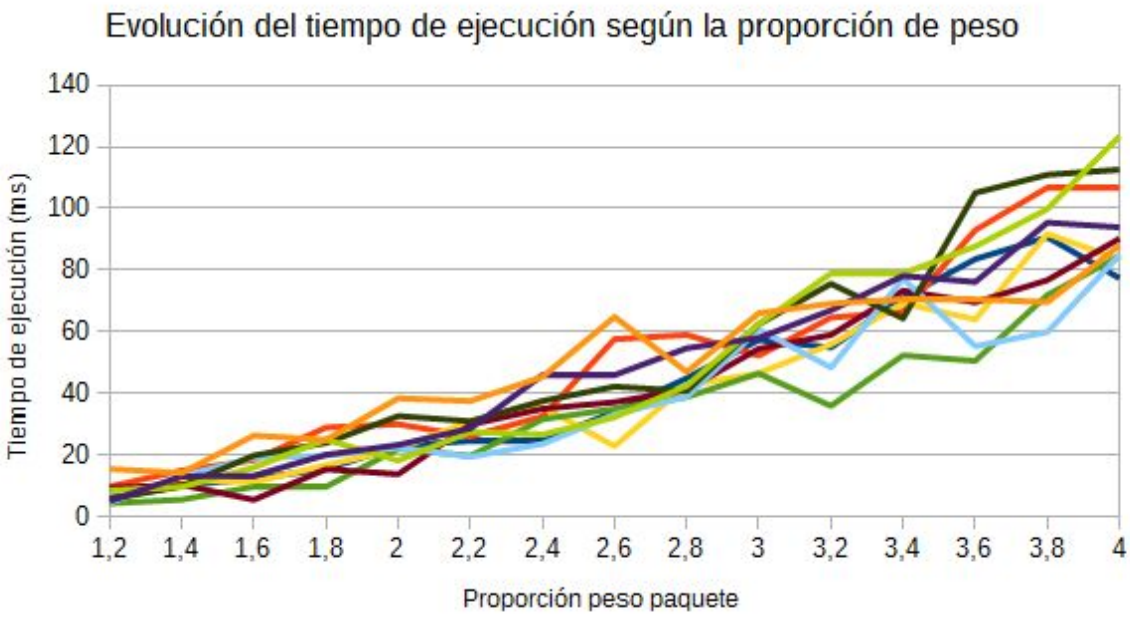




Se puede observar a partir de las gráficas incluidas, que variando la k hemos encontrado el precio más óptimo (k = 5), no nos hemos alejado mucho del punto de probabilidad 0 de aceptación de estados y hemos variado el número de iteraciones hasta 10.000 y mantenido lambda a 0.001, consiguiendo aceptar aún más estados peores y alargando la convergencia, pero acortando la diferencia entre estados sucesivos, acercándonos más al óptimo del precio.

