UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Práctica de Búsqueda Local

Inteligencia artificial

Algorismes Bàsics per la Intel·Ligència Artificial

Autors

Artur Aubach Altes

Daniel López García

Grup 11

Profesor: Miquel Rodríguez Sansaloni

Quadrimestre Primavera 2023/2024

TABLA DE CONTENIDOS

1. IDENTIFICACIÓN DEL PROBLEMA

1.1 Contextualización

En el contexto urbano contemporáneo, los servicios de préstamo de bicicletas se han convertido en una solución esencial para la movilidad sostenible y eficiente. Sin embargo, estos servicios enfrentan desafíos operativos significativos. Uno de los problemas más persistentes es la distribución desigual de bicicletas en diferentes estaciones a lo largo del día, afectando la accesibilidad y la eficiencia del servicio.

1.2 Descripción del Problema

Bicing, un prominente servicio de préstamo de bicicletas, se encuentra en este desafío. A lo largo del día, las bicicletas tienden a acumularse en estaciones particulares, creando un desequilibrio que afecta la disponibilidad y accesibilidad para los usuarios en otras áreas de la ciudad. Bicing ha recolectado datos estadísticos detallados sobre la demanda y el movimiento de bicicletas, proporcionando una base sólida para abordar este problema.

La ciudad, conceptualizada como un cuadrado de 10x10 kilómetros con calles que forman una cuadrícula, donde cada manzana tiene 100×100 metros. Las estaciones de Bicing se encuentran en los cruces entre las calles. Aunque la demanda de bicicletas varía, la cantidad total de bicicletas, B, y estaciones, E, permanece constante. Bicing enfrenta el reto de optimizar la distribución de estas bicicletas para satisfacer la demanda prevista en cada estación, cada hora.

1.3 Elementos del Problema

La empresa cuenta con una flota de F furgonetas, cada una con la capacidad de transportar hasta 30 bicicletas. Estas furgonetas son cruciales para redistribuir las bicicletas y acercarse a la demanda prevista. Sin embargo, existen restricciones: cada furgoneta puede realizar un único viaje por hora, y no puede haber dos furgonetas que recojan bicicletas de la misma estación.

Bicing nos paga un euro por cada bicicleta que transportemos para satisfacer la demanda, pero también nos cobra un euro por cada bicicleta que aleje a una estación de su previsión. Además, el transporte de bicicletas incurre en costos variables dependiendo del número de bicicletas transportadas y la distancia recorrida.

$$((nb + 9)\ div\ 10),\ div\ es\ división\ entera$$

1.4 Objetivo

Nuestro objetivo es desarrollar una estrategia eficiente para redistribuir las bicicletas, teniendo en cuenta las previsiones de demanda, las restricciones operativas y los costos asociados. Buscamos maximizar la satisfacción del cliente asegurando que las bicicletas estén disponibles donde y cuando se necesiten, al mismo tiempo que minimizamos los costos operativos para Bicing y para nuestra operación de transporte.

1.5 Estrategia Propuesta y Búsqueda Local

Enfocamos nuestra estrategia en la colocación estratégica de furgonetas para transportar bicicletas entre estaciones, buscando maximizar el beneficio derivado de satisfacer la demanda fluctuante de las diferentes ubicaciones. Creemos esto porque el hecho de mover una bici de una estación que ya cubre su demanda y más a una estación que no llega a su demanda nos proporciona unos ingresos mientras que el hecho de que sacar bicis de una estación que no llegue a su demanda nos provoca una penalización. Este hecho, combinado a un coste asociado al transporte de las bicis, nos permite encontrar una colocación que satisfaga mejor las distintas demandas de las estaciones logrando una distribución más eficiente de las bicicletas.

Nuestro enfoque se basa en la búsqueda local, un método que persigue encontrar la mejor solución posible, maximizando un parámetro (en nuestro caso, el beneficio asociado al transporte de bicicletas) dentro de un espacio de soluciones concretas mediante modificaciones a una solución inicial a través de operadores. Usamos este enfoque porque no se nos pide encontrar la solución óptima, sinó la mejor solución posible, y esto es precisamente lo que hace la búsqueda local.

2. IMPLEMENTACIÓN DEL ESTADO

2.1 Estructura del Estado

La implementación del estado en nuestro sistema se basa en la representación y manipulación de las estaciones de Bicing y las furgonetas utilizadas para el traslado de bicicletas. El estado se define por los siguientes componentes:

- **Traslados:** Una lista de las furgonetas, donde cada furgoneta está asociada con su origen, destinos y la cantidad de bicicletas que transporta.
- **Parámetros:** Los parámetros asociados con el estado actual, incluyendo la cantidad total de bicicletas, la distancia recorrida por las furgonetas, y el número de furgonetas asignadas.
- **Beneficio Total:** El beneficio acumulado, calculado en base a las bicicletas transportadas que satisfacen la demanda y los costos asociados con el transporte (dependerá según el heurístico).

2.2 Componentes del Estado

2.2.1 Estaciones

La clase *Estacion* representa una estación individual de Bicing. Cada estación tiene coordenadas específicas (X, Y) y atributos que indican el número de bicicletas no utilizadas en la hora actual y la previsión del número de bicicletas en la siguiente hora. La lista de estaciones se maneja a través de la clase *Estaciones*, que inicializa y gestiona las estaciones basadas en una semilla para la generación de números aleatorios.

2.2.1 Furgonetas

La clase *furgonetas* representa las furgonetas utilizadas para el traslado de bicicletas entre estaciones. Cada furgoneta tiene un origen, uno o dos destinos, y la cantidad de bicicletas que transporta a cada destino. La igualdad entre furgonetas se determina comparando sus orígenes y destinos.

2.3 Representación del Estado

El estado se representa como una cadena que incluye los traslados de furgonetas, los parámetros asociados, el beneficio total acumulado, la distancia total recorrida

por las furgonetas y el número de furgonetas asignadas. Esta representación facilita la visualización y el seguimiento del estado actual del sistema.

Ejemplo de Representación del Estado:

```
return f"Trasllats={([str(i) for i in self.list_furgos])} | Pararetres:
{self.params} | beneficio_total:{self.beneficio_total}
```

En cuanto a como está implementado en el código(lo de arriba es únicamente como lo mostramos por pantalla) nuestro estado son instancias de una clase llamada `StateRepresentation` que se compone de tres partes(tiene tres argumentos):

- self.params: los parámetros del problema
- self.list_est: una lista de instancias de la clase Estación
- self.list_furgos: una lista de instancias de una nueva clase creada llamada furgonetas.

La nueva clase: esta nueva clase furgonetas contiene:

- self.origen: la estación de donde sale la furgo
- self.dest1: la primera estación de destino
- self.dest2: si hay, la segunda estación de destino. En caso contrario None
- self.cant_bicis1: la cantidad de bicis que transportamos al primer destino
- self.cant_bicis2: si hay un segundo destino, la cantidad de bicis que transportamos al mismo. En caso contrario 0.

Por último cabe recalcar que hemos realizado cambios en los métodos __eq__ para la clase furgonetas y la clase estación para de esta manera poder comparar fácilmente instancias de esta clase y, únicamente para la clase estación, hemos modificado su método hash.

3. OPERADORES

La identificación y aplicación de operadores eficientes es un componente esencial en la búsqueda de soluciones óptimas en el espacio de estados. Los operadores son acciones que transforman un estado en otro, y su selección adecuada es crucial para garantizar que se explore todo el espacio de soluciones posibles. En esta sección, se presentan varios operadores que podrían ser aplicados. Sin embargo, es importante mencionar que no todos ellos pueden ser utilizados en la implementación final del algoritmo, ya que su eficacia puede variar dependiendo del estado inicial y otros factores.

3.1 Operadores Definidos

3.1.1 Crear Furgo *(crea_furgo)*

Este operador se encarga de crear una nueva furgoneta para el traslado de bicicletas. Se especifica el origen, el destino y la cantidad de bicicletas a transportar. Únicamente asigna un destino. Tiene determinadas restricciones: solo podrá crear una furgo si no hemos superado el límite de furgos y si la estación de origen que estamos asignando no tiene ninguna furgo ya asignada además de comprobar que la estación de destino no sea igual que la de origen

- **origin:** La estación de origen donde la furgoneta recogerá las bicicletas.
- **bikes1:** La cantidad de bicicletas que la furgoneta llevará en su primera entrega.
- **dest1:** La primera estación de destino donde la furgoneta entregará las bicicletas.

En caso de cojer 1 bicicleta:

Coste de ramificación: $O\ (estaciones^2)$

En caso de cojer rang(maxBicis) bicicletas:

Coste de ramificación: $O\ (estaciones^2 * maxbicicletas)$

3.1.2 Borrar Furgo *(borrar_furgo)*

Este operador elimina una furgoneta existente del estado actual. Es útil para ajustar o reducir el número de furgonetas en operación. Solo tiene una restricción, y es que haya ya alguna furgoneta asignada(la longitud de la lista de instancias de furgonetas es mayor que 0)

- **furgo:** La furgoneta que se eliminará del estado actual.

Coste de ramificación: $O\ (maxfurgo)$

3.1.3 Mover Origen de Furgo *(furgo_move_origin)*

Permite cambiar el origen de una furgoneta existente. Tiene dos restricciones: que la nueva estación de origen que estamos dando no tenga ninguna furgoneta y que el origen sea distinto a los destinos(comprueba también el segundo destino si hay) de la furgoneta.(además de que previamente se había comprobado que hubieran furgonetas asignadas)

- **furgo:** La furgoneta cuyo origen se cambiará.
- **originnew:** La nueva estación de origen.

Coste de ramificación: $O\ (estaciones\ \ *\ maxfurgo)$

3.1.4 Modificar Primera Parada de Furgo (furgo_move_dest)

Cambia el primer destino de una furgoneta por un nuevo destino. Tiene restricciones comprobando que el nuevo destino sea distinto a la estación de origen de la furgo, al viejo primer destino y al segundo destino(si hay).(además de que previamente se había comprobado que hubieran furgonetas asignadas)

- **furgo:** La furgoneta que se modificará.
- **destnew1**: La nueva primera estación de destino.

Coste de ramificación: $O\ (estaciones\ \ *\ maxfurgo)$

3.1.5 Modificar Segunda Parada de Furgo *(asignar_dest2)*

Este operador asigna un nuevo segundo nuevo destino a una furgoneta ya creada. Si a la furgo todavía no se le había asignado un nuevo destino(dest2=None) además del nuevo destino se le proporcionaba una cantidad de bicicletas(1). Si ya tenía un segundo destino asignado simplemente cambiamos este destino por el nuevo destino proporcionado. Comprobamos que el nuevo segundo destino sea distinto a la estación de origen, a la primera estación de destino de la furgo y, si ya tenía la furgo una segunda estación de destino asignada, que sea distinta a la antigua segunda destinación(además de que previamente se había comprobado que hubieran furgonetas asignadas)

- **furgo:** La furgoneta que se modificará.
- **dest2** La nueva segunda estación de destino.
- **bikes2:** La nueva cantidad de bicicletas que se entregarán al segundo destino(si no existía antes en la furgo).

Coste de ramificación: $O\ (estacion\ *\ maxfurgo)$

3.1.6 Incrementar Número de Bicicletas Transportadas *(incrementar_num_bici)*

Este operador aumenta la cantidad de bicicletas que una furgoneta está transportando al destino indicado en una unidad. Aumentaremos el primer y segundo destino si un bool, que por defecto es False, es True. En caso contrario solo aumentaremos en 1 el primer destino. Este bool se proporcionará al operador como verdadero únicamente si existe un segundo destino. Únicamente tiene como restricción que la cantidad de bicicletas que está transportando la furgoneta no sea superior al límite(30).

