Informe de la Práctica 1

Paula Esteve Sabater, Claudia Gallego Torralbo, Verónica Oñate Villagrasa

11 de Noviembre de 2024

Universidad Politécnica de Cataluña

Grado Inteligencia Artificial

Algoritmos Básicos de la Inteligencia Artificial

Contenidos

Introducción

Descripción del problema

El problema planteado aborda la optimización logística de una empresa ficticia llamada Ázamon, que necesita asignar de forma eficiente sus paquetes a las ofertas de transporte disponibles diariamente, proporcionadas por diversas empresas de paquetería. El objetivo principal es minimizar los costos asociados al transporte y al almacenamiento de los paquetes, así como maximizar la satisfacción de los clientes, medida en términos de "felicidad" relacionada con la rapidez en la entrega.

En este contexto, cada paquete tiene un peso específico y una prioridad de entrega, cada una con su respectivo costo.

Las ofertas de transporte, por su parte, especifican el peso máximo que pueden transportar, el costo por kilogramo transportado y el tiempo en días en que el paquete será entregado.

Además, un aspecto clave es el costo adicional por almacenamiento, que se aplica a aquellos paquetes que no pueden ser recogidos inmediatamente, esto genera un costo de almacenamiento de 0,25 euros por kilogramo y día. Un factor adicional a considerar es la "felicidad" del cliente, que se incrementa cuando el paquete llega antes del plazo indicado por la prioridad elegida por el usuario. La felicidad se cuantifica en función de los días de adelanto respecto a la fecha máxima de entrega establecida por el cliente, incentivando así la entrega anticipada como un valor añadido.

La empresa debe, por tanto, asignar todos los paquetes del día a alguna oferta, asegurándose de no exceder la capacidad máxima de las ofertas, y tratando de cumplir los plazos de entrega para cada prioridad. El desafío consiste en encontrar una asignación que logre equilibrar estos factores, optimizando los costos y maximizando la satisfacción de los clientes.

Descripción de los elementos del estado del problema

Cada componente representa una pieza clave en la estructura de datos necesaria para la implementación y resolución del problema:

1. **Paquete**: Este elemento representa la información específica de cada paquete que Ázamon necesita enviar. Cada paquete tiene dos atributos principales: su peso, expresado en múltiplos de 0,5 kg, hasta un máximo de 10 kg; y su prioridad de entrega, que determina el tiempo límite de entrega esperado por el cliente. Existen tres niveles de prioridad:
    - 0 para entrega al día siguiente,
    - 1 para entrega en un plazo de 2 a 3 días,
    - 2 para entrega en un plazo de 4 a 5 días.
2. **Oferta**: Este elemento encapsula la información de una oferta de transporte de las empresas de paquetería. Cada oferta tiene tres atributos principales:
    - **Peso máximo** que se puede transportar, con valores entre 5 y 50 kg en intervalos de 5 kg,
    - **Precio por kilogramo** transportado, que es específico de cada oferta y puede variar diariamente,
    - **Número de días de entrega**, que indica el tiempo que tardará en entregarse el paquete una vez recogido, con valores entre 1 y 5 días.
3. **Operadores activos**: Los operadores son las acciones que se pueden aplicar sobre el estado para explorar nuevas configuraciones en el espacio de búsqueda y obtener soluciones potenciales. En este caso, los operadores principales son:
    - **Asignar paquete a una oferta**: Permite asignar un paquete a una oferta específica de transporte, siempre que se cumplan las restricciones de peso y tiempo de entrega.
    - **Insertar un paquete:** Asigna un paquete a una oferta cualquiera
    - **Retirar paquete de una oferta**: Elimina un paquete previamente asignado, dejándolo disponible para una posible reasignación en otra oferta.
    - **Intercambiar paquetes entre ofertas**: Este operador intercambia paquetes entre dos ofertas, permitiendo explorar combinaciones alternativas y optimizando el espacio de búsqueda en función de los costos y la satisfacción del cliente.
4. **Métodos para calcular el coste**: Para evaluar cada solución, se calculan diferentes costos asociados a la asignación de paquetes:
    - **Costo de transporte**: Suma del costo de envío calculado por cada oferta, en función del peso de los paquetes asignados y el precio por kilogramo de cada oferta.
    - **Costo de almacenamiento**: Calculado como el producto entre el peso del paquete, el costo de almacenamiento diario (0,25 euros por kg), y el número de días adicionales en que el paquete debe almacenarse si no se recoge de inmediato.

- **Felicidad del cliente**: Un valor positivo que se incrementa por cada día que el paquete llega antes de la fecha máxima indicada en la prioridad del cliente. Esto permite equilibrar el costo y la satisfacción en la función heurística.

5. **Métodos para aplicar los operadores**: Para manipular y ajustar la asignación de paquetes mediante los operadores activos, los métodos deben:
   - **Validar restricciones**: Antes de aplicar un operador, verificar que no se superen las restricciones de peso de la oferta y que se cumplan los tiempos de entrega según la prioridad.
   - **Asignar o reasignar paquetes**: Implementar la asignación directa de paquetes a una oferta específica o cambiar un paquete de una oferta a otra.
   - **Actualizar costos**: Tras cada operación, recalcular los costos de transporte y almacenamiento, así como el nivel de felicidad, para mantener la evaluación de la solución actualizada.
   - **Generar sucesores**: En el contexto de algoritmos de búsqueda como Hill Climbing o Simulated Annealing, se debe implementar la generación de estados sucesores al aplicar operadores de forma aleatoria o sistemática, permitiendo la exploración del espacio de búsqueda.

Por qué es un problema de búsqueda local

Este problema se podría categorizar como un problema de búsqueda local, ya que requiere la exploración de un conjunto de soluciones donde cada solución se relaciona con una asignación específica de paquetes a ofertas de transporte, buscando minimizar los costos y mejorar la satisfacción del cliente, considerando las restricciones de peso y tiempos de entrega. En este contexto, cada estado implica una configuración de asignación, y los operadores permiten el cambio hacia "vecinos" de ese estado, es decir, soluciones similares con ligeras modificaciones (como redistribuir o intercambiar paquetes entre ofertas). A diferencia de otros tipos de problemas de búsqueda, no existe un único estado final que se deba lograr; en cambio, se busca una solución que cumpla con criterios de optimización. La búsqueda local es apropiada en este caso, ya que permite la mejora continua de la calidad de la solución, sin garantizar una óptima global, pero avanzando hacia una solución de alta calidad mediante estrategias heurísticas que evalúan cada estado teniendo en cuenta costos y satisfacción del cliente. Este enfoque es ideal para algoritmos de búsqueda local como Hill Climbing o Simulated Annealing, que permiten la exploración de soluciones de manera local, avanzando hacia configuraciones más óptimas sin la necesidad de revisar todas las combinaciones posibles dentro del espacio de búsqueda.

Preguntas iniciales

¿Qué elementos intervienen en el problema?

Paquetes: Cada paquete tiene un peso y una prioridad de entrega, que determina el tiempo máximo en el que debe ser entregado.
Ofertas de transporte: Cada oferta tiene una capacidad de peso, un precio por kilogramo y un tiempo de entrega.
Costos asociados: Incluyen el costo de transporte (en función del peso y el precio por kilogramo de cada oferta) y el costo de almacenamiento (cuando los paquetes deben esperar antes de ser recogidos).
Felicidad del cliente: Se mide en función de los días de adelanto con que un paquete llega respecto al tiempo máximo deseado, como incentivo para mejorar la satisfacción del cliente.

¿Cuál es el espacio de búsqueda?

El espacio de búsqueda es el conjunto de todas las posibles asignaciones de paquetes a las ofertas de transporte disponibles. Cada combinación representa un estado único en el espacio de soluciones.

¿Qué tamaño tiene el espacio de búsqueda?

El tamaño del espacio de búsqueda es muy grande, ya que depende de la cantidad de paquetes y las opciones de ofertas disponibles para cada paquete. Si, por ejemplo, hay *n* paquetes y *m* ofertas posibles, el número de combinaciones posibles sería aproximadamente m^n, lo que implica un crecimiento exponencial a medida que aumentan los paquetes y las ofertas, haciendo que el espacio de búsqueda sea difícil de explorar completamente.