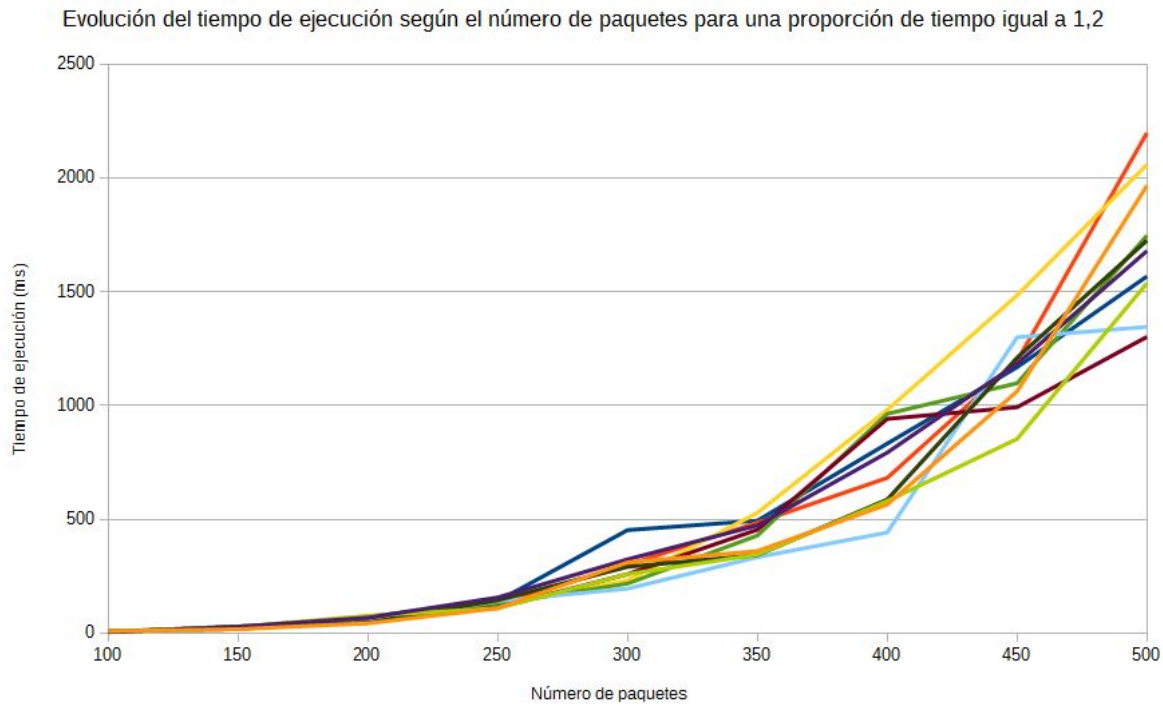
4. Dado el escenario de los apartados anteriores, estudiando cómo evoluciona el tiempo de ejecución para hallar la solución en función del número paquetes y la proporción de peso transportable.
Usad el algoritmo de Hill Climbing y la misma heurística.

- **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: Al aumentar la proporción de peso transportable, aumentará también el tiempo de ejecución del algoritmo.
  Planteamiento: Generamos diversas soluciones para cada proporción y observaremos los resultados.
  Hipótesis: El tiempo de ejecución aumentará con la proporción del peso (H0) o se mantendrá igual.
  Método:
  Asignamos un valor aleatorio a la semilla y ejecutamos el código con dicha semilla para las diferentes proporciones, desde 1.2 a 4 (aumentando 0.2 en cada iteración). Este proceso se repite diez veces para obtener varios resultados y así compararlos entre ellos.



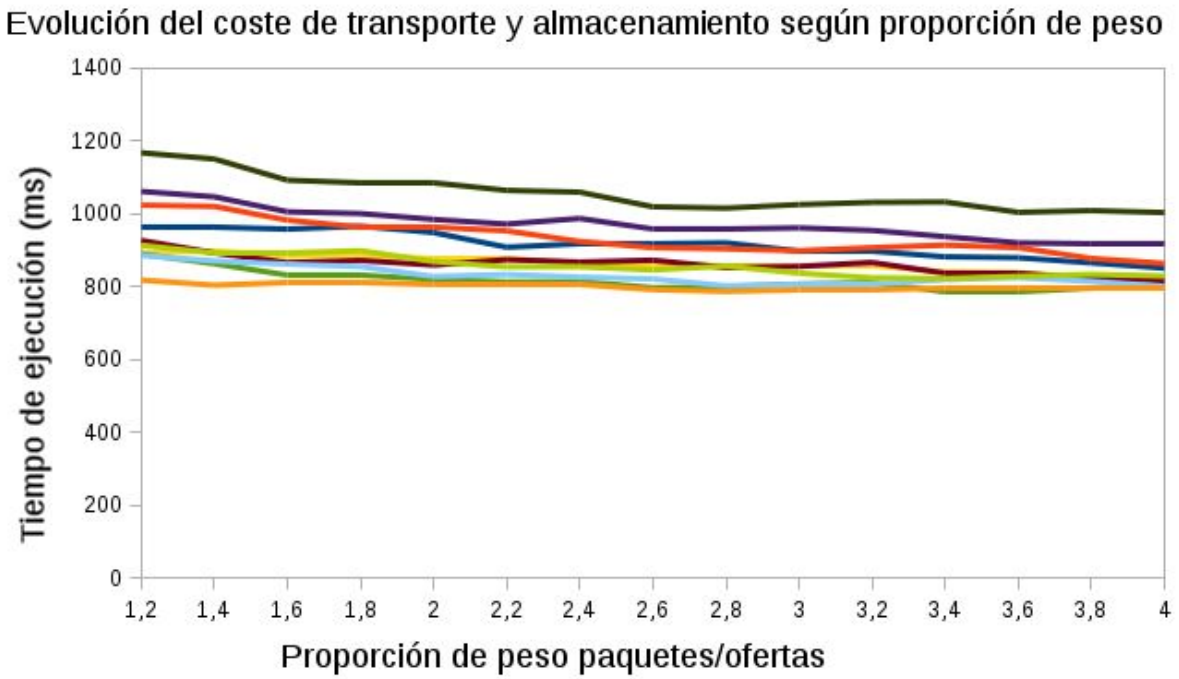
Como se puede observar en la gráfica, conforme la proporción de peso entre paquetes y ofertas aumenta, el tiempo de ejecución del Hill Climbing también; esto se debe a que al aumentar dicha proporción el número de ofertas disponibles es mayor, y por tanto la función generadora de estados sucesores tiene la posibilidad de mover cada uno de los paquetes a más ofertas que con una proporción menor, aumentando así el tiempo de ejecución.