- **furgo:** La furgoneta cuya carga de bicicletas se modificará.
- **bibic_nou1:** La nueva cantidad de bicicletas(la antigua cantidad + 1).
- **dest2:** booleano true or false, es True si existe una segunda destinación y es False por defecto. Nos indica si la cantidad de bicis a aumentar es la de la primera destinación o la de la segunda

En caso de sumar:

Coste de ramificación: $O\ (furgos)$

3.1.7 Reducir Número de Bicicletas Transportadas *(reduir_num_bici)*

Este operador reduce la cantidad de bicicletas que una furgoneta está transportando al destino indicado en una unidad. Reduciremos el primer y segundo destino si un bool, que por defecto es False, es True. En caso contrario sólo disminuimos en 1 el primer destino. Este bool se proporcionará como verdadero únicamente si existe un segundo destino. Únicamente tiene como restricción que la cantidad de bicicletas que está transportando la furgoneta no sea superior al límite(30).

- **furgo:** La furgoneta cuya carga de bicicletas se modificará.
- **bibic_nou1:** La nueva cantidad de bicicletas(la antigua cantidad + 1).

- **dest2:** booleano true or false, es True si existe una segunda destinación y es False por defecto. Nos indica si la cantidad de bicis a aumentar es la de la primera destinación o la de la segunda

Coste de ramificación: $O\ (furgos)$

4. ESTRATEGIAS PARA HALLAR LA SOLUCIÓN INICIAL

Al observar el problema, hemos detectado que la solución inicial juega un papel crucial en la eficiencia y efectividad de las estrategias de optimización posteriores. La calidad de la solución inicial puede influir significativamente en la capacidad del algoritmo para explorar el espacio de soluciones y encontrar una solución óptima. Por lo tanto, hemos decidido implementar tres tipos de soluciones iniciales: una básica y dos estrategias greedy, cada una enfocada en un objetivo específico.

4.1 Solución 1: Estado Vacío

La primera estrategia es la más básica y consiste en iniciar con un estado vacío, representado como *[]*. En esta estrategia, no se asignan furgonetas ni se realiza ningún traslado de bicicletas inicialmente. Esto proporciona un punto de partida neutro, permitiendo que el algoritmo de optimización determine completamente la asignación y el traslado de bicicletas.

Estado vacío, no se realiza ninguna asignación inicial de furgonetas ni traslado de bicicletas.

Coste: O

4.2 Solución 2: Maximización del Beneficio

La segunda estrategia, *Solucio_greedy1*, se enfoca en maximizar el beneficio. Prioriza la recogida del mayor número posible de bicicletas para potenciar el beneficio. Se identifican las estaciones con exceso de bicicletas y aquellas con déficit, y se asignan furgonetas para trasladar las bicicletas de las estaciones con exceso a las que tienen déficit. La cantidad de bicicletas a trasladar se optimiza para asegurar que se maximice el beneficio.

Se implementa la función *Solucio_greedy1*, que toma como entrada las estaciones, el número de furgonetas disponibles y el número máximo de bicicletas que puede llevar una furgoneta. La función identifica las estaciones con exceso y déficit de bicicletas y asigna furgonetas para trasladar las bicicletas, maximizando así el beneficio.

Coste: $O(estaciones\ log\ estaciones)$

4.3 Solución 3: Minimización del Costo de Transporte

La tercera estrategia, *Solucio_greedy3*, se centra en minimizar el costo de transporte. Prioriza la reducción de la distancia de transporte de las bicicletas, teniendo en cuenta la necesidad de satisfacer la demanda de bicicletas en las estaciones. Se calcula la distancia entre las estaciones con exceso y déficit de bicicletas, y se asignan furgonetas para realizar los traslados más cortos primero, asegurando así que el costo de transporte sea mínimo.

Se implementa la función *Solucio_greedy3*, que también toma las estaciones, el número de furgonetas y el número máximo de bicicletas como entrada. Esta función calcula la distancia entre las estaciones con exceso y déficit de bicicletas y asigna furgonetas para minimizar la distancia total de transporte, reduciendo así el costo de transporte.

Coste: $O(furgos * estaciones^2 \, log \, estaciones)$

5. FUNCIONES HEURÍSTICAS

5.1 Heurístico de ingresos:

Este heurístico busca maximizar la cantidad de ingresos derivados de transportar bicis penalizando el hecho de sacar bicis de una estación de manera que la aleje de su demanda. Para esto recorremos la lista de instancias de la clase furgo en la cual tenemos toda la información de de que estación sale, a que estaciones se dirige y con cuantas bicicletas irá a una y otra estación si es que se dirige a dos destinos distintos.

Este heurístico dispone de una penalización por sacar bicis de una estación que no llegue a su demanda, pero no nos es necesario que tenga penalizaciones relacionadas con el hecho de que se salga del espacio de soluciones porque al generar las acciones nos aseguramos, a través de condiciones, de que esto no pueda ser posible. Además, no tenemos la necesidad de colocar pesos ya que únicamente buscamos maximizar un único parámetro: los ingresos por la cantidad de bicis transportadas.

Coste del heurístico de ingresos: $O\ (maxfurgos)$

Esto es porque lo único que recorremos es nuestra lista de furgos asignadas que, en el peor de los casos, sería de longitud igual al máximo número de furgos de las que disponemos.

5.2 Heurístico de beneficio:

Este heurístico busca maximizar la diferencia de los ingresos derivados de transportar bicis (penalizando el hecho de sacar bicis de una estación de manera que la aleje de su demanda) y el coste de su transporte de acuerdo a la fórmula: (((número_de_bicis*9) div 10) * Km_recorridos) donde div es la división entera . Para esto nos aprovechamos del heurístico de ingresos, llamándolo para que nos proporcione los ingresos, y calculamos la distancia recorrida mediante un nuevo recorrido por la lista de furgos asignadas calculando el coste de las bicicletas que transportan por los kilómetros que recorren obteniendo el coste. Ahora simplemente hacemos la resta de los ingresos y el coste obteniendo el beneficio

Coste del heurístico de beneficio: $O\ (maxfurgos^2)$

Esto es porque lo único que hacemos es recorrer nuestra lista de furgos asignadas

dos veces, una para calcular los ingresos y otra para calcular el coste.

6. EXPERIMENTO

6.1 Experimento 1

Determinar qué conjunto de operadores da mejores resultados.

Conjunto 1:

Con todos los operadores planteados en 3.OPERADORES, con la condición de que que *crea_furgo* solo se aplica creando una furgoneta que coja 1 bici.

Conjunto 2:

Con todos los operadores planteados en 3.OPERADORES , con la condición de que que *crea_furgo* pueda coger un rango entre las bicicletas permitidas.

Conjunto 3:

El mismo conjunto de operadores que "conjunto 2", pero quitando el operador "incrementar_num_bici".

Nos hemos asegurado que cada conjunto de operadores puedan recorrer todo el espacio de soluciones.

Observación: Existen variaciones en la efectividad de diferentes operadores para resolver problemas específicos.
Planteamiento: Comparar el rendimiento de tres distintos conjuntos de operadores en términos de número de pasos, tiempo y beneficio total.
Hipótesis:

- **Hipótesis Nula (H0):** No hay diferencias significativas entre los operadores en términos de número de pasos, tiempo y beneficio total.
- **Hipótesis Alternativa (H1):** Existen diferencias significativas entre al menos dos de los operadores en uno o más de los parámetros medidos.

6.1.1 Método

La replicabilidad y consistencia son esenciales en la ejecución de experimentos científicos. Para garantizar que cada experimento se realiza bajo las mismas condiciones, seguimos un protocolo estricto.

6.1.1.1 Preparación del entorno experimental:

Consistencia: Aseguramos un entorno controlado y estandarizado para cada réplica del experimento, minimizando las variaciones externas que puedan afectar los resultados.

6.1.1.2 Selección de semillas

Elegimos 15 semillas aleatorias para garantizar la diversidad y la aleatoriedad en las inicializaciones, lo que nos permitirá una visión más generalizada del rendimiento de los operadores.

6.1.1.3 Ejecución de experimentos:

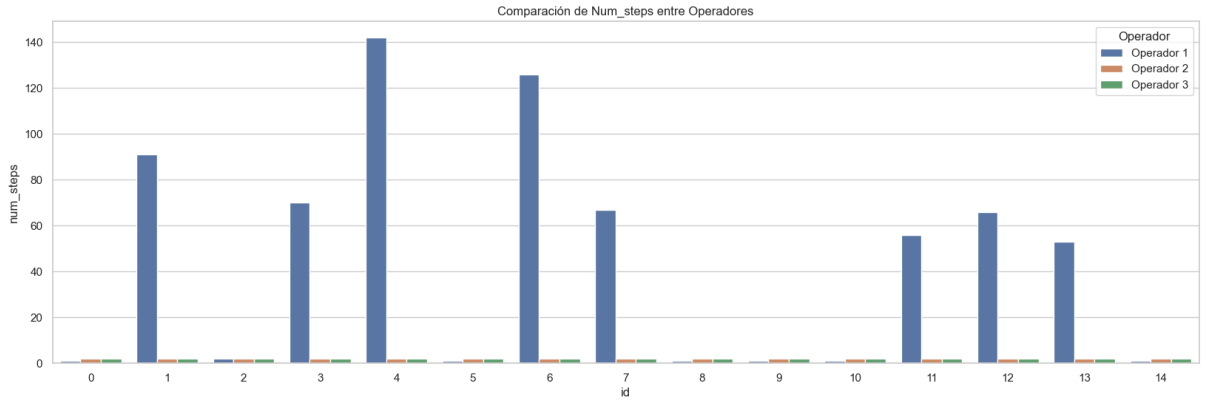
- Con cada semilla, realizamos un experimento para cada uno de los tres conjuntos de operadores, totalizando 45 experimentos.
- Utilizamos el algoritmo de Hill Climbing con inicialización avariciosa para evaluar la eficiencia y eficacia de cada conjunto de operadores.
- Utilizaremos el heurístico de beneficio total.
- Y las mismas condiciones que el experimento 1.