¿Qué es un estado inicial?

Un estado inicial es una asignación completa inicial de todos los paquetes a alguna oferta de transporte, respetando las restricciones básicas de peso y tiempo de entrega de cada oferta. Este estado puede ser generado aleatoriamente o siguiendo alguna estrategia sencilla que cumpla con los requisitos mínimos de asignación.

¿Qué condiciones cumple un estado final?

En este problema, un estado final no es un objetivo único, sino una solución que optimiza los criterios de costos y felicidad del cliente, dentro de las restricciones de peso y tiempo de entrega. Un estado final "aceptable" cumple con las restricciones de capacidad de las ofertas, respeta los tiempos de entrega según la prioridad de los paquetes, y logra una

asignación con los costos mínimos posibles y la máxima felicidad del cliente, aunque no necesariamente sea un óptimo global.

¿Qué operadores permiten modificar los estados?

Los operadores que permiten modificar los estados incluyen:

- **AssignPackage**
- **InsertPackage**
- **RemovePackage**
- **SwapAssignments**

¿Qué factor de ramificación tienen los operadores de cambio de estado?

AssignPackage:

- Cada uno de los n paquetes puede asignarse a cualquiera de las m ofertas, cumpliendo con las restricciones.
- **Factor de ramificación**: $O(m \times n)$

InsertPackage:

- Solo implica la acción de asignar uno de los n paquetes.
- **Factor de ramificación**: $O(n)$

RemovePackage:

- Solo implica la acción de retirar uno de los n paquetes, sin reasignarlo.
- **Factor de ramificación**: $O(n)$

SwapAssignments:

- Se intercambian dos de los n paquetes
- **Factor de ramificación**: $O(n^2)$

Estructura e implementación del programa

En este apartado se describen y justifican las implementaciones utilizadas en cada uno de los archivos y clases del programa.

Estructura general

- *main.py*

El archivo main.py funciona como un menú para ejecutar diferentes experimentos del proyecto. Importa el módulo experiments, que contiene funciones específicas para cada experimento, y utiliza la entrada del usuario para seleccionar el experimento deseado. Dependiendo del número de experimento ingresado, llama a la función correspondiente de experiments

- *experiments.py*

El archivo experiments.py gestiona la ejecución de los múltiples experimentos de optimización del modelo de asignación de paquetes a ofertas. Define funciones para generar estados iniciales y ejecuta diferentes configuraciones con algoritmos como hill_climbing y simulated_annealing, evaluando métricas como el costo, felicidad del cliente y tiempo de ejecución. Los experimentos exploran el impacto de parámetros como la proporción de peso transportable, el número de paquetes y el parámetro alpha en los resultados, generando gráficos y archivos CSV para facilitar la comparación y el análisis de estos efectos en el rendimiento del modelo.

- *azamon_operators.py*

Define los operadores utilizados para manipular las asignaciones de paquetes a ofertas. Estos incluyen las clases AssignPackage, SwapAssignments, RemovePackage e InsertPackage, cada una representando una acción específica que puede realizarse en el problema. Todos los operadores heredan de la clase base AzamonOperator, proporcionando una estructura coherente para aplicar acciones sobre el problema de asignación.

- *azamon_problem_opt.py*

Este archivo define la clase principal del problema de optimización, llamada AzamonProblem. En esta clase se configura el espacio de búsqueda de posibles estados y se define el conjunto de acciones válidas que pueden aplicarse a cada estado, junto con las reglas de evaluación de costos y felicidad. Incluye métodos para calcular diferentes heurísticas, teniendo en cuenta el coste de transporte y almacenamiento y teniendo en cuenta los costes y la felicidad de los clientes, que ayudan a guiar la búsqueda hacia configuraciones más óptimas. Además, AzamonProblem verifica si un estado cumple todas las restricciones del problema (por ejemplo, los límites de peso y días de entrega), lo cual indica si el estado actual es una solución válida. Dentro de la clase se definen

- *azamon_problem_parameters.py*

Este archivo centraliza los parámetros del problema, como la capacidad máxima de peso de las ofertas, los límites de días de entrega, el precio por kilogramo, y las restricciones de entrega para cada paquete. Estos parámetros configuran las características y restricciones específicas de los paquetes y ofertas, asegurando que las asignaciones sean viables y optimizadas. Tener estos valores en un archivo separado permite modificar las especificaciones del problema sin afectar la lógica del resto del proyecto.

- *azamon_state_opt.py*

Este archivo define la clase StateRepresentation, que modela el estado actual de las asignaciones de paquetes a ofertas. La clase incluye métodos para aplicar operadores sobre el estado, permitiendo ejecutar acciones como asignar, intercambiar, eliminar o insertar paquetes. También proporciona métodos para calcular el costo y la felicidad asociados a la configuración actual, lo cual facilita la evaluación de cada estado. Además, StateRepresentation maneja atributos internos que reflejan el estado de las asignaciones, tales como los paquetes que están sin asignar, lo cual ayuda en la evaluación de restricciones.

- **apply_action**: Aplica un operador específico (asignación, intercambio, eliminación, inserción).
- **heuristic_cost** y **heuristic_happiness**: Evalúan el coste logístico y la felicidad del cliente, respectivamente.
- **heuristic_cost_happy**: Evalúa el coste logístico y la felicidad del cliente en combinación, según la importancia, que se le da con la ponderación, de cada uno.

Estado

En este proyecto, la implementación del estado facilita la asignación de paquetes a las ofertas, permitiendo evaluar de inmediato el impacto de cada modificación en las restricciones y los objetivos de optimización. Este estado centra los factores clave del problema: los límites de peso de cada oferta, los plazos de entrega según la prioridad, los costes de transporte y almacenamiento, y el nivel de satisfacción del cliente. Gracias a esto, se pueden aplicar operadores como asignar, intercambiar, eliminar o insertar paquetes en las ofertas de manera ágil, manteniendo siempre actualizadas las restricciones.

El estado también incorpora funciones heurísticas para calcular el coste (heuristic_cost) y la satisfacción (heuristic_happiness), permitiendo evaluar cada situación en tiempo real y guiar la búsqueda hacia soluciones óptimas. Para garantizar el cumplimiento de las reglas, el cálculo del coste añade penalizaciones cuando se infringen restricciones (como exceder el peso o retrasar la entrega), favoreciendo los estados viables. En conjunto, esta estructura proporciona una gestión flexible y efectiva de las asignaciones, buscando un equilibrio entre eficiencia operativa y satisfacción del cliente.

Operadores

- **AssignPackage**: Permite asignar un paquete a una oferta específica, o modificar su asignación actual si ya tiene una. Esto se realiza siempre y cuando se cumplan ciertas restricciones, como la capacidad de peso de la oferta y el límite de tiempo de entrega según la prioridad del paquete. En su implementación, primero se calcula el peso total de los paquetes ya asignados a cada oferta para asegurarse de que no se supere la capacidad permitida. Luego, para cada paquete, se evalúa si puede ser asignado a una nueva oferta. Este cambio solo se realiza si la oferta tiene suficiente capacidad para el peso adicional del paquete y si puede cumplir con el tiempo de entrega máximo permitido. Si cumple con estas condiciones, se crea una acción AssignPackage que contiene la asignación del paquete a la nueva oferta, y la acción se agrega a la lista de posibles cambios. El propósito principal de AssignPackage es ofrecer flexibilidad en la ubicación de los paquetes, optimizando su distribución para mejorar la capacidad de las ofertas y respetar las restricciones de tiempo de entrega.

- **InsertPackage**: Permite insertar un paquete que actualmente no tiene asignación en una oferta específica. Este operador es el complemento de RemovePackage y se utiliza para reincorporar paquetes en ofertas cuando hay capacidad suficiente para asignarlos. La implementación revisa los paquetes en la lista de paquetes faltantes y evalúa si pueden ser asignados a alguna oferta que cumpla con la capacidad de peso y el tiempo de entrega según la prioridad del paquete. Si se encuentra una oferta que cumple con estas condiciones, se crea una acción InsertPackage que asigna el paquete a esa oferta. Este operador es esencial para restablecer la asignación de paquetes previamente eliminados, completando así la solución de manera más óptima.