- **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: Al aumentar el número de paquetes que serán transportados, aumentará también el tiempo de ejecución del algoritmo.
  Planteamiento: Generamos diversas soluciones para cada número de paquetes y observamos los resultados.
  Hipótesis: En aumentar el número de paquetes, aumentará también el tiempo de ejecución del algoritmo.
  Método:
  Asignamos un valor aleatorio a la semilla y ejecutamos el código con dicha semilla para los diferentes números de paquetes, desde 100 a 500 (aumentando de 50 en 50). Este proceso se repite diez veces para obtener varios resultados y así compararlos entre ellos.



Como se puede observar en la gráfica, conforme el número de paquetes aumenta también aumenta el tiempo de ejecución; esto se debe a que cuantos más paquetes tenemos más posibles estados sucesores tenemos (mover cada paquete a cada oferta, recordando que cuantos más paquetes tenemos más ofertas, también), y por tanto el tiempo de ejecución aumenta.

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?



Como se puede observar en la gráfica, aunque al principio es beneficioso, el aumento de la proporción acaba por no influir en el coste significativamente, estabilizándose en un valor con fluctuaciones mínimas. Por tanto, dado que el aumento del número de ofertas implica que el tiempo de ejecución es mayor, no merece la pena aumentar la proporción para mejorar los resultados de la búsqueda.

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 cómo hacéis la exploración de su ponderación en la función. No hace falta que la exploración sea exhaustiva, solo hace falta que veáis tendencias.

Observación: Al querer entregar los paquetes antes del plazo mínimo, aumentará el coste de la entrega.
Planteamiento: Generamos diversas soluciones diferentes para analizar los resultados.

Hipótesis: El precio aumentará aumentará (H0) o se mantendrá igual.
Método:
Disponemos de los siguientes recursos:

- Algoritmo de Hill Climbing.
- Generador de soluciones iniciales 2.
- Función heurística 2: que tiene en cuenta el coste del transporte y la felicidad.
- Successor function 2: solo usa el operador de mover.

Cogemos diez semillas aleatorias y teniendo en cuenta las funciones mencionadas, imprimimos por pantalla los resultados: el precio inicial, el final, el tiempo total de ejecución y la felicidad obtenida.
Estos son los resultados obtenidos:

Iteración	Precio inicial	Precio final	Felicidad
1	1101.595	1102.575	3
2	970.09	971.78	3
3	916.085	917.290	3
4	1046.6049	1047.699	3
5	883.399	884.359	3
6	839.059	840.3049	3
7	963.53	966.42	3
8	1035.0649	1036.239	3
9	1031.23	1032.3	3
10	1095.715	1096.59	3
Media (desv. típica)	988.2 (88.97)	989.6 (88.84)	3

Observando los resultados vemos que no aumenta mucho la felicidad a causa de que nuestro generador de soluciones iniciales cae pronto en un máximo local. Aún así se consigue algún ligero grado de felicidad y aumenta muy poco el coste por intentar redistribuir los paquetes de manera que lleguen antes y hagan más felices a los clientes.