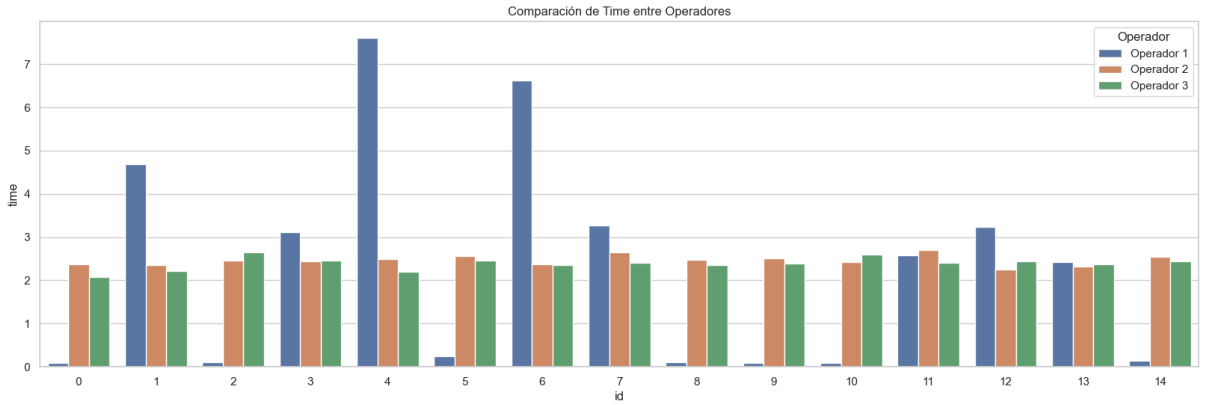
6.1.1.4 Parámetros de medición:

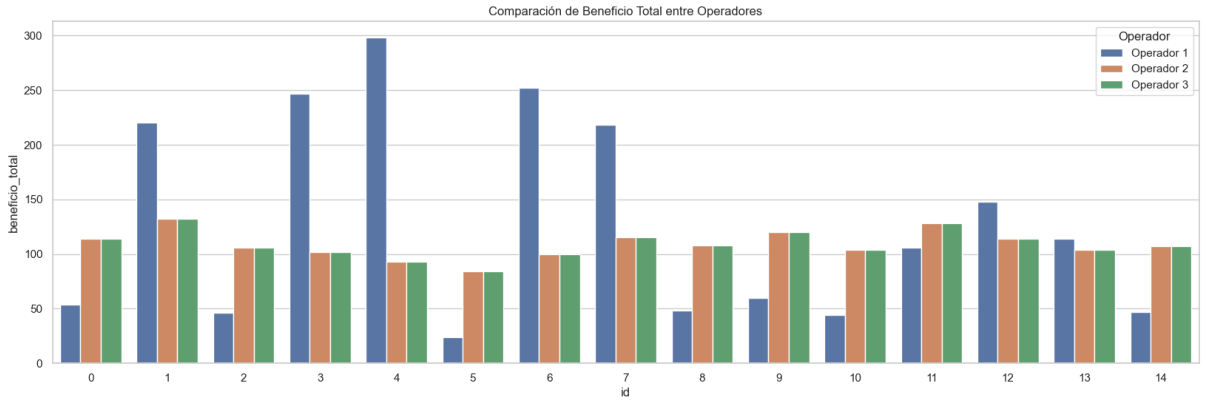
- **num_steps (Pasos del Algoritmo):** Registramos cuántos pasos necesita el algoritmo para converger a una solución. Esto nos dará una idea de su rapidez y, posiblemente, de su capacidad para evitar mínimos locales.
- **time (Tiempo de Ejecución):** Medimos el tiempo que tarda el algoritmo en alcanzar la solución final. Este es un indicador directo de la eficiencia del operador.
- **beneficio_total (Calidad de la Solución):** Evaluamos el beneficio de la solución final proporcionada por el algoritmo. Este valor nos dará una perspectiva sobre la calidad de las soluciones obtenidas.

6.1.2 Resultados del experimento

6.1.2.1 Plots

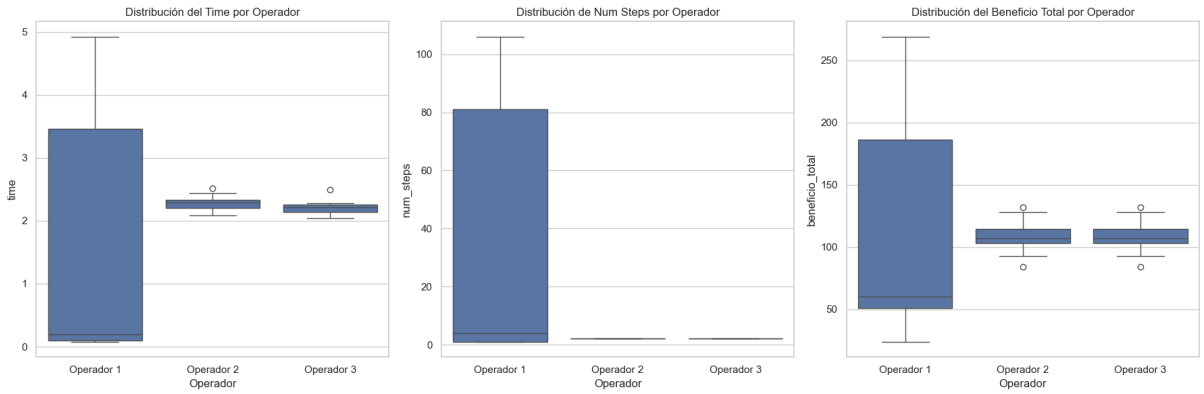




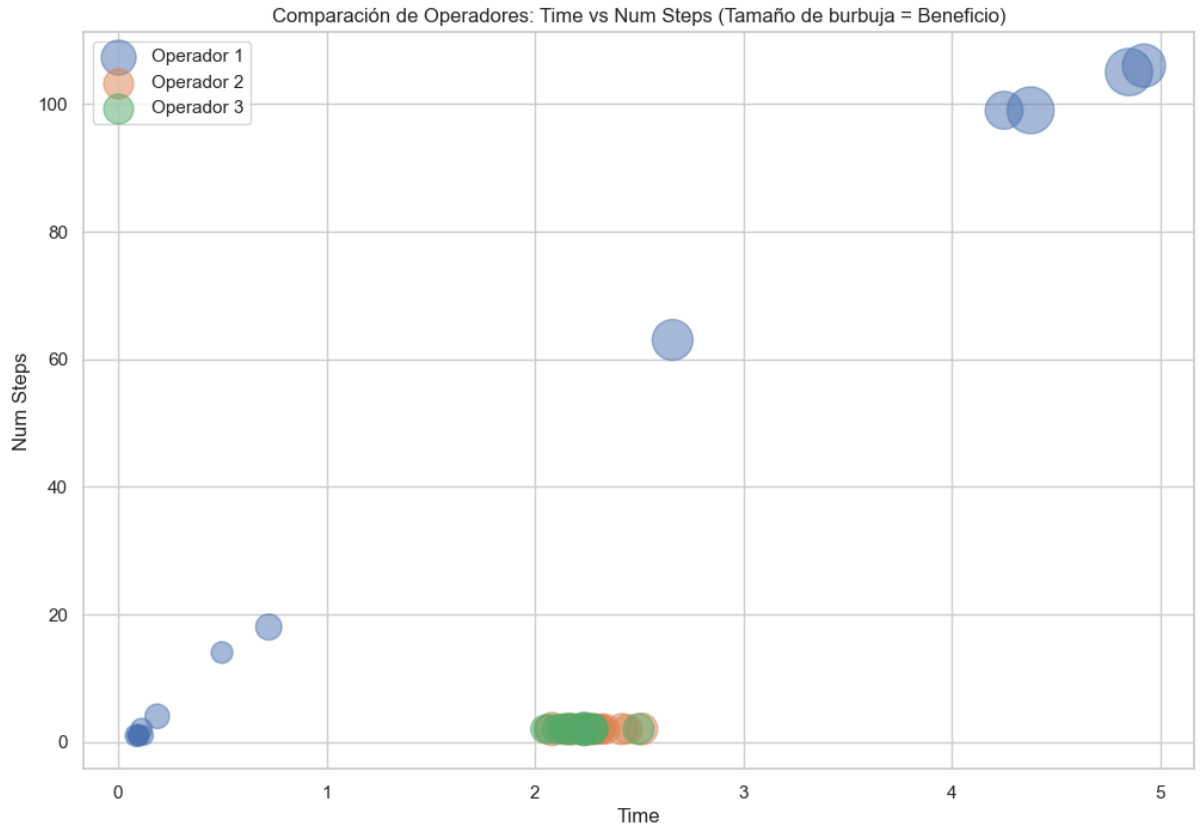


En las tres gráficas iniciales, se destaca claramente que el número de pasos (num_steps) es significativamente mayor para el conjunto de operadores 1, lo que indica una diferencia notable en su desempeño comparado con los otros dos conjuntos. Al comparar los conjuntos de operadores 2 y 3, se observa que sus

resultados en número de pasos y beneficio son muy similares, destacando un poco más rapidez para los operadores 3.



Analizando las gráficas, es evidente que el conjunto de operadores 1 muestra una varianza considerablemente más amplia en sus resultados en comparación con los conjuntos 2 y 3. Esta variabilidad sugiere una inconsistencia en el rendimiento del conjunto de operadores 1, reflejando posiblemente una adaptabilidad diferencial a distintos escenarios o condiciones de prueba.



Esta gráfica revela una tendencia interesante: para el conjunto de operadores 1, existe una correlación casi lineal entre el número de pasos y el tiempo, indicando que un aumento en los pasos se traduce proporcionalmente en un aumento en el

tiempo de ejecución. Sin embargo, esta relación lineal no está presente entre los conjuntos de operadores 2 y 3, donde el tiempo parece no incrementarse tan marcadamente con el aumento de pasos. Esto podría sugerir una mayor eficiencia o un manejo diferente de los procesos por parte de estos dos últimos conjuntos de operadores.

6.1.2.2 T-students

Comparación entre Operador 1 y Operador 2

Número de Pasos (num_steps):

Valor t: 2.775

Valor p: 0.0097

Conclusión: Hay una diferencia estadísticamente significativa en el número de pasos entre el Operador 1 y el Operador 2, con un valor p menor que 0.05. Esto sugiere que uno de los operadores es más eficiente en términos de número de pasos.

Tiempo (time):

Valor t: -1.416

Valor p: 0.1679

Conclusión: No hay una diferencia estadísticamente significativa en el tiempo entre el Operador 1 y el Operador 2, ya que el valor p es mayor que 0.05.

Beneficio Total (beneficio_total):

Valor t: 0.125

Valor p: 0.9013

Conclusión: No hay una diferencia estadísticamente significativa en el beneficio total entre el Operador 1 y el Operador 2, con un valor p mucho mayor que 0.05.

Comparación entre Operador 1 y Operador 3

Número de Pasos (num_steps):

Mismos resultados y conclusiones que en la comparación entre Operador 1 y Operador 2.

Tiempo (time):

Valor t: -1.271

Valor p: 0.2141

Conclusión: Al igual que con el Operador 2, no hay una diferencia significativa en el tiempo entre el Operador 1 y el Operador 3.

Beneficio Total (beneficio_total):

Mismos resultados y conclusiones que en la comparación entre Operador 1 y Operador 2.

Comparación entre Operador 3 y Operador 2

Número de Pasos (num_steps):

Valor t: NaN

Valor p: NaN

Conclusión: Esta prueba no se pudo realizar correctamente. Puede ser debido a datos idénticos o errores en los datos.

Tiempo (time):

Valor t: -1.853

Valor p: 0.0744

Conclusión: No hay una diferencia estadísticamente significativa en el tiempo entre el Operador 3 y el Operador 2, aunque el valor p está cerca del umbral de 0.05.

Beneficio Total (beneficio_total):

Valor t: 0.0

Valor p: 1.0

Conclusión: No hay ninguna diferencia en el beneficio total entre el Operador 3 y el Operador 2; los resultados son idénticos.

6.1.3 Expectativas vs Realidad

Inicialmente, creíamos que habría diferencias notables entre todos los conjuntos de operadores. Al formular los diferentes conjuntos, supusimos que el conjunto de operadores 3 no encontraría la solución más óptima, dado que carece de la función

"incrementar_num_bici". Sin embargo, la realidad nos sorprendió, ya que este conjunto empató en desempeño con el conjunto 2.

Teníamos la expectativa de que habría una mayor diferencia temporal entre los conjuntos de operadores 2 y 3, pero esto no se materializó en los resultados.

Por otro lado, anticipamos que el conjunto de operadores 1 sería el que presentaría peores resultados, y esta suposición se confirmó en la práctica.