- **SwapAssigments**: Permite intercambiar la asignación de dos paquetes entre sus respectivas ofertas. Este intercambio facilita nuevas configuraciones que podrían mejorar el uso de la capacidad y cumplir mejor con las prioridades de entrega. En la implementación de este operador, la función revisa todas las combinaciones posibles de dos paquetes para evaluar si el intercambio es factible. Para hacerlo, se asegura de que los pesos resultantes para ambas ofertas no superen sus respectivas capacidades después del intercambio y que ambas ofertas puedan cumplir con los tiempos de entrega según las prioridades de los paquetes involucrados. Si se cumplen estas condiciones, se genera una acción SwapAssignments que intercambia los dos paquetes entre las ofertas. Este operador es útil para buscar configuraciones que mejoren el uso de los recursos o el cumplimiento de restricciones de tiempo sin necesidad de mover paquetes a nuevas ofertas.

- **RemovePackage**: Elimina un paquete de su asignación actual, dejándolo temporalmente sin una oferta asignada. Este operador resulta útil para liberar capacidad en una oferta cuando no es posible encontrar una mejor configuración sin antes quitar ciertos paquetes. En su implementación, el operador revisa cada paquete asignado y, si el paquete está en una oferta y no está en la lista de paquetes faltantes (es decir, aquellos que no tienen asignación), permite su eliminación. Esto implica marcar el paquete como sin asignación y colocarlo en la lista de paquetes faltantes, lo que indica que deberá ser reasignado posteriormente si se desea una solución completa.

Generación de la solución inicial

- *Generate_initial_state*

La función generate_initial_state crea un estado inicial para el problema de asignación de paquetes a ofertas, tomando en cuenta la prioridad de cada paquete. Primero, genera listas aleatorias de paquetes y ofertas a partir de los parámetros dados: el número de paquetes (num_package), una semilla (seed) y una proporción de peso (proportion). Luego, define un conjunto de parámetros AzamonParameters que contiene información esencial, como el peso máximo de las ofertas, el peso de cada paquete, su prioridad, el límite de días de entrega según la prioridad, las capacidades de cada oferta, los límites de días de entrega de las ofertas y el precio por kilogramo de cada oferta. Estos parámetros configuran el estado del sistema.

En la fase de asignación, sigue un enfoque basado en la prioridad de los paquetes. Organiza los paquetes según su prioridad (de 0 a 2, donde 0 es la prioridad más alta) y, para cada paquete, busca una oferta que pueda acomodar el peso del paquete y que cumpla con el límite de días de entrega según su prioridad. Si encuentra una oferta que cumple con ambas condiciones, el paquete se asigna a esa oferta, y se guarda el índice de la oferta en la lista v_p. Si no hay ninguna oferta adecuada, se añade -1 en v_p para indicar que el paquete no fue asignado. Finalmente, la función devuelve un objeto StateRepresentation que incluye los parámetros (params) y la lista de asignaciones v_p.

- *Generate_initial_state_2*

La función generate_initial_state_2 también genera un estado inicial, pero ignora la prioridad de los paquetes al asignarlos a las ofertas. Al igual que en la primera función, genera listas de paquetes y ofertas de forma aleatoria, y configura los mismos parámetros en AzamonParameters. Sin embargo, en el proceso de asignación, la función no considera la prioridad de los paquetes; en su lugar, intenta asignar cada paquete a una oferta que tenga suficiente capacidad para soportar su peso, sin tener en cuenta los días de entrega asociados a su prioridad.

Si encuentra una oferta con suficiente capacidad, asigna el paquete a esa oferta y registra el índice en v_p. En caso de que no haya ninguna oferta adecuada, añade -1 en v_p para indicar que el paquete no fue asignado. Al igual que en la primera función, el resultado final es un StateRepresentation que contiene los parámetros y la lista de asignaciones v_p.

Funciones heurísticas

- *heuristic_cost*

La heurística heuristic_cost calcula el coste total asociado a la asignación de paquetes considerando transporte, almacenamiento y penalizaciones por incumplimiento de restricciones. Primero, calcula el coste de transporte para cada paquete asignado, multiplicando su peso por el precio por kilogramo de la oferta asignada. A continuación, añade el coste de almacenamiento para las ofertas con tiempos de entrega de tres días o más, multiplicando el peso del paquete por un coste de almacenamiento diario, que depende de los días límite de la oferta. Esta medida intenta desincentivar el uso de ofertas de mayor duración, salvo que sea necesario. Además, la función incorpora penalizaciones para situaciones específicas, como la penalización por entrega tardía, que añade 1000 unidades si el tiempo de entrega de la oferta asignada excede el tiempo máximo permitido para el paquete, en función de su prioridad. También se aplica una penalización por exceso de peso, que se suma si el peso total de los paquetes asignados a una oferta supera su capacidad, penalizando con 500 unidades. Finalmente, se penaliza la falta de asignación de paquetes, con 1000 unidades por cada paquete sin asignación, incentivando la asignación completa de los paquetes. Esta combinación de costes y penalizaciones busca equilibrar el uso eficiente de las ofertas y cumplir con las restricciones, penalizando configuraciones que no optimizan adecuadamente los recursos disponibles o que no respetan las prioridades y limitaciones de entrega.

- *heuristic_happiness*

La función heuristic_happiness se basa en la función update_happiness, que mide la satisfacción de los clientes en función de cuán bien se cumplen los tiempos de entrega según la prioridad de cada paquete. Para cada paquete, update_happiness establece un número mínimo de días de entrega permitidos según su nivel de prioridad: los paquetes de mayor prioridad requieren una entrega más rápida (por ejemplo, 1 día para prioridad alta). La función luego compara este mínimo de días con el tiempo de entrega permitido por la oferta asignada. Si la oferta cumple con o mejora el tiempo mínimo de entrega, la felicidad se calcula como la diferencia entre estos tiempos, maximizando la satisfacción en la medida en que la entrega sea más rápida de lo necesario. En cambio, si la oferta asignada no cumple con el tiempo mínimo, la felicidad del paquete se establece en 0. Así, la heurística de felicidad se calcula como la suma de estos valores de felicidad para todos los paquetes, proporcionando una medida de satisfacción total que refleja cuán bien se están cumpliendo las expectativas de entrega. Este enfoque incentiva configuraciones que prioricen el cumplimiento de los tiempos de entrega, maximizando así la experiencia positiva del cliente.

- *heuristic_cost_happy*

La heurística heuristic_cost_happy combina el coste logístico y la felicidad del cliente en una sola métrica ponderada, lo que permite al modelo equilibrar eficiencia y satisfacción en la asignación de paquetes a ofertas. La función utiliza un parámetro de ponderación alpha que

controla la importancia relativa de ambos factores: si alpha es bajo (cercano a 0), el modelo se enfoca principalmente en minimizar el coste total, que incluye transporte, almacenamiento y penalizaciones, sin dar prioridad a la felicidad del cliente. En cambio, si alpha es alto (cercano a 1), el modelo da más peso a la felicidad, considerando aceptable asumir mayores costes si se cumple con los tiempos de entrega esperados y, en consecuencia, se aumenta la satisfacción del cliente. La fórmula es (1 - alpha) * heuristic_cost() - (alpha * heuristic_happiness()), en la cual la primera parte minimiza el coste y la segunda maximiza la felicidad, proporcionando una métrica flexible que adapta el balance entre eficiencia logística y experiencia del cliente según el valor de alpha.

Experimentos

Primer experimento

Condiciones:

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 50 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.

Caracteristícas:

Observación	Es probable que algunos conjuntos de operadores encuentren mejores soluciones que otros y de manera más rápida.
Planteamiento	Usamos los 4 operadores que tenemos y observamos el coste medio de cada combinación de operadores possible
Hipótesis	La mejor combinación es con swap i assign.
Método	Para cada combinación de operadores, ejecutaremos 10 experimentos con semillas del 1 al 10.
Experimento	• Elegiremos 10 semillas aleatorias, una para cada réplica • Ejecutaremos 10 réplicas por cada combinación de operadores y calcularemos las medias • Experimentaremos con problemas de 50 paquetes • Usaremos el algoritmo de Hill Climbing • La proporción de peso será 1,2. • La estrategia de inicialización es la primera (con prioridad) • El heurístico solo tiene en cuenta el coste. • Mediremos el coste medio y el tiempo de ejecución para realizar la combinación.