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

Comparar la combinación de operadores:

Réplica del primer experimento. Usando Simulated Annealing comparamos los operadores “mover y permutar” con el operador “mover”. Los criterios que tenemos en cuenta son: 100 como número de paquetes, 1.2 como proporción de peso transportable, la generación de soluciones iniciales 1 y la función heurística 1.

Réplica	Solución		Tiempo de ejecución (ms)	
	Mover y permutar	Mover	Mover y permutar	Mover
1	992.08	993.21	9858	3506
2	985.22	986.45	9751	3799
3	915.85	917.98	8649	3424
4	888.61	889.10	9990	4541
5	960.92	962.16	10061	4234
6	1020.68	1023.64	11024	3727
7	990.96	994.27	10172	3555
8	1055.37	1056.48	8642	3010
9	873.79	874.71	11266	4154
10	920.41	923.27	10962	4356
Media (desv. típica)	960.4 (59.11)	962.1 (59.36)	10038 (903.6)	3831 (480.7)

El proceso aplicado ha sido exactamente el mismo que para el experimento 1. En este caso en vez de usar la búsqueda por Hill Climbing aplicamos Simulated Annealing con los valores que hemos encontrado en el experimento 3, Lambda = 0.001, k = 25, Iteraciones = 10.000 y pasos = 10.

Solo con recoger los resultados hemos confirmado lo que veíamos desde que buscábamos los valores adecuados para el Simulated Annealing, este algoritmo tarda mucho más que el Hill Climbing. Se ve claramente porque en la tabla los resultados ya son de la magnitud de segundos.

En este nuevo caso, también obtenemos una media en cuanto a coste muy similar para los dos conjuntos de operadores. Pero en cuanto a tiempo, cuando el algoritmo mueve y permuta tarda algo menos que el triple de tiempo que cuando solo se dedica a mover paquetes.

El operador que tarda menos, de nuevo, consigue un coste ligeramente peor pero podemos asumir esa diferencia teniendo en cuenta que el tiempo de ejecución es casi 3 veces más rápida.

Se puede concluir que la relación de tiempo entre los dos conjuntos de operadores es muy similar al Hill Climbing pero tardando realmente mucho más.

Comparar el generador de soluciones iniciales:

Nueva versión del segundo experimento, una comparativa entre las dos generaciones de soluciones iniciales. Esta vez compararemos que solución inicial es mejor para el algoritmo de Simulated Annealing.
La metodología ha sido exactamente la misma que en el caso ya calculado para el Hill Climbing pero ahora solo con un cambio de algoritmo, dándole los valores obtenidos en el experimento 3.

	Solución inicial		Solución final		Tiempo de ejecución (ms)	
	1	2	1	2	1	2
1	1294.47	1071.57	1042.09	1040.22	3051	3009
2	1158.19	944.60	893.65	892.66	3094	3083
3	1015.07	825.70	777.35	779.43	3552	3689
4	1219.59	1054.06	1000.49	1000.08	3025	3094
5	1166.87	925.35	895.67	897.13	4161	4128
6	1210.90	1010.07	980.78	980.94	3506	3411
7	1064.06	943.10	887.40	887.68	2732	2873
8	1282.39	1136.09	1092.83	1091.98	3987	3932
9	1368.37	1089.49	1044.33	1045.02	2664	2716
10	1189.59	1023.36	967.93	964.08	3331	3216
Media (desv. típ.)	1197 (105.5)	1002 (92.75)	958.3 (94.73)	957.9 (93.94)	3310 (497.4)	3315 (465.6)