6.1.4 Conclusiones

Definitivamente, existe una marcada distinción entre el conjunto de operadores 1 y los demás. Sin embargo, no se observa una diferencia significativa entre los conjuntos 2 y 3. A pesar de que ambos logran el mismo número de pasos y obtienen idéntico beneficio, el conjunto de operadores 3 muestra una ligera ventaja en términos de velocidad en general.

Con base en estos resultados, concluimos que el conjunto de operadores 3 es el más óptimo.

Se confirma la Hipótesis Alternativa (H1).

6.1.5 Comentarios adicionales

Parece que los operadores 2 y 3 obtienen el mismo resultado debiso a su beneficio idéntico, pero no es así.

```
main_instance = Main(model, maxim_furgonetas, numero_estaciones,
numero_bicis, semilla = 5 ,operadors=2 , heuristic=heuris,
initial_state=3)
Trasllats:['Furgo sale de (3500, 3500) y va a (2900, 3300) con 2',
'Furgo sale de (3100, 4800) y va a (3800, 5100) con 3', 'Furgo sale de
(4300, 4500) y va a (4200, 3600) con 5', 'Furgo sale de (3500, 7300) y
va a (2500, 7100) con 13', 'Furgo sale de (7400, 3800) y va a (7400,
4900) con 1', 'Furgo sale de (2000, 1400) y va a (3100, 4800) con 30'],
beneficio_total: 84

main_instance = Main(model, maxim_furgonetas, numero_estaciones,
numero_bicis, semilla = 5 ,operadors=3 , heuristic=heuris,
```

```
initial_state=3)
Trasllats:['Furgo sale de (3500, 3500) y va a (2900, 3300) con 2',
'Furgo sale de (3100, 4800) y va a (3800, 5100) con 3', 'Furgo sale de
(4300, 4500) y va a (4200, 3600) con 5', 'Furgo sale de (3500, 7300) y
va a (2500, 7100) con 13', 'Furgo sale de (7400, 3800) y va a (7400,
4900) con 1', 'Furgo sale de (8300, 600) y va a (7400, 3800) con 30'],
beneficio_total: 84
```

6.2 Experimento 2

¿Qué estrategia de generación de la solución inicial da mejores resultados?

Compararemos tres estrategias de generación inicial: la solución vacía y dos soluciones greedys

6.2.1 Condiciones

Elegiremos 10 semillas aleatorias, una para cada réplica, y usaremos el algoritmo hill climbing con el heurístico de ingresos.

Mediremos num_step(cantidad de pasos antes de solución), time(tiempo de ejecución) y el beneficio_total (valor de los ingresos).

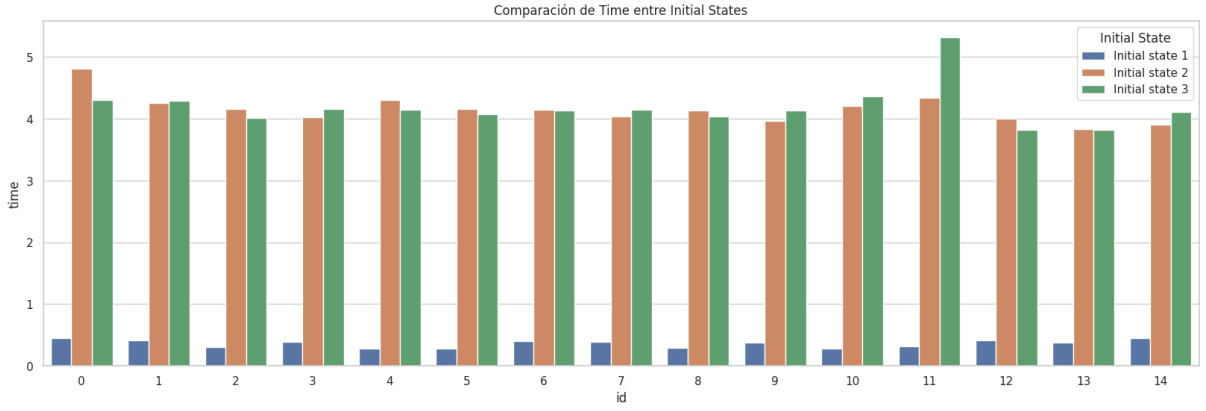
Num_step, sin embargo, no será informativo en este experimento y no la tendremos en consideración.

6.3.2 Hipótesis

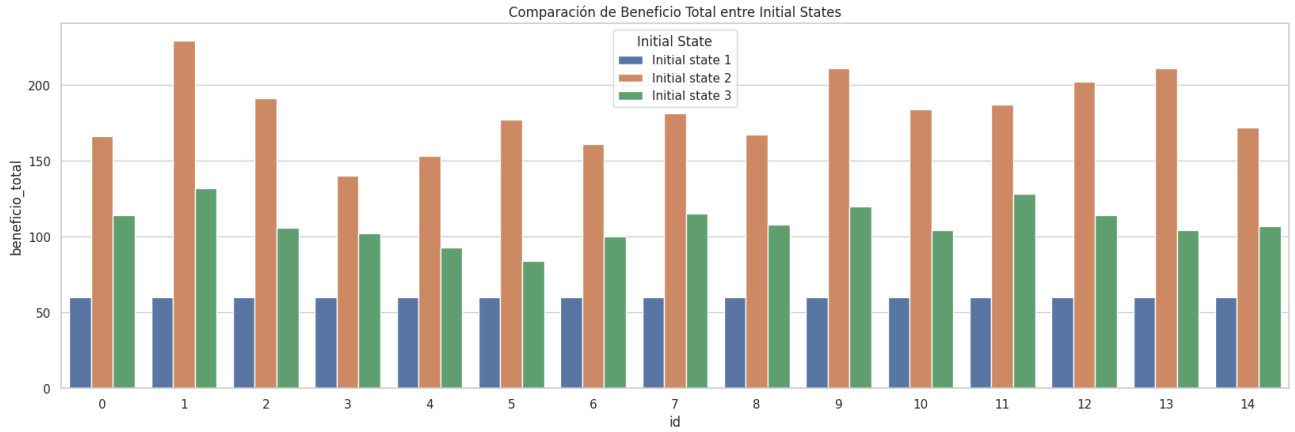
- **Hipótesis Nula (H0):** la solución inicial no influye en la solución o el tiempo de ejecución del programa
- **Hipótesis Alternativa (H1):** hay una solución inicial que nos proporciona un mejor beneficio.

6.2.3 Resultados del experimento

Plot de las soluciones iniciales respecto el Tiempo:



Plot de los ingresos de cada solución inicial



De estos gráficos de barras ya podemos comenzar a extraer observaciones y hacer comparaciones. Podemos observar claramente que la más rapida de las soluciones iniciales es siempre la solución vacía(initial state 1) pero también observamos que esta es, igualmente, la que obtiene el menor ingreso. La solución greedy 1(initial state 2) y 2(initial state 3) vemos que tienen tiempos de ejecuciones muy similares pero que, claramente, la primera de las soluciones greedy obtiene los mejores resultados.

Boxplot de los tiempos de ejecución, número de pasos y ingresos de cada solución inicial:

Podemos corroborar, ignorando el boxplot de los pasos por no ser informativo, las observaciones anteriores: la segunda y tercera solución tardan prácticamente lo mismo mientras la primera es la más rápida y la segunda obtiene, claramente, la mejor solución.

6.2.4 Expectativas vs Realidad

Esperábamos que nuestra tercera solución fuese la que mejor beneficio proporcionaba mientras resultó, claramente, que era nuestro segundo generador de estados iniciales la que mejores soluciones proporcionaba.

6.2.5 Conclusiones

Decidimos usar para los siguientes experimentos la segunda solución inicial por ser la mejor claramente, ya que la diferencia de tiempo es tan pequeña que es despreciable. En consecuencia, podemos concluir que nuestra primera hipótesis es falsa y nuestra segunda hipótesis es cierta

6.3 Experimento 3

¿Qué parámetros K y λ permiten al algoritmo simulated annealing encontrar las mejores soluciones ?

6.3.1 Condiciones

Probaremos todas las posibles combinaciones de los valores K=1,5,25 y 40 y λ=1,0.1,0.01

Haremos tres réplicas, hechas con tres semillas distintas, para cada posible combinación de valores y con estas calcularemos el promedio de sus diferentes tiempos de ejecución, número de pasos e ingresos.

Esto lo haremos con un límite establecido en 10000 y la solución inicial concluida anteriormente y usaremos el algoritmo simulated annealing con el heurístico de ingresos.

Mediremos num_step(cantidad de pasos antes de solución), time(tiempo de ejecución y el beneficio_total (valor de los ingresos)

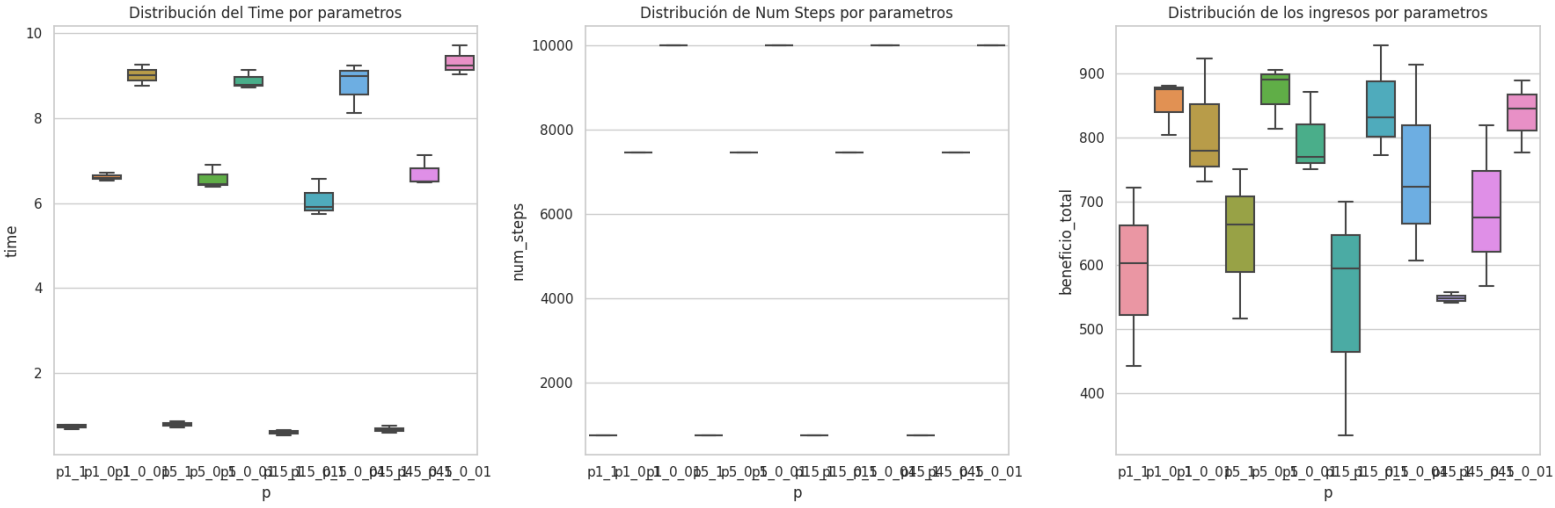
6.3.2 Hipótesis

- **Hipótesis Nula (H0):** los parámetros no provocan cambios evidentes en el algoritmo
- **Hipótesis Alternativa (H1):** hay una combinación de parámetros que nos proporciona los mejores resultados.