Tabla 1: Características del experimento 1

Resultados:

Probamos todas las combinaciones y escribimos en una tabla los resultados de las diferentes combinaciones con el coste y tiempo medio.

COSTE MEDIO	TIEMPO MEDIO	OP1	OP2	OP3	OPERADORES
489.10	3.39	True	True	True	Todos
489.10	2.36	True	True	False	Assign, Swap
489.99	0.99	True	False	True	Assign, Remove/Insert
490.02	0.59	True	False	False	Assign
1000.81	3.65	False	True	True	Swap, Remove/Insert
1050.81	3.08	False	True	False	Swap
2348.56	0.09	False	False	True	Remove/Insert

Tabla 2: Resultados de la ejecución del Hill Climbing con las distintas combinaciones de operadores

Análisis resultados:

Podemos observar que la mejor combinación de operadores es el Swap i Assign, es la que nos da un coste más bajo en un tiempo más bajo.

En las pruebas de cada combinación de operadores, podemos observar que utilizar los operadores Remove/Insert no varía significativamente el coste, esto se deba a que por la penalización actual del heurístico no tiene en cuenta esas soluciones, ya que la penalización es muy alta. Además, puede parecer que son mejores, ya que tienen menos tiempo de ejecución, y esto ocurre porque los operadores tienen menos restricciones y, por tanto, va más rápido.

También podemos observar en la media del coste que si no utilizamos el operador Assign, no se llega a encontrar una solución válida (la media incluye casos donde hemos penalizado solución no válida), ya que por sí solo el operador del Swap no puede explorar todo el espacio de soluciones y el generador inicial no te garantiza encontrar una solución válida en todos los casos. Por su lado, como ya explicado anteriormente, por la penalización del heurístico, las opciones que usan Remove/Insert tienen una penalización muy alta y se suelen descartar, ya que los otros operadores suelen encontrar una solución válida sin penalización.

Conclusión:

Hemos llegado a la conclusión que la combinación ideal es swap y assign, ya que nos permite explorar todo el espacio de soluciones, obteniendo resultados de coste mínimo y tiempo aceptable.

Segundo experimento

Condiciones:

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.

Caracteristícas:

Observación	Pueden haber métodos de inicialización que obtienen mejores soluciones
Planteamiento	Escogemos diferentes métodos de inicialización y observamos sus soluciones
Hipótesis	El generador inicial con prioridad obtendrá mejores resultados que el generador asignado sin prioridad.
Método	Para cada método de generación de solución inicial, ejecutaremos 10 experimentos con semillas del 1 al 10.
Experimento	• Elegiremos 10 semillas aleatorias, una para cada réplica • Ejecutaremos 10 réplicas por cada estado generador y calcularemos las medias • Experimentaremos con los parámetros del experimento anterior • Usaremos el algoritmo de Hill Climbing • El heurístico solo tiene en cuenta el coste. • Mediremos el coste medio y el tiempo de ejecución final para cada generador.

Tabla 3: Características del experimento 2

Resultados:

COSTE MEDIO	TIEMPO MEDIO	GENERADOR
489.10	2.64	Con prioridad
489.18	2.24	Sin prioridad

Tabla 4: Resultados de la ejecución del Hill Climbing con dos estrategias de generación de estado inicial

Conclusión:

Dado que los generadores iniciales no garantizan encontrar una solución válida, ambos tienen que hacer modificaciones para acercarse al espacio de soluciones, sin embargo, en el mejor de los casos del generador con prioridad si proporciona una solución válida, mientras que el generador sin prioridad no dará solución válida, ya que no tiene en cuenta esa restricción.

Por tanto, tiene sentido que el generador con prioridad te proporcione una solución con un coste medio ligeramente mejor.

Tercer experimento

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.

Caracteristícas:

Observación	Pueden haber parámetros del Simulated Annealing que proporcionen mejores resultados que otros.
Planteamiento	Escogeremos una serie de valores para el parámetro k, unos para el parámetro λ y determinaremos un límite de iteraciones máximas para observar el beneficio medio de cada uno así como el tiempo de ejecución.
Hipótesis	Los parámetros k = 20 y λ = 0.005 darán los mejores resultados.
Método	Para cada combinación de parámetros realizaremos 5 ejecuciones en cada una de las 10 semillas, con el objetivo de que los resultados sean menos susceptibles a la aleatoriedad del Simulated Annealing. Los valores de k serán 0.001, 0.05, 1, 10 y 20. Los valores de λ serán 0.001, 0.005, 0.01, 0.1 y 0.5 Una vez escogidos los parámetros miraremos mediante gráficos si el número de iteraciones (1000) es el adecuado y si se podría reducir o aumentar.
Experimento	• Elegiremos 10 semillas aleatorias, una para cada réplica • Ejecutaremos 10 réplicas 5 veces para cada combinación de parámetros • Experimentaremos con el mismo escenario, heurística, operadores y estrategia que en el experimento anterior. • Usaremos el algoritmo de Simulated Annealing • Mediremos el coste medio y el tiempo de ejecución final para cada combinación de parámetros

Tabla 5: Características del experimento 3

La razón por la que hemos escogido estos valores para k y lambda para probar diferentes combinaciones, es porque preferimos que el enfriamiento del algoritmo simulated annealing sea más lento para hacer una búsqueda más exhaustiva y manteniendo valores bajos de k para reducir la posibilidad de aceptar soluciones peores.

Resultados:

Inicialmente, el experimento tiene un límite de 1000 pasos y los resultados para cada combinación han sido los siguientes pasos:

Coste	**0.001**	**0.05**	**1**	**10**	**20**
0.001	598.39	580	602.12	529.61	551.56
0.005	569.65	589.82	550.03	551.23	494.37
0.01	590.15	609.34	600.49	540.08	629.17
0.1	499.89	549.57	579.69	620.33	688.89
0.5	649.68	579.51	589.39	570	619.89

Tabla 6: Resultados de la función heurística de la ejecución del Simulated Annealing con distintos valores de los parámetros k y lambda con 1000 iteraciones

Tiempo	**0.001**	**0.05**	**1**	**10**	**20**
0.001	0.82	0.82	0.87	0.94	0.95
0.005	0.83	0.82	0.84	0.89	0.91
0.01	0.83	0.83	0.84	0.84	0.86
0.1	0.82	0.82	0.85	0.82	0.86
0.5	0.82	0.83	0.82	0.83	0.82

Tabla 7: Resultados del tiempo de ejecución de la ejecución del Simulated Annealing con distintos valores de los parámetros k y lambda

Análisis resultados:

Con estos resultados podemos concluir que nuestra hipótesis no era correcta, ya que la combinación k = 1 y λ = 0.005 no se encuentra entre ninguna de las combinaciones que proporcionan mejores resultados.

Además, observamos que los mejores resultados son k=20 y lam =0.005 y k=0.001 y lam =0.1. Sin embargo, observando los datos y visualizando los gráficos de la evolución del coste según las iteraciones de algunas réplicas, hemos llegado a la conclusión de que el número de iteraciones es demasiado bajo, ya que no encontraba la solución mínima.

