



UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Inteligencia Artificial

Práctica de Búsqueda Local

Adrian Cristian Crisan
Javier Castaño
Priyanka Amarnani
Mark Smithson

DEPARTAMENTO DE COMPUTER SCIENCE

22 de octubre de 2022

Índice

1. Parte Descriptiva	1
1.1. Descripción del problema	1
1.1.1. Elementos del problema	1
1.1.2. Búsqueda local	1
1.1.3. Espacio de búsqueda	2
1.2. Implementación del estado	2
1.3. Operadores del problema	3
1.4. Generación de soluciones iniciales	4
1.5. Función heurística	9
2. Parte Experimental	10
2.1. Experimento 1	10
2.2. Experimento 2	12
2.3. Experimento 3	15
2.4. Experimento 4	17
2.5. Experimento 5	20
2.6. Experimento 6	25
3. Conclusiones	28
4. Trabajo de innovación: Alexa	29
4.1. Breve descripción	29
4.2. Reparto del trabajo	29
4.3. Referencias	29
4.4. Dificultades	29

1. Parte Descriptiva

1.1. Descripción del problema

El problema plantea el caso de una empresa de generación de electricidad que gestiona un parque de centrales eléctricas y busca maximizar su beneficio. Distinguimos diferentes tipos de centrales y clientes. Los clientes se caracterizan por el consumo y existen tarifas por cada tipo de cliente y consumo. También cabe destacar que los clientes se separan en dos grandes grupos: los clientes garantizados y los no garantizados. En el caso de no poder suministrar a un cliente no garantizado durante un día, este recibirá una indemnización en función del consumo.

1.1.1. Elementos del problema

El problema está formado por diferentes características y elementos, entre ellos hay dos que destacan como principales y que nos marcan unas condiciones y restricciones concretas. Estas características nos permiten plantear el problema de una cierta manera y nos guían hacia la implementación que hemos usado. En nuestra implementación se ha buscado una manera de representar el estado que nos permitiese tener la menor complejidad posible para que los algoritmos de búsqueda local usados tardasen lo menos posible. Los dos elementos principales son:

- **Clientes:** En el caso de los clientes nos guardamos a qué central está cada uno asignado en caso de que lo esté.
- **Centrales:** Con las centrales la única información que necesitamos es la cantidad de energía que le queda por repartir entre clientes.

1.1.2. Búsqueda local

Para resolver el problema utilizaremos la búsqueda local. Esta metodología consiste en partir de una solución inicial e ir estudiando los próximos estados posibles. Para llegar a estos estados aplicamos operadores sobre el estado inicial/anterior. La búsqueda local nos permite encontrar una solución válida para nuestro problema de manera más rápida y eficiente en comparación con otros métodos como el backtracking. Este problema puede ser abordado con búsqueda local ya que cumple los requisitos para ello. Estos requisitos son:

- Es un problema de optimización con el único objetivo de generar un estado final según una función de calidad.
- El camino que se toma para llegar a una solución óptima es totalmente irrelevante.

Estas características hacen posible poder aplicar búsqueda local para resolver este problema.

1.1.3. Espacio de búsqueda

El espacio de búsqueda es el conjunto de las posibles soluciones a nuestro problema. Es muy importante realizar un estudio del espacio de búsqueda de nuestro problema para verificar si realmente vale la pena la aplicación de un algoritmo de búsqueda local. Para nuestro caso partimos de la idea de tener n clientes y m centrales. Si suponemos que tenemos un vector de n posiciones donde cada n_i indica que central esta subministrando al cliente i . En caso de que el cliente no este asignado a ninguna central diremos que $n_i = -1$. Esto hace que el espacio de soluciones sea del orden de $O(n^{m+1})$. Como podemos observar seria completamente ilógico utilizar un algoritmo de *backtracking* o similar para un problema con estas características.