6.3.3 Resultados del experimento

Boxplot de la cantidad de pasos, tiempo de ejecución e ingresos con distintas combinaciones de k y lambda:

El plot puede agruparse de tres en tres, las tres primeras cajas son las que tienen



K=1 y la lambda va reduciéndose. De igual forma podemos agrupar el resto de cajas(información útil puesto que no se ven adecuadamente los valores de los parámetros en el eje x)

De estos tres plots podemos extraer mucha información. Podemos observar que los algoritmos que más pasos hacen y más tiempo tardan en ejecutar son siempre los mismos: aquellos con lambda de valor 0.01, lo cual tiene sentido teniendo en cuenta que lambda es el valor con el que bajamos la energía, con el que 'enfriamos'.
Los ingresos son otra historia y fluctúan mucho más pero podemos ver que la quinta 'box', que se corresponde con k=5 y lambda=0.1, es la que más ingresos nos proporciona en promedio sin ser de las más lentas ni de las que más pasos necesita, sinó formando parte del rango intermedio.

Cabe también recalcar que, en caso de repetición, los resultados cambian. Esto nos muestra una alta fluctuación entre el beneficio pese a hacer el promedio de una serie de réplicas lo que causa que realmente no podamos estar del todo seguros de los parámetros que escojamos. Pese a esto, podemos observar que k=5 y lambda =

0.1 sigue siendo una elección bastante adecuada que tiende a proporcionarnos un buen beneficio.

6.3.4 Expectativas vs Realidad

Esperábamos un resultado parecido al que puede observarse en el ejemplo del problema de las ciudades de práctica, sin embargo nuestros resultados son mucho más irregulares y los ingresos tienden a ser más altos con K no tan grandes y el menor valor de lambda.

6.3.5 Conclusión:

Los parámetros con los que nos quedamos son k=5 y lambda = 0.1 por ser el que mayor ingreso proporciona en promedio y que, pese a no ser de las combinaciones más rápidas, sigue sin estar entre las más lentas de las combinaciones. Así pues, con esta combinación de parámetros para simulated annealing conseguimos los mejores valores sin hacer el máximo sacrificio en cuanto a tiempo de ejecución o pasos, aunque ciertamente hacemos un sacrificio pues unos 5-6 segundos de tiempo de ejecución para un problema del tamaño del nuestro sigue siendo mucho.

En consecuencia, podemos concluir que nuestra primera hipótesis es falsa y nuestra segunda hipótesis es cierta

6.4 Experimento 4

¿Cómo evoluciona el tiempo de ejecución para hallar la solución en función del número de estaciones, furgonetas y bicicletas?

6.4.1 Condiciones y planteamientos

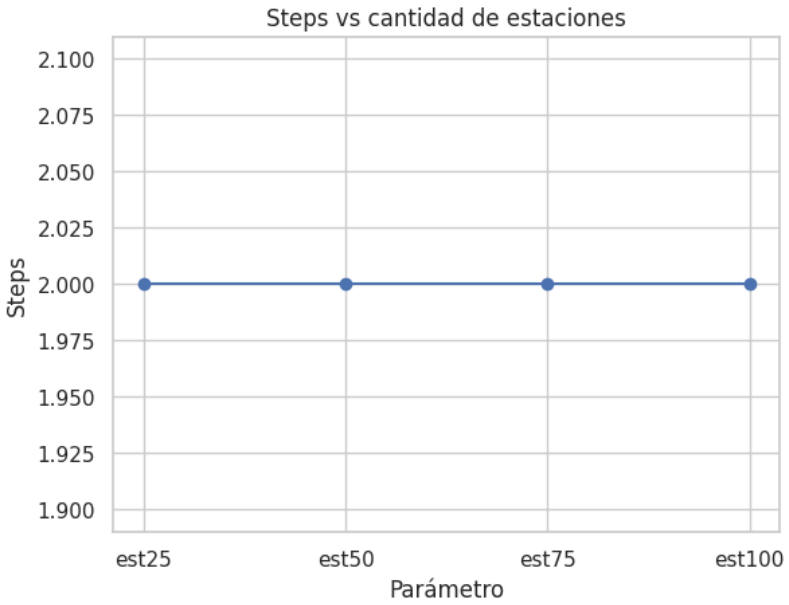
Usaremos una proporción 1 a 50 entre estaciones y bicicletas y 1 a 5 entre furgonetas y estaciones. Empezaremos con 25 estaciones e iremos aumentando la cantidad de 25 en 25 hasta las 200 estaciones. Para esto usaremos una misma semilla y el algoritmo de Hill Climbing con el heurístico de ingresos y la segunda solución inicial.

6.4.2 Hipótesis

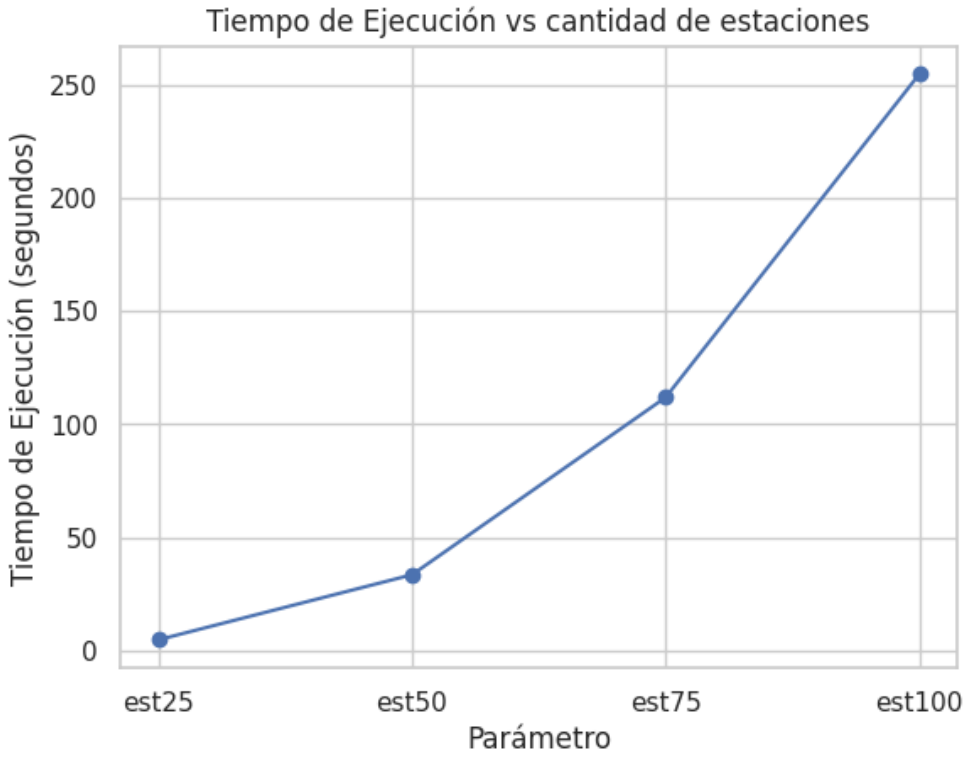
- **Hipótesis Nula (H0):** El incremento en el número de estaciones no provoca un cambio real en el tiempo
- **Hipótesis Alternativa (H1):** A más estaciones más tiempo de ejecución

6.4.3 Resultados del experimento

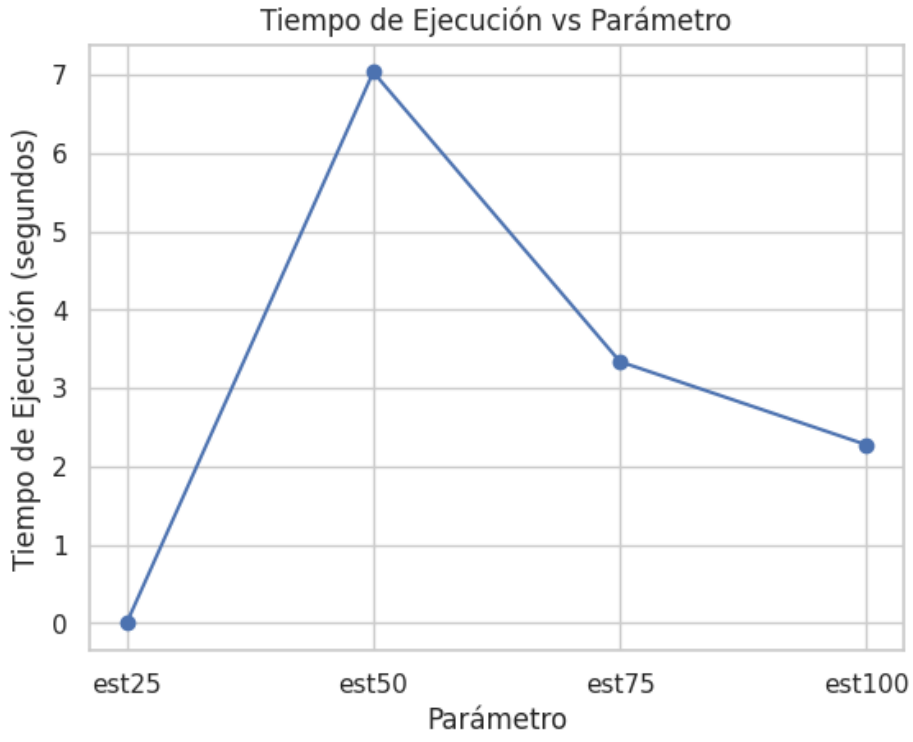
Plot de la cantidad de pasos respecto la cantidad de estaciones:



Plot del tiempo de ejecución respecto la cantidad de estaciones:



Plot del incremento del tiempo de ejecución respecto al tiempo de ejecución anterior:



Estos plots nos proporcionan mucha información. Podemos observar que la cantidad de estaciones(y la correspondiente cantidad de furgonetas y bicicletas) parece no afectar al número de pasos que hillclimbing dará antes de hallar la mejor solución(dentro del heurístico que estamos usando, el de ingresos).
En el plot del tiempo respecto a la cantidad de estaciones vemos un claro aumento, un aumento muy importante, del tiempo de ejecución tardando la cuarta ejecución ya más de 250 segundos mientras la primera apenas tardaba 1 segundo. Esto nos indica que claramente nuestro código escala muy mal a medida que aumentamos la cantidad de estaciones.
En cuanto al último plot, este nos indica el aumento del tiempo de ejecución de una cantidad de estaciones respecto la cantidad de estaciones anterior y, como puede observarse, tenemos un gran pico donde el tiempo de ejecución se multiplica por 7 y después una clara tendencia a la baja y la estabilización del incremento del tiempo de ejecución respecto a la cantidad de estaciones.

6.4.4 Expectativas vs Realidad

Nuestras expectativas eran que a medida que añadíamos complejidad al problema, a medida que añadíamos estaciones, el tiempo de ejecución iría aumentando y así ha sido, y de forma muy notable.

6.4.5 Conclusiones:

Podemos ver que nuestro código escala considerablemente mal aunque parezca que este incremento comience a estabilizarse ya que un incremento de 2 respecto a 100 segundos es ya un valor de tiempo de ejecución muy elevado.
En consecuencia, podemos concluir que nuestra primera hipótesis es falsa y nuestra segunda hipótesis es cierta

6.5 Experimento 5

Comparación de Técnicas: Hill Climbing vs. Simulated Annealing en Heurísticas Diferentes

Observación: Hill Climbing y Simulated Annealing son técnicas de optimización comúnmente utilizadas, pero su rendimiento puede variar significativamente según la heurística y el problema abordado.