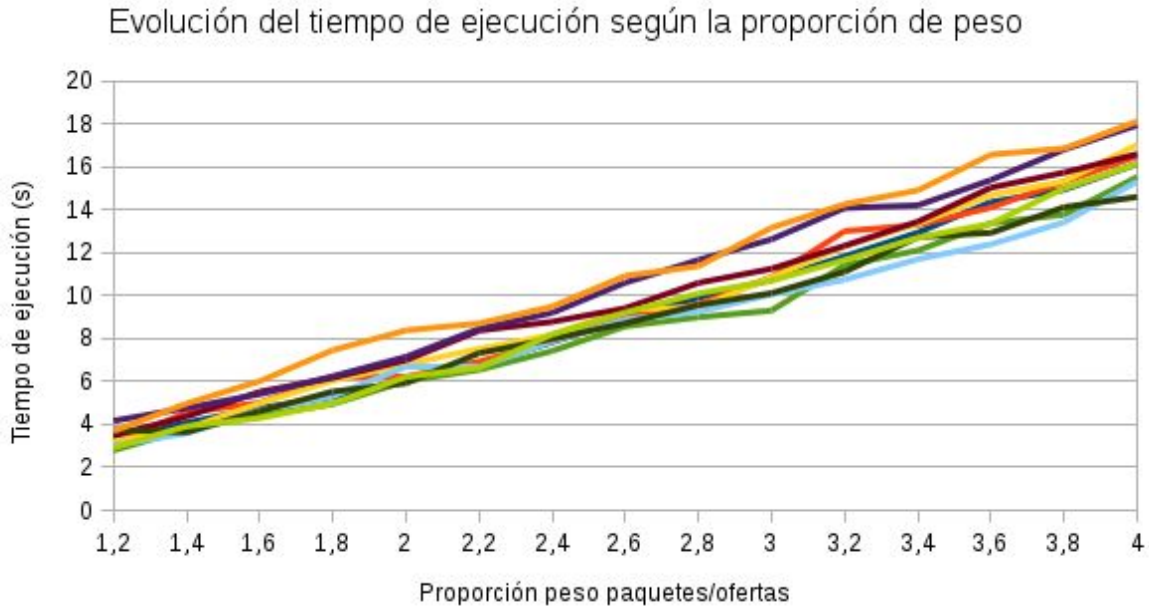
De nuevo, lo primero a destacar de este experimento es el tiempo que tarda el algoritmo de Simulated Annealing. Tanto para la primera función generadora de soluciones iniciales como para la segunda, tarda alrededor de 3 segundos en ejecutarse. En cambio el algoritmo de Hill Climbing tardaba aproximadamente 35 ms para el primer generador y 13 para el segundo.
Con ambos generadores de soluciones se consigue prácticamente el mismo resultado, siendo el segundo el que consigue un coste más económico por una diferencia de tan solo 40 céntimos, pero también tarda 5 ms más. La diferencia es menospreciable, podríamos escoger cualquiera de las dos soluciones iniciales como válidas.
Si comparamos el resultado final con el obtenido con el Hill Climbing:

Solución final 1: 1001 €. - Solución inicial 1: 1229 €. - Tiempo 1: 35.68 ms.
Solución final 2: 1001 €. - Solución inicial 2: 1039€. - Tiempo 2: 13 ms.

Vemos que con este nuevo algoritmo y con los valores que le hemos asignado, se consiguen en media soluciones mejores, una diferencia de 42.70 € para el primer generador y 43.10 € para el segundo.

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.

Para este nuevo experimento hemos seleccionado la función heurística 1, el primer generador de soluciones iniciales y la successor function 2, ya que tarda bastante menos.