1.2. Implementación del estado

Para representar el estado es necesario buscar un equilibrio entre un estado suficientemente representativo y óptimo en cuestión de lo que ocupa en memoria, ya que en caso contrario, la generación de todos los sucesores nos dará problemas. Desde el primer momento prestamos especial atención al uso de la memoria, por lo que empezamos definiendo los atributos que no cambiarían en todo el problema para así poder declararlos estáticos. Lo primero que definimos estático son los atributos `seed`, `n_centrales`, `n_clientes`, `prop_clientes`, `clientes_garant`, `centrales` y `clientes`. El primer atributo estático, fuera de los que define el problema, que consideramos fue la distancia que hay entre las centrales y los clientes, dado que estas se mantienen constantes a lo largo del problema. Esta información se guarda en la array 2-dimensional de `dists`. La distancia se calcula de la siguiente forma:

$$d(A, B) = \sqrt{(B_x - A_x)^2 + (B_y - A_y)^2}$$

Como atributos no estáticos, primero tenemos una array de reales que guarda la energía que le queda disponible por servir a cada central, este atributo es `CCanServe`.

Otro atributo importante en nuestra implementación es el de `CLServed`. Este atributo es una array de enteros que nos guarda para cada cliente la central que tiene asignada, si es que tiene.

$$CLServed[i] = \begin{cases} \text{central_asignada,} & \text{si el cliente } \rightarrow i \text{ tiene una central asignada} \\ -1, & \text{en caso contrario} \end{cases}$$

Al principio este atributo se implementó como una array de booleanos donde en cada posición guardábamos si los clientes estaban asignados o no. Más tarde se cambió por la array de enteros, facilitando la consulta de que central tiene asignada un determinado cliente, mejorando el coste del cálculo de $O(n^2)$ a $O(1)$. Este cambio también nos hizo quitar una matriz $O(n * m)$ de booleanos que teníamos previamente declarada para guardar la información de que clientes estaban asignados a que centrales. Esto mejoró mucho el tiempo que tardaba en ejecutarse el algoritmo.

Por último, tenemos el atributo de `beneficio` que va guardando el beneficio del estado.

Creemos que esta es una buena representación del estado ya que nos permite usar los operadores de manera eficiente, puesto que tenemos ya muchos datos pre-calculados. Declarar los atributos que no cambian su valor como estáticos nos ayudará en consumir menos memoria, sobretodo si consideramos la generación de miles, incluso millones, de estados sucesores.

1.3. Operadores del problema

Estos operadores son los que hemos definido para el problema. Para facilitar el factor de ramificación diremos que trabajamos con m centrales y n clientes.

1. Añadir

- *Parámetros:* Los parámetros serán el índice cl de un cliente y el índice C de una central.
- *Precondición:* El cliente cl no tiene ninguna central asignada. La central C puede producir energía para el cliente cl .
- *Postcondición:* El cliente cl tiene asignada la central C y se recalcula el beneficio.
- *Factor de ramificación:* Cualquier cliente puede añadirse a cualquier central así que el factor de ramificación es de $O(n * m)$.

2. Eliminar

- *Parámetros:* Los parámetros serán el índice cl de un cliente y el índice C de una central.
- *Precondición:* El cliente cl tiene asignada la central C . El cliente cl es de tipo *No Garantizado*.
- *Postcondición:* El cliente cl pasa a no tener ninguna central asignada. La central C deja de subministrar electricidad al cliente cl y se recalcula el beneficio.
- *Factor de ramificación:* Solo los clientes *No Garantizados* pueden eliminarse de las centrales. Estos pueden eliminarse de cualquier central. Por tanto el factor de ramificación es de $O(p_{ng} * n * m)$. Donde p_{ng} es la proporción de clientes *No Garantizados*.

3. Mover

- *Parámetros:* Los parámetros serán el índice cl de un cliente, el índice C_1 de una central y el índice C_2 de otra central.
- *Precondición:* $C_1 \neq C_2$. La central C_1 subministra al cliente cl . La central C_2 puede subministrar electricidad al cliente cl .

- *Postcondición:* La central C_1 deja de subministrar electricidad al cliente cl . La central C_2 subministra electricidad al cliente cl y se recalcula el beneficio.
- *Factor de ramificación:* Cualquier cliente puede moverse a ser subministrado por cualquier otra central, por tanto el coste es $O(n * m)$.