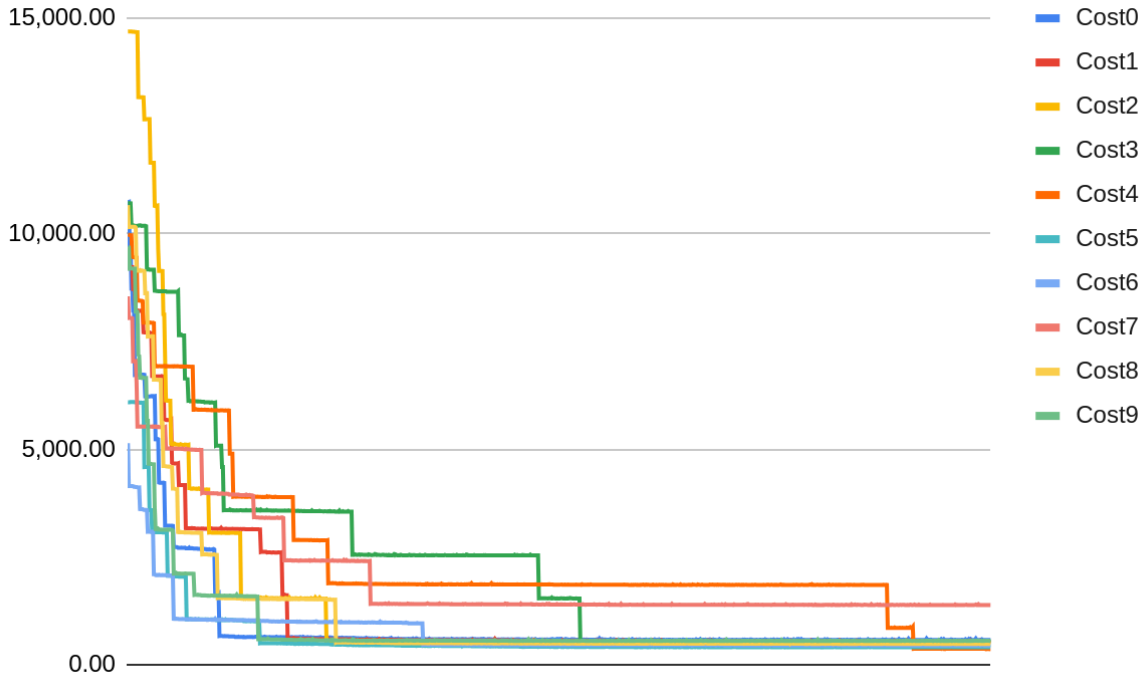

Figura 1: Gráfica del coste respecto a las iteraciones en el caso de k=20 y lambda = 0.1

Como se muestra en la gráfica para algunas semillas, parece ser que el enfriamiento no se ha completado, ya que sigue habiendo variación y , por lo que se necesitaría un número mayor de iteraciones para esos parámetros. Esto tiene sentido teniendo en cuenta que estamos usando valores bastante bajos de lambda.

Para poder asegurar que realmente el valor no pueda ser mejorado por el algoritmo Simulated Annealing con un límite superior, probaremos de realizar el experimento de nuevo, pero con un límite de 5000 pasos para asegurarnos.

En este caso los resultados son:

Coste	**0.001**	**0.05**	**1**	**10**	**20**
0.001	479.09	479.1	479.04	479.34	480.18
0.005	479.09	479.13	479.1	479.1	479.1
0.01	479.13	479.13	479.1	479.1	479.12
0.1	479.1	479.1	479.13	479.13	479.16
0.5	499.18	479.25	479.29	499.19	499.16

Tabla 9: Resultados de la función heurística de la ejecución del Simulated Annealing con distintos valores de los parámetros k y lambda con 5000 iteraciones

Tiempo	0.001	0.05	1	10	20
0.001	4	4.09	4.54	5.27	3.38
0.005	2.62	2.49	2.48	2.65	2.51
0.01	2.52	2.43	2.34	2.48	2.43
0.1	2.52	2.58	2.66	2.76	2.68
0.5	0.81	0.8	0.91	0.83	0.94

Tabla 10: Resultados del tiempo de ejecución de la ejecución del Simulated Annealing con distintos valores de los parámetros k y lambda con 5000 iteraciones

Podemos observar que en el coste medio de las combinaciones al hacer 5000 iteraciones no hay mucha variación, son bastante similares entre ellos, sin embargo, el tiempo de ejecución si varía significativamente a diferencia de en las pruebas con 1000 iteraciones donde el tiempo de ejecución era más o menos constante.

También se puede observar que el coste es significativamente más bajo de manera general, lo que nos lleva a pensar que el número de iteraciones es más que suficiente.

Conclusiones:

De las pruebas realizadas, los mejores resultados están expresados en color verde, para mejor visualización, tanto para el coste como para el tiempo de ejecución.

Las conclusiones que podemos extraer de estos datos es que dependiendo del número de iteraciones que decidamos utilizar deberemos utilizar unos parámetros u otros.

Como al hacer 1000 iteraciones el tiempo de ejecución se mantiene más o menos constante y además valoramos más minimizar el coste heurístico, nos fijamos en las combinaciones con un menor valor, que en este caso corresponde a la combinación k=20 y lam=0.005.

En cambio, si efectuamos 5000 iteraciones, el coste heurístico tiene mínima variancia, pero el tiempo de ejecución contiene valores tan dispares como 0.8 y 5.27: En consecuencia nos sale más a cuenta escoger una combinación con un menor tiempo de ejecución, ya que a escala mayor y considerando futuras aplicaciones, ya que no necesitaremos tanta potencia de computación. Estas condiciones las cumple la combinación k=0.05 y lam=0.5

Entre las dos combinaciones, el coste medio es ligeramente menor para la combinación k=0.05 y lam=0.5 realizada con 5000 iteraciones y como mencionado anteriormente este hecho es al que le damos más peso. Por tanto, será la que utilizaremos para el resto de experimentos.

Cuarto experimento

Dado el escenario de los apartados anteriores, estudiad cómo evoluciona el tiempo de ejecución para hallar la solución en función del número paquetes y la proporción de peso transportable. Fijad el número de paquetes en 50 e id aumentando la proporción del peso transportable desde 1, 2 de 0, 2 en 0, 2 hasta ver la tendencia. Fijad la proporción de peso en 1,2 e id aumentando el número de paquetes desde 50 de 10 en 10 hasta ver la tendencia. Usad el algoritmo de Hill Climbing y la misma heurística.

Caracteristícas:

Observación	El tiempo de ejecución varía en función de la complejidad del problema (número de paquetes y proporción de peso).
Planteamiento	Queremos comprobar cómo afecta la complejidad del problema al tiempo de ejecución del programa, aumentando gradualmente el número de paquetes y la proporción.
Hipótesis	El tiempo de ejecución incrementa exponencialmente cuando aumenta el número de paquetes o su proporción.
Método	Para la primera parte, iremos generando soluciones iniciales, aumentando la proporción de 0.2 en 0.2 y ejecutando un HIll Climbing, dejando fijo el número de paquetes. Para la segunda parte, seguiremos el mismo método, pero dejando fija la proporción y aumentando el número de paquetes
Experimento	• Elegiremos 10 semillas aleatorias, una para cada réplica. • Ejecutaremos 10 réplicas para cada parte de experimento • Experimentaremos con la misma heurística usada anteriormente, variando el número de paquetes o la proporción de peso. • Usaremos el algoritmo de Hill Climbing • Mediremos la evolución del tiempo de ejecución

Tabla 11: Características del experimento 4

Resultado aumentando la proporción:

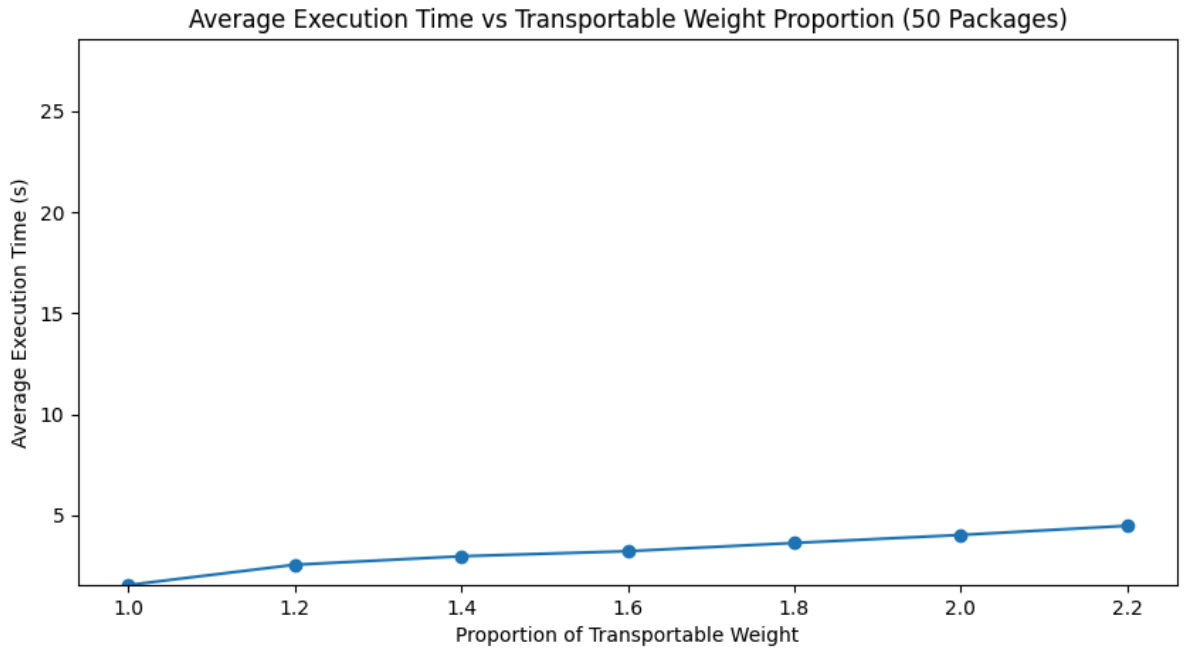

Figura 2: Gráfica de la media de tiempo de ejecución respecto al aumento de proporción de peso