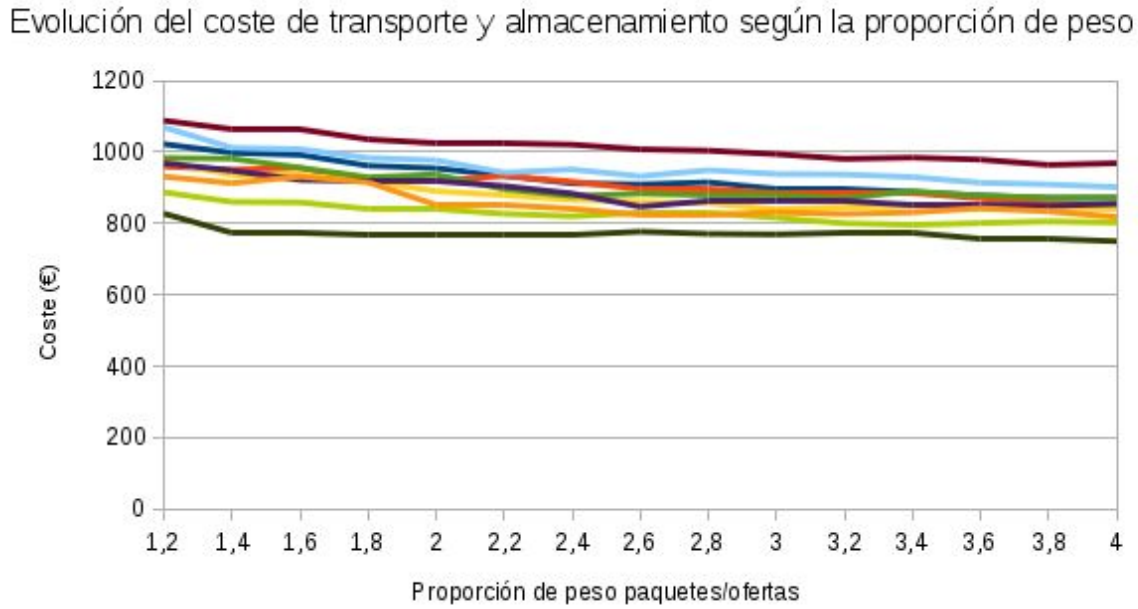
Como era de esperar, ocurre lo mismo que con el Hill Climbing, a medida que aumenta la proporción de peso transportable, aumenta rápidamente el tiempo de ejecución, pero tardando esta vez mucho más, puesto que este algoritmo es más lento.

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.

Aplicando la misma metodología que hasta ahora, generamos soluciones a partir de semillas con valor aleatorio y observamos los resultados.

Estudiando la gráfica se puede observar que aumenta de forma inmediata el tiempo de ejecución con el aumento del número de paquetes, lo mismo ocurre con el Hill Climbing.
En este ejemplo es donde probablemente queda mejor plasmada la gran diferencia de tiempo de ejecución del algoritmo de Simulated Annealing.

Análisis de los resultados del experimento en el que se aumenta la proporción de las ofertas.



Como se puede observar, tenemos un caso similar al Hill Climbing, en que el aumento de la proporción de peso de los paquetes/ofertas no hace que varíe mucho el coste de la solución. Se mantiene más o menos constante en todo momento. Además se mantiene en un intervalo de precios muy similares. Es decir, no vale realmente la pena aumentar la proporción de peso de los paquetes/ofertas si la intención es reducir costes, ni para Hill

Climbing ni para Simulated Annealing, puesto que el resultado que se va a obtener es prácticamente el mismo.

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?

> 0.25€ (kg/día): no nos es tan rentable almacenar los paquetes en el almacén esperando que la empresa de transporte (oferta) nos lo recoja, por lo tanto daremos preferencia a ofertas que entregan la paquetería con el menor tiempo para ahorrarnos costes innecesarios, que pueden superar incluso a los de una oferta que previamente podía parecer cara. Gracias a esto el índice de felicidad de los clientes podría ser superior, ya que una compra que llega antes es un buen motivo para volver a comprar en el mismo lugar. Todo esto depende obviamente del incremento, variando así sustancialmente el resultado de la solución.

< 0.25€ (kg/día): en este caso no es tan rentable contratar ofertas con pocos días de plazo de entrega, siempre intentaremos elegir el máximo del plazo de entrega para ahorrar costes, a no ser que haciéndolo de la otra forma las ganancias/pérdidas sean aproximadamente las mismas. Por lo tanto la solución se basaría más en encontrar un buen conjunto de ofertas donde asignar los paquetes en el mismo plazo o quizás en algunos casos un poco antes, ya que nos interesa mantener la paquetería en el almacén para ahorrar costes. Como en el caso anterior depende de la variación, y será esta la que marque cuánto cambia la solución.