Planteamiento: Estimar y comparar la diferencia en el beneficio obtenido, la distancia total recorrida y el tiempo de ejecución al aplicar Hill Climbing y Simulated Annealing usando dos heurísticas diferentes.

Hipótesis:

- **Hipótesis Nula (H0):** No hay diferencias significativas entre Hill Climbing y Simulated Annealing en términos de beneficio, distancia recorrida y tiempo de ejecución para las dos heurísticas implementadas.
- **Hipótesis Alternativa (H1):** Existen diferencias significativas entre Hill Climbing y Simulated Annealing.

6.5.1 Método

La replicabilidad y consistencia son esenciales en la ejecución de experimentos científicos. Para garantizar que cada experimento se realiza bajo las mismas condiciones, seguimos un protocolo estricto.

6.5.1.1 Preparación del entorno experimental:

Uniformidad: Mantenemos un entorno controlado y constante para cada iteración de los algoritmos, asegurando que las únicas variables sean las técnicas y las heurísticas utilizadas.
Variables controladas: Las mismas condiciones que el experimento 1, excepto por los heurísticos que usaremos 1 y el 2 en el 5.FUNCIONES HEURÍSTICAS . Para los parametros de Hill Climbing hemos usado unos predeterminados (k:int=1,lam:int=0.01,limit:int=20000) ya que no se especificaba el uso de parámetros obtenidos del apartado 6.3 Experimento 3.

6.5.1.2 Selección de de semillas

Seleccionamos un conjunto de 10 semillas, garantizando una representatividad adecuada de diversos escenarios y complejidades.

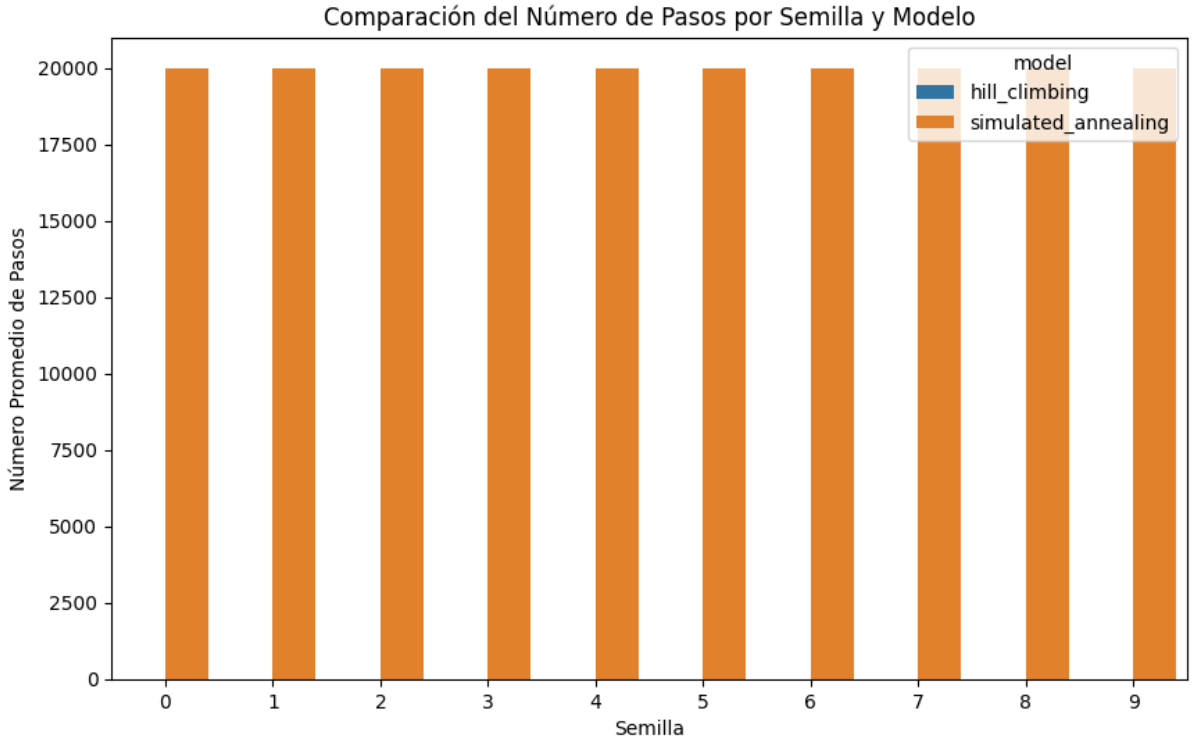
6.5.1.3 Ejecución de experimentos:

Cada técnica (Hill Climbing y Simulated Annealing) se aplica a las mismas instancias utilizando ambas heurísticas. Esto resulta en un total de 40 experimentos (2 técnicas × 2 heurísticas × 10 instancias).

6.5.1.4 Parámetros de medición:

beneficio_obtenido (Calidad de la Solución): Evalúa el beneficio neto logrado por cada algoritmo, reflejando la efectividad de la heurística y la técnica de optimización.
distancia_total (Eficiencia en Distancia): Mide la distancia total recorrida por la solución propuesta, indicando la eficiencia en términos de coste de desplazamiento.
tiempo_ejecución (Eficiencia Temporal): Registra el tiempo que cada algoritmo tarda en alcanzar una solución, proporcionando una medida directa de su eficiencia computacional.

6.5.2 Resultados del experimento



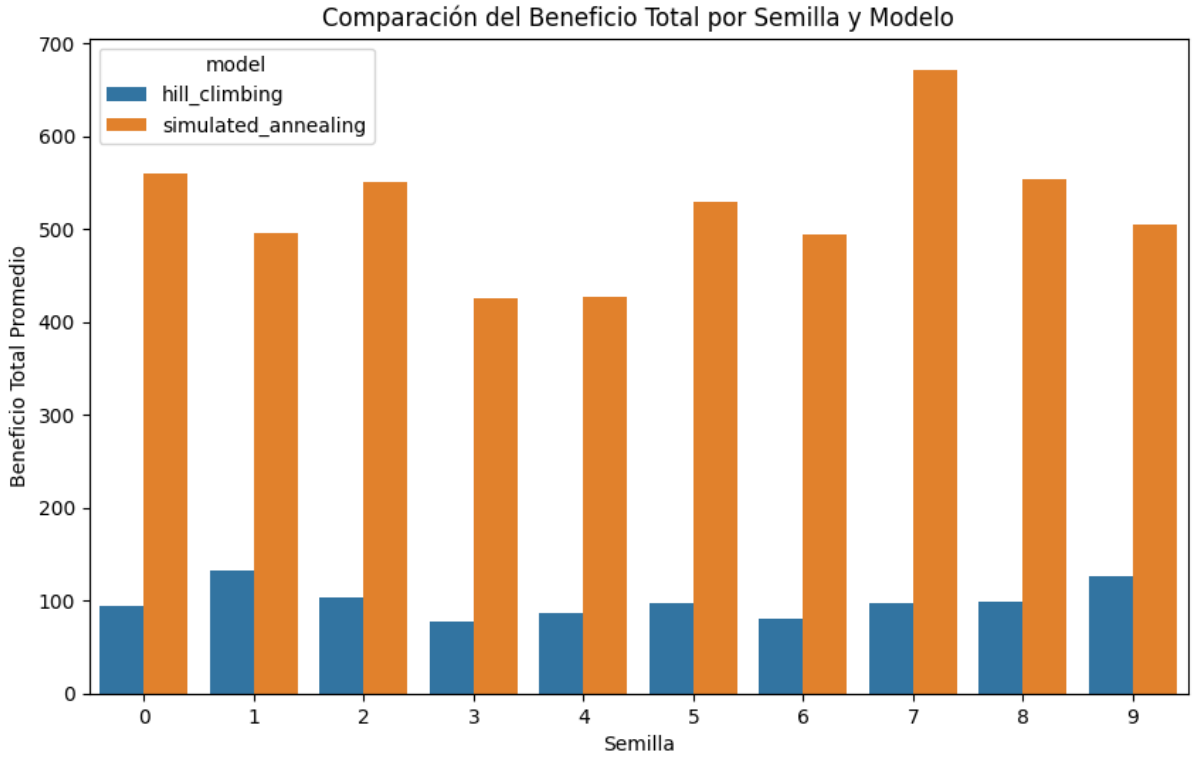
A primera vista, las barras de "hill_climbing" y "simulated_annealing" parecen tener alturas similares para cada semilla, lo que indica que ambos modelos tienen un número similar de pasos en estos experimentos.

Como se mencionó, el número de pasos para "hill_climbing" ronda entre 5-10, lo que es casi insignificante en comparación con la escala del gráfico que llega hasta 20,000. Dado que el gráfico está escalado a este alto valor, es difícil distinguir visualmente cualquier variación menor en el número de pasos.

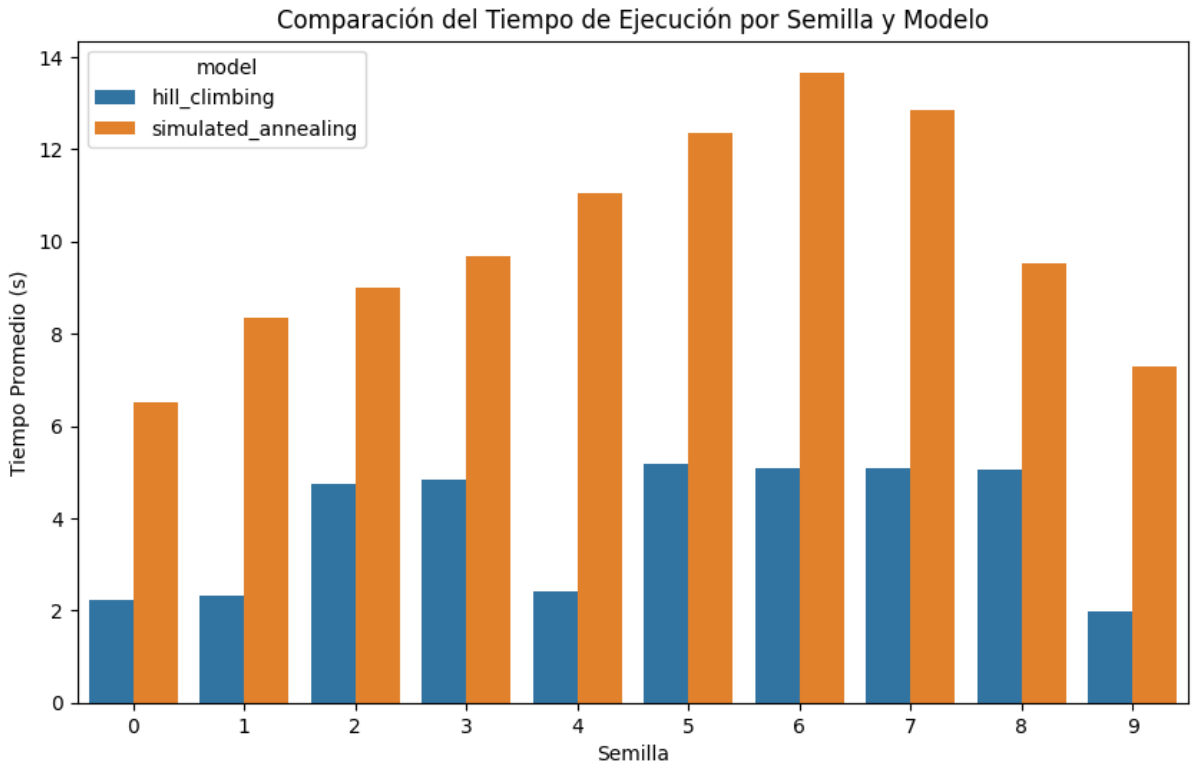
Es notable que "hill_climbing" alcanza constantemente los 20,000 pasos, que es el número máximo de pasos en este contexto. Esto indica que "hill_climbing" podría no estar encontrando una solución óptima antes de alcanzar el límite de pasos o que está atrapado en un mínimo local. El hecho de que alcance este límite en cada semilla sugiere que puede haber problemas con la configuración o la implementación de este algoritmo en este contexto específico.