4. Swap

- *Parámetros:* Los parámetros serán el índice cl_1 de un cliente, el índice C_1 de una central, el índice cl_2 de otro cliente y el índice C_2 de otra central.
- *Precondición:* $cl_1 \neq cl_2$. $C_1 \neq C_2$. Al cliente cl_1 le subministra electricidad la central C_1 y al cliente cl_2 le subministra electricidad la central C_2 . La central C_1 es capaz de dejar de subministrar electricidad al cliente cl_1 y pasar a subministrar electricidad al cliente cl_2 . La central C_2 es capaz de dejar de subministrar electricidad al cliente cl_2 y pasar a subministrar electricidad al cliente cl_1 .
- *Postcondición:* La central C_1 deja de subministrar al cliente cl_1 y pasa a subministrar al cliente cl_2 . La central C_2 deja de subministrar al cliente cl_2 y pasa a subministrar al cliente cl_1 . Se recalcula el beneficio.
- *Factor de ramificación:* Cada cliente puede intercambiarse con todos los demás. Como se evitan repeticiones para no hacer swap de cl_1 con cl_2 y luego swap de cl_2 con cl_1 . El factor de ramificación es de $O(\frac{n*(n+1)}{2})$.

5. Vaciar central

- *Parámetros:* El índice de una central C .
- *Precondición:* Todos los clientes que tengan electricidad subministrada por la central C pueden ser asignados a otras centrales.
- *Postcondición:* Todos los clientes de la central C han sido asignados a otras centrales y C pasa a no subministrar a nadie.
- *Factor de ramificación:* Cada central puede vaciarse. Por tanto el factor de ramificación es de $O(m)$.

Creemos que estos operadores son adecuados para el problema ya que abarcan un gran espacio de búsqueda. Después de realizar varias pruebas previas a los experimentos hemos decidido no usar el operador de **Vaciar central**. Este operador tenía un cómputo grande lo cual ralentizaba mucho el algoritmo, y no abarcaba un gran espacio de soluciones.

1.4. Generación de soluciones iniciales

Para realizar el estudio y análisis del problema hemos trabajado con 5 soluciones iniciales distintas. Para hablar del coste de cada solución inicial, diremos que los clientes serán n y las centrales como m .

1. **Asignación de clientes garantizados a centrales en orden de llegada:** En esta solución los clientes garantizados se asignan a las centrales sin ningún tipo de criterio o orden concreto, la asignación se hace por como llegan los clientes en el momento de ejecución. Se coge una central y se llena con todos los clientes garantizados que pueda servir, y se repite el mismo proceso con la siguiente central. Tiene un coste de $O(n * m)$.