Resultados aumentando el número de paquetes:

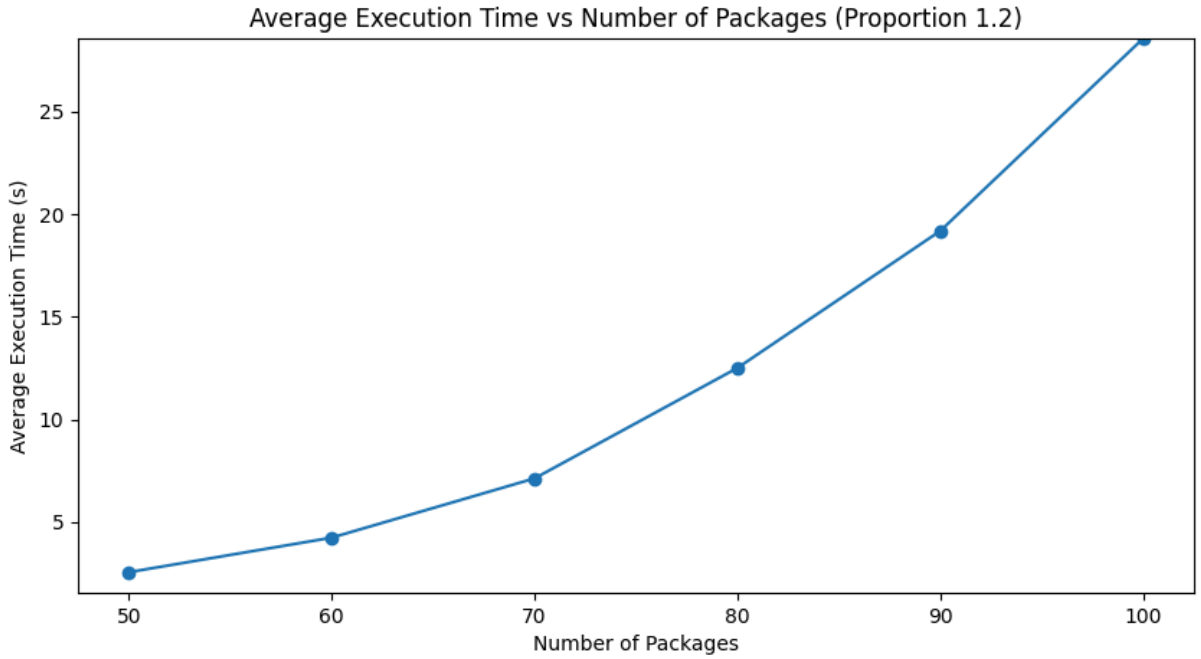

Figura 3: Gráfica de la media del tiempo de ejecución respecto al aumento del número de paquetes

Análisis y conclusiones:

A partir de las gráficas generadas, se observa que el tiempo de ejecución promedio aumenta gradualmente cuando se incrementa la proporción de peso transportable, manteniendo constante el número de paquetes en 50. Este comportamiento sugiere que al elevar la proporción de peso, la complejidad del problema también aumenta un poco, lo que incrementa ligeramente el tiempo de ejecución.

En cambio, a la hora de fijar la proporción en 1.2 e ir aumentando el número de paquetes, se observa que el tiempo de ejecución promedio muestra un crecimiento acelerado, cercano al exponencial. A medida que se incrementa el número de paquetes, el tiempo de ejecución aumenta de forma notable, alcanzando un valor mucho mayor cuando se llega a 100 paquetes. Este comportamiento indica que el aumento en el número de paquetes incrementa significativamente la complejidad computacional.

En conjunto, las dos gráficas sugieren que el número de paquetes tiene un impacto mucho mayor en el tiempo de ejecución que la proporción de peso. Mientras que aumentar el número de paquetes produce un incremento pronunciado en el tiempo de ejecución, aumentar la proporción de peso solo causa un crecimiento leve. Esto indica que la complejidad del problema depende más de la cantidad de elementos a procesar (número de paquetes) que de las restricciones específicas (proporción de peso).

Volviendo al planteamiento de la hipótesis inicial, podemos afirmar que al aumentar el número de paquetes el tiempo sí se incrementa de manera exponencial, pero al aumentar la proporción el tiempo sigue una tendencia de crecimiento más gradual y lineal.

Quinto experimento

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

Caracteristícas:

Observación	El tiempo de ejecución varía en función de la complejidad del problema (número de paquetes y proporción de peso).
Planteamiento	Queremos comprobar cómo afecta el aumentar la proporción al coste de transporte y almacenamiento.
Hipótesis	El coste se reduce al aumentar la proporción de peso.
Método	Para la primera parte, iremos generando soluciones iniciales, aumentando la proporción de 0.2 en 0.2 y ejecutando un Hill Climbing, dejando fijo el número de paquetes. Para la segunda parte, seguiremos el mismo método, pero dejando fija la proporción y aumentando el número de paquetes
Experimento	• Elegiremos 10 semillas aleatorias, una para cada réplica. • Ejecutaremos 10 réplicas para cada parte de experimento • Experimentaremos con la misma heurística usada anteriormente, variando el número de paquetes o la proporción de peso. • Usaremos el algoritmo de Hill Climbing • Mediremos la evolución del tiempo de ejecución

Tabla 12: Características del experimento 5

Resultados:

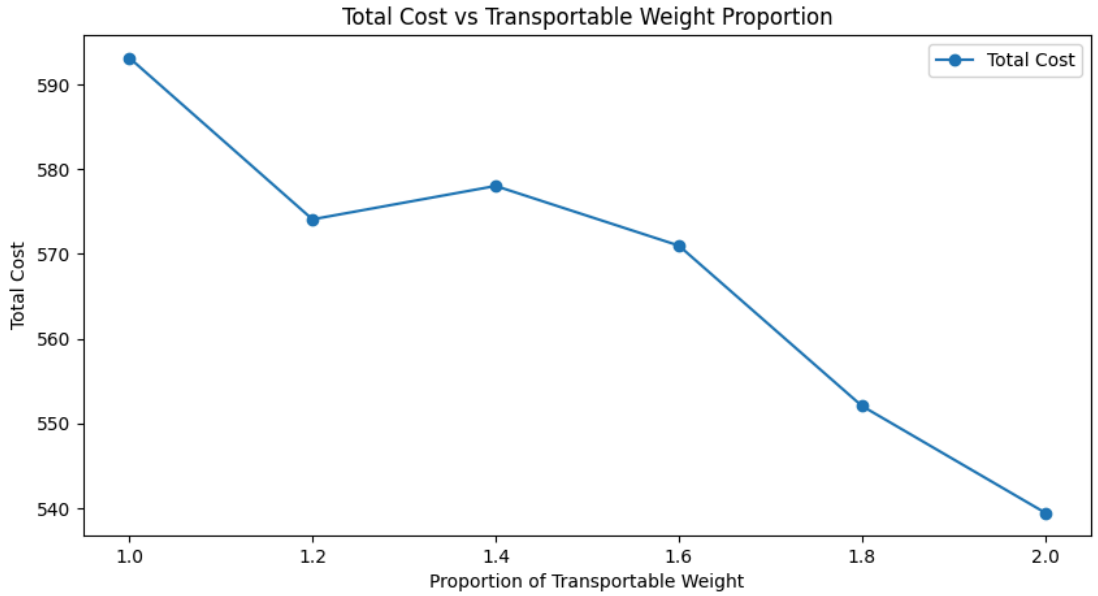

Figura 4: Gráfica del valor de la heurística de coste respecto al aumento de la proporción de peso

Conclusiones:

El experimento demuestra que incrementar la proporción de peso transportable en las ofertas contribuye a reducir el coste total de transporte y almacenamiento. En las primeras proporciones, los costes se mantienen altos, lo cual sugiere que una capacidad limitada de transporte dificulta una distribución óptima de los paquetes, incrementando el coste Sin embargo, conforme la proporción de peso transportable aumenta, se observa una disminución constante en el coste total, lo que indica que el algoritmo puede aprovechar mejor las opciones de transporte disponibles, permitiendo reducir el número de trayectos necesarios.