Si "hill_climbing" llega al número máximo de pasos debido a "simulated_annealing", podría ser debido a la configuración de la temperatura (k). Una temperatura inicial

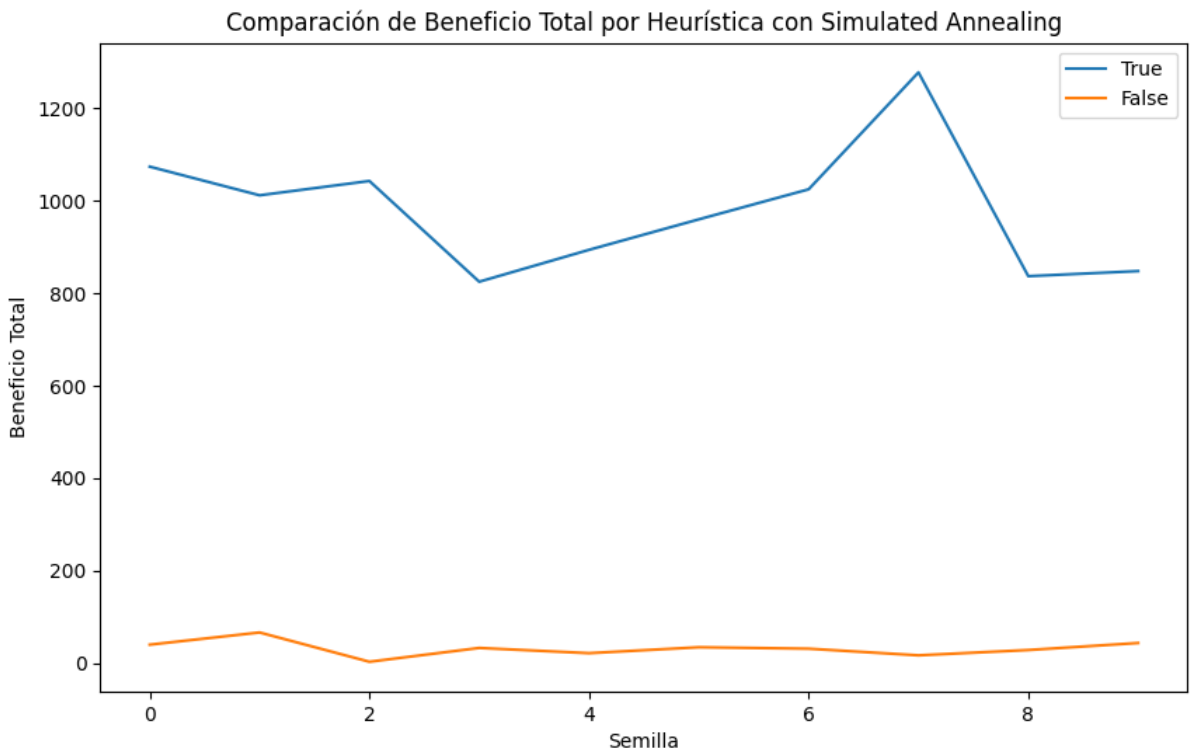
muy alta en el algoritmo de recocido simulado permite una mayor exploración al principio, pero si es demasiado alta, puede resultar en una búsqueda menos dirigida y, por lo tanto, más pasos. Además, si el valor de lambda disminuye muy rápidamente, la probabilidad de aceptar soluciones peores se reduce drásticamente, lo que puede hacer que el algoritmo se estanque.



Este gráfico ilustra una comparativa entre dos modelos, "hill_climbing" y "simulated_annealing", en relación con el beneficio total promedio obtenido por cada semilla. Es evidente que el modelo "simulated_annealing" ostenta una superioridad notoria en casi todas las semillas, logrando beneficios que se acercan o superan los 500 puntos. En contraste, "hill_climbing" muestra un rendimiento considerablemente más bajo, con beneficios que no superan los 200 puntos en la mayoría de las semillas. La marcada diferencia entre ambos modelos sugiere que "simulated_annealing" podría estar mejor adaptado para encontrar soluciones óptimas o que su mecanismo de enfriamiento y aceptación de soluciones es más efectivo en este contexto específico que el enfoque directo de "hill_climbing".



La gráfica compara el tiempo de ejecución promedio de dos modelos, "hill_climbing" y "simulated_annealing", para diversas semillas. Se observa que el modelo "simulated_annealing" generalmente requiere más tiempo, con tiempos de ejecución que oscilan entre los 8 y 14 segundos, lo que podría atribuirse a su proceso iterativo de enfriamiento y búsqueda de soluciones óptimas. Por otro lado, "hill_climbing" muestra un tiempo de ejecución más breve en la mayoría de las semillas, fluctuando mayormente entre 4 y 7 segundos. Esta diferencia en tiempos sugiere que, aunque "simulated_annealing" puede ofrecer un mejor rendimiento en términos de beneficio, como se vio en el gráfico anterior, este rendimiento superior tiene el costo de un tiempo de procesamiento más prolongado en comparación con "hill_climbing".



La gráfica muestra una comparación del beneficio total obtenido por dos heurísticas distintas ("True": sin coste transporte y "False": con coste de transporte) utilizando Simulated Annealing en función de diferentes semillas. Se puede observar que la heurística "True" consistentemente logra un beneficio significativamente mayor que la heurística "False" a lo largo de todas las semillas probadas. El rendimiento de la heurística "False" permanece relativamente constante y bajo, mientras que la heurística "True" muestra algunas fluctuaciones, alcanzando su punto máximo alrededor de la semilla 7.

6.5.3 Expectativas vs Realidad

Al comienzo de nuestra investigación, teníamos la expectativa de que el algoritmo "simulated_annealing" alcanzaría el número máximo de parámetros, apoyándonos en las conclusiones obtenidas del EXPERIMENTO 3. Esta anticipación no solo se ha confirmado, sino que también ha reforzado la "Hipótesis Alternativa (H1)": Existen diferencias significativas entre Hill Climbing y Simulated Annealing.

Si bien era previsible que "simulated_annealing" arrojara resultados más destacados, debido a su capacidad de recorrer de manera más extensa los espacios de soluciones en comparación con "hill_climbing", lo realmente sorprendente fue el tiempo de ejecución. A pesar de su profunda búsqueda en el espacio de soluciones, el tiempo que empleó "simulated_annealing" no superó de forma considerable al de "hill_climbing".

6.5.4 Comentarios adicionales

Considerando los resultados obtenidos y las dinámicas de los algoritmos evaluados, surgen algunas reflexiones adicionales. Una de las consideraciones más relevantes es la elección de métricas para evaluar el desempeño de "simulated_annealing". Si bien las métricas utilizadas ofrecieron una perspectiva clara en términos generales, es posible que la adopción de indicadores diferentes o adicionales hubiera permitido una comprensión más profunda y específica del comportamiento y eficiencia de "simulated_annealing". Tal vez, al diseñar métricas personalizadas para este algoritmo, podríamos haber obtenido insights más granulares sobre sus características únicas y sobre cómo optimizar aún más su rendimiento en futuras implementaciones.

6.6 Experimento 6

6.6.1 Condiciones

Observación: La cantidad de furgonetas utilizadas en el sistema puede afectar en el calculo.

Planteamiento: Determinar el número óptimo de furgonetas, variando desde 5 hasta el número máximo de estaciones (25) de dos en dos, para maximizar el beneficio total manteniendo un tiempo de ejecución razonable.

Hipótesis

- **Hipótesis Nula (H0):** El incremento en el número de furgonetas hasta el número de estaciones no produce mejoras significativas en el beneficio total más allá de un cierto punto.
- **Hipótesis Alternativa (H1):** Existe un número óptimo de furgonetas, que puede ser menor al número total de estaciones, que maximiza el beneficio total.

6.6.2 Método

Para garantizar la replicabilidad y consistencia, seguimos un protocolo riguroso:

6.6.2.1 Preparación del entorno experimental

Consistencia: Se asegura un entorno controlado y estandarizado.
Variables controladas: Las mismas condiciones que el experimento 1, excepto de por el máximo número de furgos.

6.6.2.2 Selección de semillas

Aleatoriedad: Se utilizan 10 semillas distintas para garantizar variabilidad en las condiciones iniciales.

6.6.2.3 Ejecución de experimentos

Procedimiento: Con cada semilla, se prueba cada cantidad de furgonetas desde 1 hasta el número máximo de estaciones (25).
Número de experimentos: Se realizarán 10 (semillas) × 10 (número de furgonetas) = 100 experimentos.

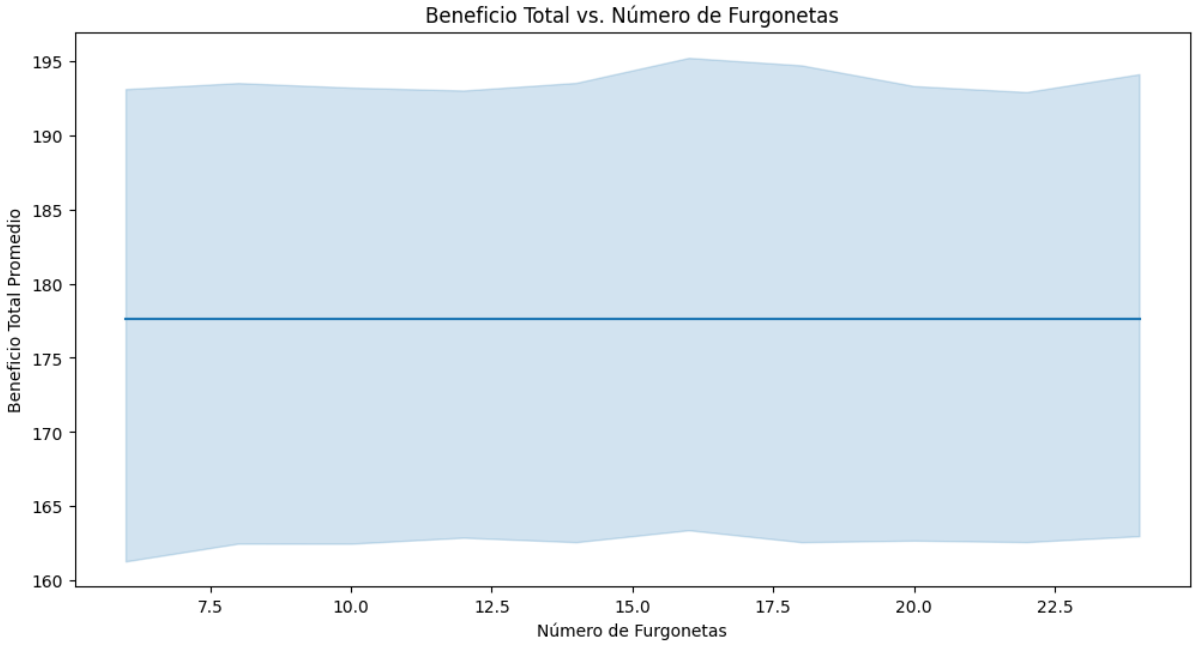
6.6.2.4 Parámetros de medición

time (Tiempo de Ejecución): Medimos el tiempo que tarda el algoritmo en alcanzar la solución final. Este es un indicador directo de la eficiencia del operador.