En resumen, el incremento de la proporción de peso transportable es una estrategia efectiva para reducir costes, especialmente en las primeras etapas. Sin embargo, hay un límite en el que continuar aumentando la proporción no genera beneficios adicionales considerables. Por lo tanto, una proporción de peso moderadamente alta, cercana a 1.8, parece ser la mejor opción para equilibrar eficiencia y costes en el sistema.

A partir de los resultados obtenidos, la hipótesis de que el coste total se reduce al aumentar la proporción de peso transportable ha sido confirmada en gran medida. Al incrementar la proporción de peso, se observa una disminución clara y consistente en el coste de transporte y almacenamiento, lo que indica que el sistema es capaz de manejar los envíos de manera más eficiente cuando las opciones de transporte permiten una mayor capacidad.

Sexto experimento

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.

Caracteristícas:

Observación	El coste de transporte y almacenamiento y el tiempo de ejecución varía en función de la ponderación del coste de la felicidad.
Planteamiento	Queremos comprobar cómo afecta la importancia de la felicidad en el coste de transporte y almacenamiento y al tiempo de ejecución del programa, aumentando gradualmente la ponderación que multiplica la felicidad.
Hipótesis	El coste de transporte y almacenamiento incrementa y el tiempo de ejecución disminuye cuando aumenta la ponderación de la felicidad.
Método	Calcularemos una heurística combinada en la que se tenga en cuenta tanto el coste de transporte y almacenamiento como la felicidad. A cada uno de los costes le daremos una ponderación de importancia (0.1- 0.9) que entre los dos deben sumar 1. Comenzaremos con 0.1 en la felicidad y lo iremos aumentando, y disminuyendo en el coste de transporte y almacenamiento, y lo iremos aumentando de 0.1 en 0.1 para poder ver la tendencia.
Experimento	• Elegiremos 10 semillas aleatorias, una para cada réplica. • Ejecutaremos 10 réplicas para cada parte de experimento • Experimentaremos con la heurística combinando la felicidad y coste de transporte y almacenamiento, variando ponderación de la importancia de la felicidad. • Usaremos el algoritmo de Hill Climbing • Mediremos la tendencia del tiempo de ejecución

Tabla 13: Características del experimento 6

Resultados:

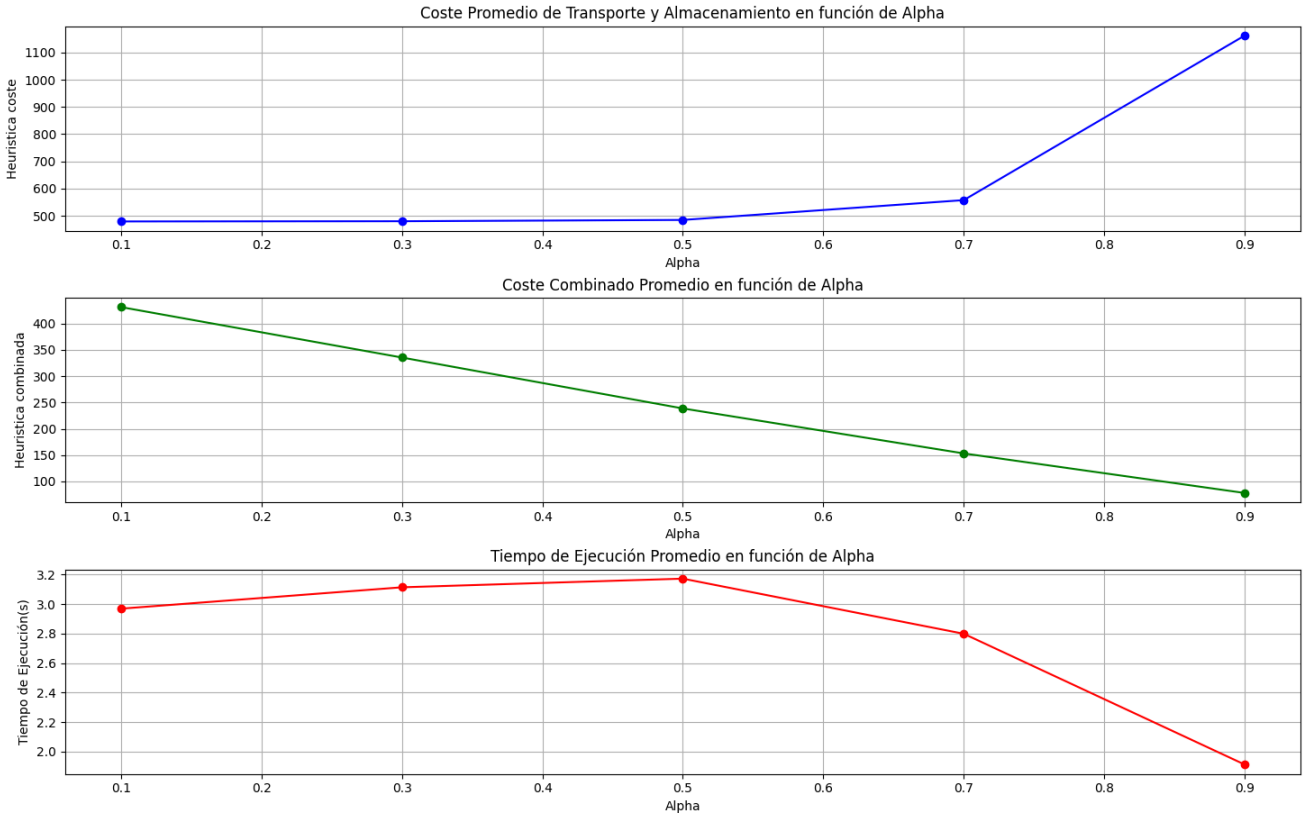

Figura 5: Gráficas que corresponden a los valores medios de la heurística de coste, valores medios de la heurística combinada y el tiempo medio de ejecución para cada valor de la ponderación de la heurística combinada (alpha), ejecutando con Hill Climbing

Conclusiones:

Los resultados indican que aumentar el valor de α, que pondera la felicidad en la función heurística, permite encontrar un equilibrio más favorable entre los costes de transporte y almacenamiento y la satisfacción de los usuarios. A medida que α crece, el coste promedio de transporte y almacenamiento aumenta ligeramente, ya que el algoritmo prioriza soluciones que buscan cumplir con los tiempos de entrega para maximizar la felicidad de los usuarios, lo que puede requerir opciones de transporte más rápidas y costosas. Sin embargo, el coste combinado, que integra tanto el coste de transporte y almacenamiento como la felicidad, disminuye consistentemente a medida que α se incrementa. Esto sugiere que priorizar la felicidad no solo mejora la satisfacción del usuario, sino que también lleva a soluciones donde el coste global se reduce, mostrando que un enfoque equilibrado entre coste y felicidad puede resultar más eficiente en términos de la función combinada.

Además, el tiempo de ejecución sigue una tendencia interesante: es mayor en valores intermedios de α (alrededor de 0.5), donde el algoritmo necesita explorar más opciones para balancear ambos objetivos, pero disminuye a medida que α se acerca a valores altos (0.9). Esto se debe a que con valores altos de α, el algoritmo converge más rápidamente al

enfocarse en maximizar la felicidad, dado que la heurística de felicidad no tiene penalizaciones, por lo que no se ejecutan bucles y el tiempo de ejecución se reduce notablemente. En conclusión, una α alto, aproximadamente entre 0.5 y 0.7, proporciona un buen balance entre coste y felicidad, permitiendo maximizar la satisfacción del usuario y optimizando el coste global de manera eficiente.

Séptimo experimento

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

Características:

Observación	El coste de transporte y almacenamiento y el tiempo de ejecución varía en función de la ponderación del coste de la felicidad .
Planteamiento	Queremos comprobar cómo afecta la importancia de la felicidad en el coste de transporte y almacenamiento y al tiempo de ejecución del programa, aumentando gradualmente la ponderación que multiplica la felicidad.
Hipótesis	El coste de transporte y almacenamiento incrementa y el tiempo de ejecución disminuye cuando aumenta la ponderación de la felicidad.
Método	Calcularemos una heurística combinada en la que se tenga en cuenta tanto el coste de transporte y almacenamiento como la felicidad. A cada uno de los costes le daremos una ponderación de importancia (0.1- 0.9) que entre los dos deben sumar 1. Comenzaremos con 0.1 en la felicidad y lo iremos aumentando, y disminuyendo en el coste de transporte y almacenamiento, de 0.1 en 0.1 para poder ver la tendencia.
Experimento	• Elegiremos 10 semillas aleatorias, una para cada réplica. • Ejecutaremos 10 réplicas para cada parte de experimento • Experimentaremos con la heurística combinando la felicidad y coste de transporte y almacenamiento ,variando ponderación de la importancia de la felicidad. • Usaremos el algoritmo de Simulated Annealing • Mediremos la tendencia del tiempo de ejecución

Tabla 14: Características del experimento 7

Resultados:

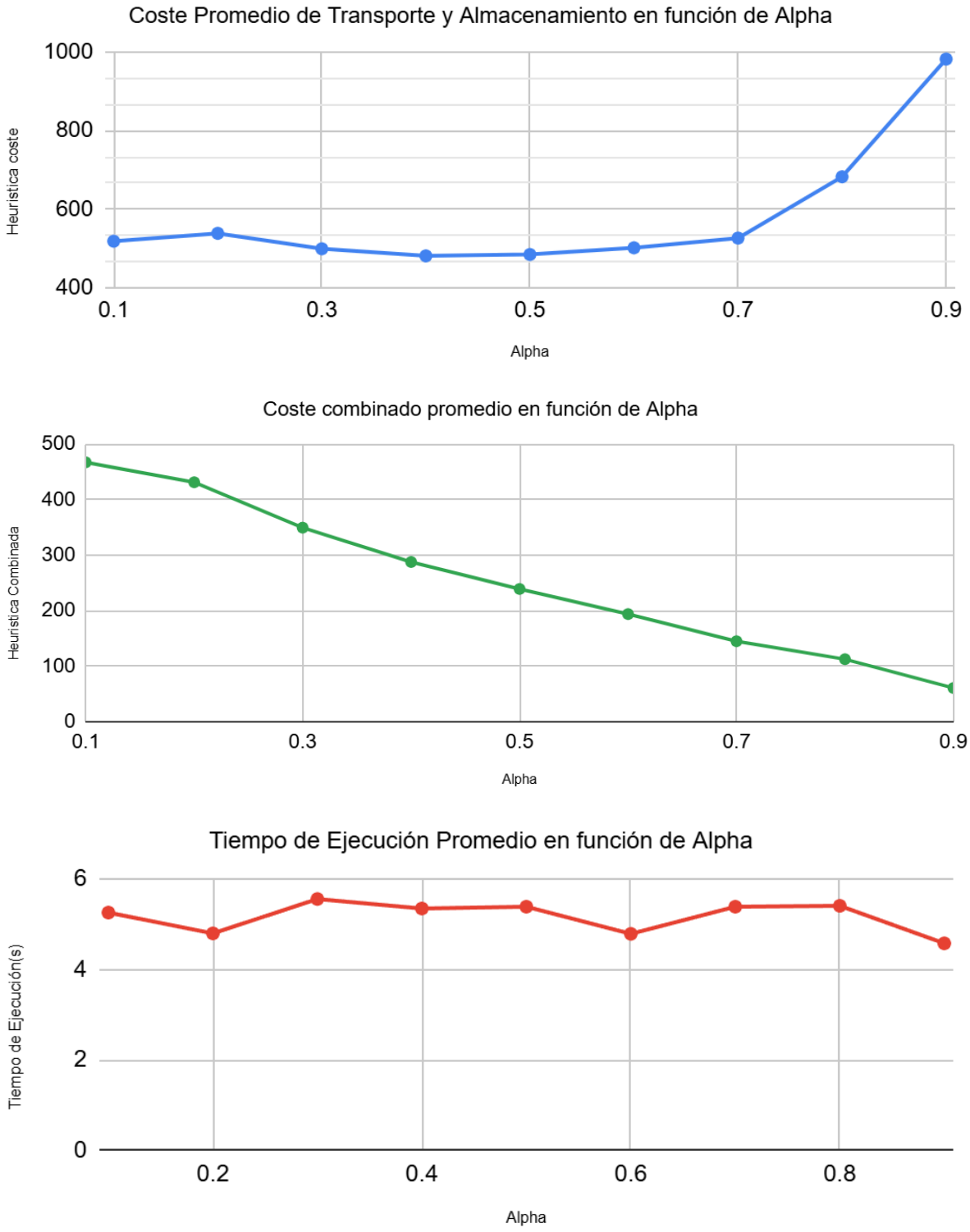

Figura 6: Gráficas que corresponden a los valores medios de la heurística de coste, valores medios de la heurística combinada y el tiempo medio de ejecución para cada valor de la ponderación de la heurística combinada (alpha), ejecutando con Simulated Annealing

Conclusiones:

Los resultados nos muestran que al aumentar el valor α ocurre un fenómeno muy similar al del apartado anterior a cuando ejecutábamos el hill climbing.

A medida que α crece, el coste promedio de transporte y almacenamiento aumenta ligeramente, ya que el algoritmo prioriza soluciones que buscan cumplir con los tiempos de entrega para maximizar la felicidad de los usuarios. Sin embargo, al igual que con el hill climbing, el coste combinado, que integra tanto el coste de transporte y almacenamiento como la felicidad, disminuye consistentemente a medida que α se incrementa.

Esto sugiere que priorizar la felicidad no solo mejora la satisfacción del usuario, sino que también lleva a soluciones donde el coste global se reduce, mostrando que un enfoque equilibrado entre coste y felicidad puede resultar más eficiente en términos de la función combinada.

Además, el tiempo de ejecución sigue una tendencia diferente al Hill Climbing, ya que en este caso el tiempo parece ser bastante constante. A pesar de cómo cambie los valores de las heurísticas, el algoritmo mantiene el mismo tiempo de ejecución, lo que nos indica que no cambia el coste computacional del algoritmo. Esto se debe a que la forma en la que el algoritmo se mueve por el espacio de soluciones buscando una no cambia.

En conclusión, una α alto, aproximadamente entre 0.5 y 0.7, proporciona un buen balance entre coste y felicidad, permitiendo maximizar la satisfacción del usuario y optimizando el coste global de manera eficiente.

Octavo experimento

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

Si aumentamos este precio, el peso del coste de almacenamiento en la función heurística también se incrementará. Esto hará que el sistema busque mantener los paquetes en el almacén el menor tiempo posible, utilizando más ofertas para evitar la acumulación. Como resultado, es probable que aumente la "felicidad" al intentar enviar los paquetes con mayor rapidez.

Por otro lado, si el precio disminuye, el peso del coste de almacenamiento se reducirá, lo cual también afectará al coste total de la heurística. Con un coste de almacenamiento más bajo, el sistema le dará menos importancia al tiempo de permanencia en el almacén, lo que reducirá el coste logístico. Sin embargo, esto puede tener un impacto negativo en la felicidad, ya que los paquetes podrían tardar más en ser enviados.

Anexos

Para evitar un documento largo y pesado con todos los resultados de los experimentos, hemos decidido utilizar aquí las tablas y gráficas más relevantes para analizar y extraer conclusiones, y crear un documento aparte donde se han documentado todos los resultados obtenidos. El anexo se ha incluido dentro del archivo zip.