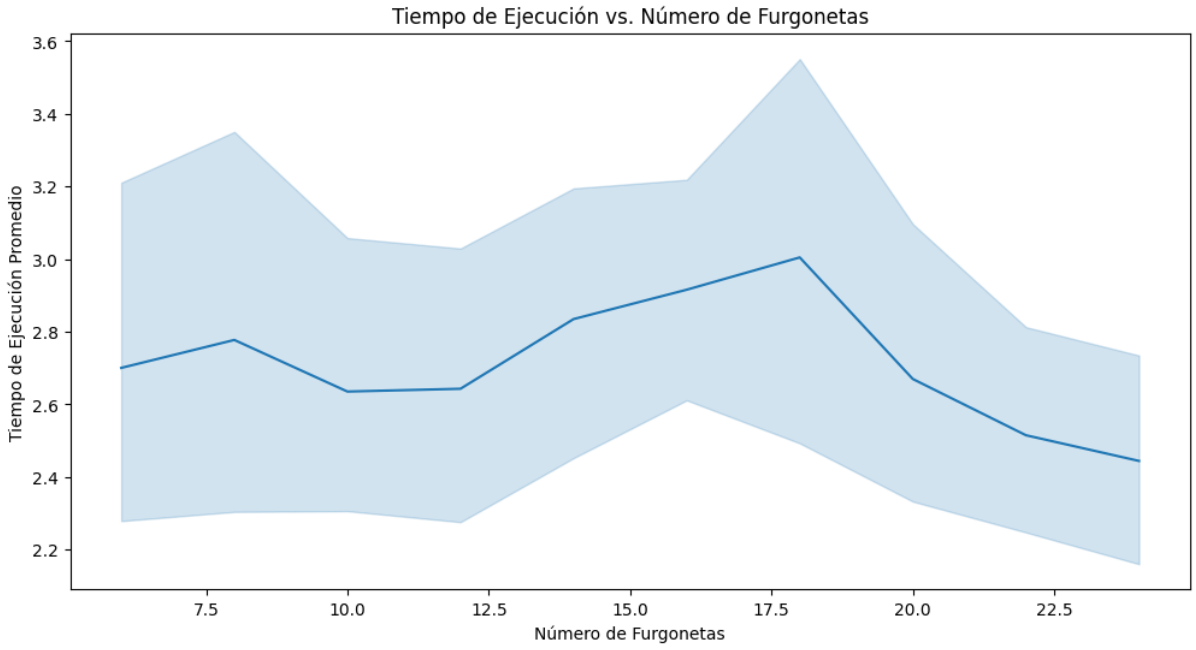
beneficio_total (Calidad de la Solución): Evaluamos el beneficio de la solución final proporcionada por el algoritmo. Este valor nos dará una perspectiva sobre la calidad de las soluciones obtenidas.

6.6.3 Resultados del experimento

6.6.3.1 Plots



Se puede observar en la gráfica "Beneficio Total vs. Número de Furgonetas" que a medida que incrementamos el número máximo de furgonetas, el beneficio total promedio no muestra una tendencia ascendente clara. Aunque hay variaciones, el beneficio se mantiene relativamente estable. Es posible que el número mínimo de furgonetas, como 5, ya esté cerca del óptimo en términos de beneficio, sugiriendo que furgonetas adicionales no aportan un beneficio significativo.



En la gráfica "Tiempo de Ejecución vs. Número de Furgonetas", se aprecia una relación variable entre el tiempo de ejecución promedio y el número de furgonetas. Al comienzo, el tiempo de ejecución parece aumentar ligeramente con el aumento del número de furgonetas, pero luego experimenta altibajos a medida que avanza en el rango de número de furgonetas.

Alrededor de 12 furgonetas, se observa un pico en el tiempo de ejecución, después del cual este tiende a disminuir hasta alrededor de las 17 furgonetas, para luego volver a aumentar. Esto sugiere que puede haber un rango óptimo de furgonetas en el cual el tiempo de ejecución es menor.

6.6.3.2 T-student

Comparación entre Operador 10 y Operador 18 furgonetas máximas

Beneficio Total (beneficio_total):

Valor t: 0.0

Valor p: 1.0

Conclusión: No hay una diferencia estadísticamente significativa en el beneficio total entre Furgonetas 10 y Furgonetas 18, con un valor p de 1.0, lo cual es mucho mayor que 0.05.

Tiempo (time):

Valor t: -1.0405

Valor p: 0.3119

Conclusión: No hay una diferencia estadísticamente significativa en el tiempo entre Furgonetas 10 y Furgonetas 18, ya que el valor p es mayor que 0.05.

6.6.4 Expectativas vs Realidad

Al inicio de nuestro análisis, teníamos ciertas expectativas sobre cómo se comportarían los resultados al variar diferentes factores. Sin embargo, nuestros hallazgos nos mostraron una realidad diferente a lo que anticipábamos.

- **Beneficio de las bicicletas:** Contrario a lo que podríamos haber pensado inicialmente, el beneficio obtenido de las bicicletas no mostró un crecimiento

lineal conforme se incrementaba el número de estas. En lugar de ello, este beneficio se mantuvo constante, lo cual fue una sorpresa para nosotros.

- **Tiempo en función del número de furgonetas:** Otro resultado inesperado fue que el tiempo no mostró un aumento proporcional al incrementar el número máximo de furgonetas. Esto puede indicarnos que, a pesar de tener más furgonetas disponibles, el problema está bien optimizado gracias a las restricciones que hemos establecido. Este hallazgo resalta la importancia de tener un diseño adecuado y unas restricciones bien definidas para lograr una ejecución eficiente.
- **Conclusión sobre el número óptimo de furgonetas:** A partir de los datos y resultados obtenidos, podemos concluir que el número óptimo de furgonetas es 10. Esto se debe a que, al utilizar 10 furgonetas, es cuando se observa el menor tiempo de ejecución, lo que indica una mayor eficiencia en la operación. Pero no afecta mucho en el resultado.

Se confirma la Hipótesis Nula (H0)

6.6.5 Comentarios adicionales

A lo largo de nuestro análisis, hemos observado ciertas tendencias y patrones que merecen ser resaltados. Una constante que hemos notado es la varianza. A pesar de las variaciones en diferentes parámetros y condiciones del estudio, la varianza tiende a mantenerse uniforme. Esto sugiere que, independientemente de las fluctuaciones puntuales, hay un nivel de consistencia en los datos que recogimos. Esta consistencia es crucial, ya que nos proporciona confianza en la robustez de nuestros hallazgos y en la fiabilidad de las conclusiones que extraemos de ellos.

7. ANÁLISIS COMPARATIVO ENTRE HILL CLIMBING Y SIMULATED ANNEALING.

Hill Climbing y Simulated Annealing son dos métodos de optimización que difieren en su enfoque y funcionamiento.

Hill Climbing genera nuevas soluciones aplicando sobre una solución inicial acciones, seleccionando la mejor de las nuevas soluciones generadas según el heurístico utilizado. Simulated Annealing, en cambio, se distingue por su enfoque aleatorio: aplica una única acción seleccionada al azar entre todas las posibles soluciones que podemos realizar. Este algoritmo, además, introduce parámetros que Hill Climbing no posee y que permiten regularlo.

Simulated Annealing busca imitar un fenómeno natural, en concreto, en un fenómeno de la metalurgia donde se calienta un metal y se va enfriando haciendo que los átomos del mismo se reorganicen en una mejor disposición. Acorde a esta intención para este algoritmo se emplea una temperatura, regulada por el valor de K, que se puede analogar al hecho sacudir una bandeja con oquedades con más o menos intensidad. Al sacudir la bandeja haremos que la canica(que esta sobre esta) vaya moviéndose entre las oquedades. Además de la temperatura se utiliza otro parámetro, lambda, para controlar el enfriamiento de esta temperatura en cada iteración, y se establece un límite que, si se alcanza, detiene el algoritmo. De nuevo con la analogía de la bandeja, lo que hace este algoritmo es sacudir una bandeja haciendo que la canica vaya saltando de oquedad en oquedad hasta que la temperatura no es suficiente para que salte a otra o la oquedad sea demasiado profunda.

Esta configuración única del Simulated Annealing permite evitar estancamientos en mesetas o máximos locales, un problema que el Hill Climbing puede experimentar. Sin embargo, los parámetros de simulated annealing deberán ser modificados hasta encontrar una combinación de parámetros que nos permita obtener un mejor resultado.

En cuanto a nuestro experimento, hemos podido ver precisamente lo anterior: hill climbing encuentra una peor solución debido a que rápidamente se encuentra ante una meseta o máximo local mientras que simulated annealing va más lejos, realiza más pasos, y halla una mejor solución. Esto se ve reflejado en sus tiempos de

ejecución, siendo más rápido el de hill climbing y realizando este menos pasos por qué rápidamente se encuentra ante una meseta o máximo local mientras que simulated annealing es más lento y realiza muchos más pasos, lo que le permite obtener mejores resultados

8. REFLEXIONES Y RESPUESTAS

Hemos visto con los resultados de los experimentos los mejores parámetros, operadores, soluciones iniciales etcétera que nos permiten obtener un algoritmo de búsqueda local que obtenga el mejor resultado posible.

1. En cuanto a operadores, de este primer experimento vimos que los operadores que nos proporcionaban el mejor resultado era nuestra tercera opción: no usamos el operador de incrementar bicis y creamos las furgonetas transportando ya entre 1 y máximo número de bicis transportables.
2. En el segundo experimento pudimos concluir de manera clara que la mejor de las soluciones iniciales propuestas es la segunda solución inicial, una solución greedy que cogía la mayor cantidad de bicis posibles y las deja en los lugares donde más bicis faltan
3. Este tercer experimento tenía como propósito conseguir la mejor combinación de los parámetros K y lam de el algoritmo simulated annealing, y concluímos que la mejor combinación era K=5 y lam=0.1. Este experimento, sin embargo, es el menos conclusivo de todos los hechos pues fluctúan mucho los resultados cada vez que ejecutamos el mismo pese a las réplicas realizadas con cada combinación de parámetros que pretendían precisamente que esto no ocurriese.
4. En este experimento analizamos como escala nuestro algoritmo y, pudimos concluir, no escala muy bién, alcanzando valores superiores a los 250 segundos al llegar a las 100 estaciones
5. En este quinto experimento los resultados no han sido muy esclarecedores debido a los parámetros del simulated annealing, sin embargo, sí hemos podido concluir que este algoritmo de búsqueda obtiene mejores resultados que el algoritmo hill climbing. Esto debe deberse a que hill climbing se encuentra con una meseta o máximo local rápidamente, problemas que simulated annealing no padece.
6. En este experimento nos ha permitido determinar que a partir de las cinco furgonetas el tiempo de ejecución y el beneficio obtenido deja de mejorar.
7. Este experimento ya fué entregado previamente como un documento aparte. Cabe recalcar que el código usado para dicho experimento especial ha sido completamente modificado, desde los heurísticos a los operadores pues

posteriormente nos hemos dado cuenta de de un gran número de errores, malas optimizaciones y malentendidos que hemos ido corrigiendo hasta transformar totalmente dicho código.