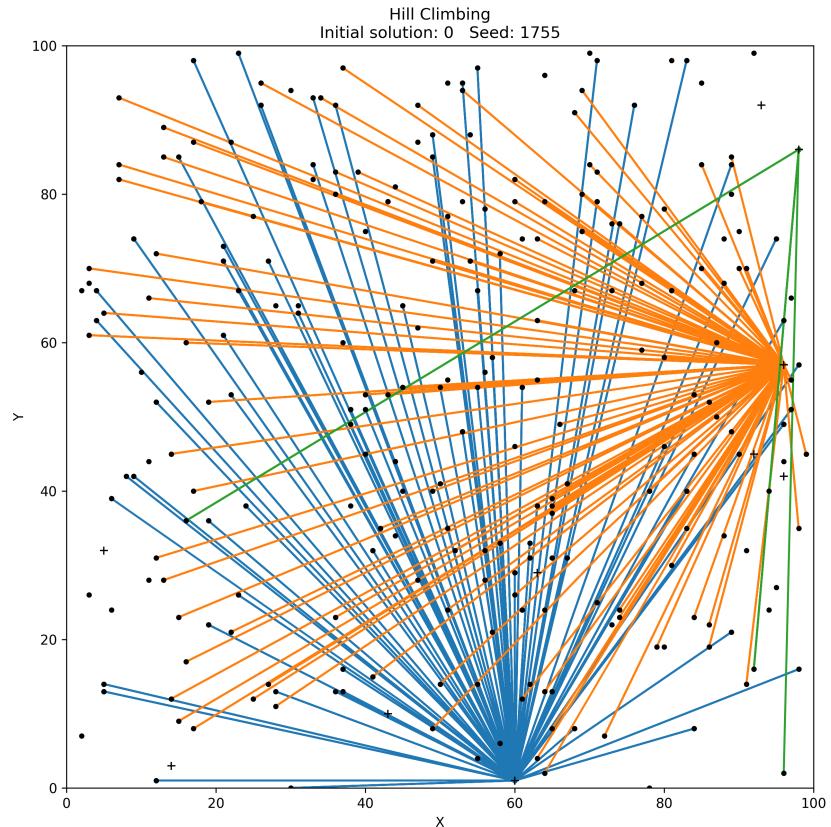


Figura 1: Solución inicial por orden de llegada

2. Asignación de clientes garantizados a centrales de manera uniforme: La segunda solución inicial que hemos considerado, trata de asignar a los clientes garantizados a centrales en orden de llegada, de la misma manera que la solución anterior, pero esta vez se trata de asignar clientes a centrales de manera uniforme, es decir, que las centrales tengan un número similar de clientes a subministrar. Tiene un coste de $O(n + n * m)$.

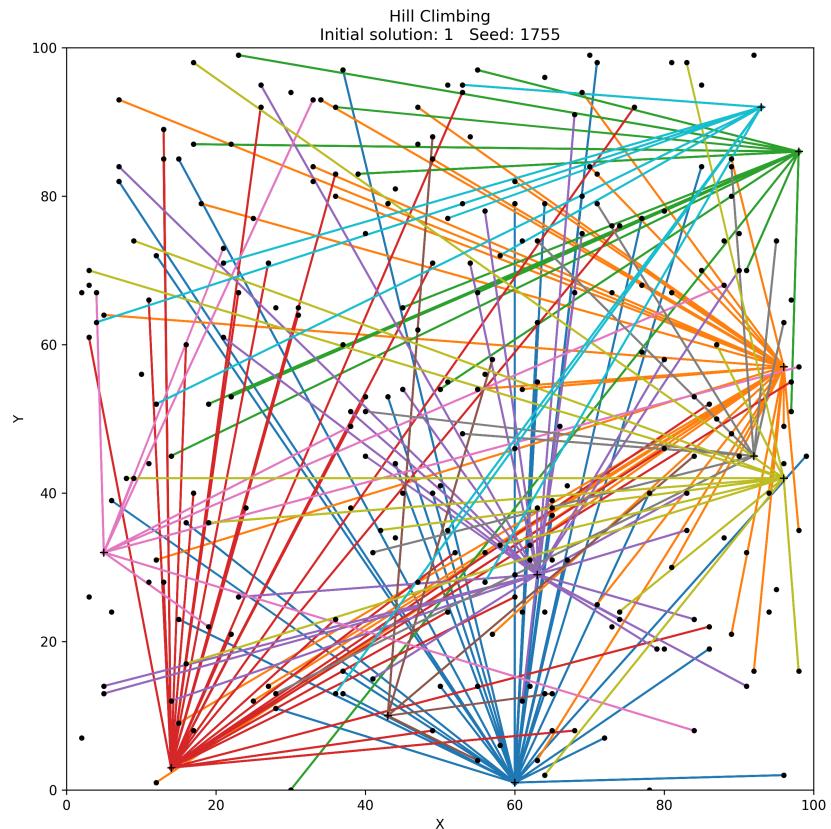


Figura 2: Solución inicial uniforme (clientes garantizados)

3. Asignación de clientes a centrales de manera uniforme: Esta solución inicial es parecida a la anterior, también trata de realizar una asignación uniforme de clientes a centrales pero trabajando con todos los clientes posible, incluyendo así a los clientes con contratos no garantizados. Tiene un coste $O(n + 2 * n * m)$.

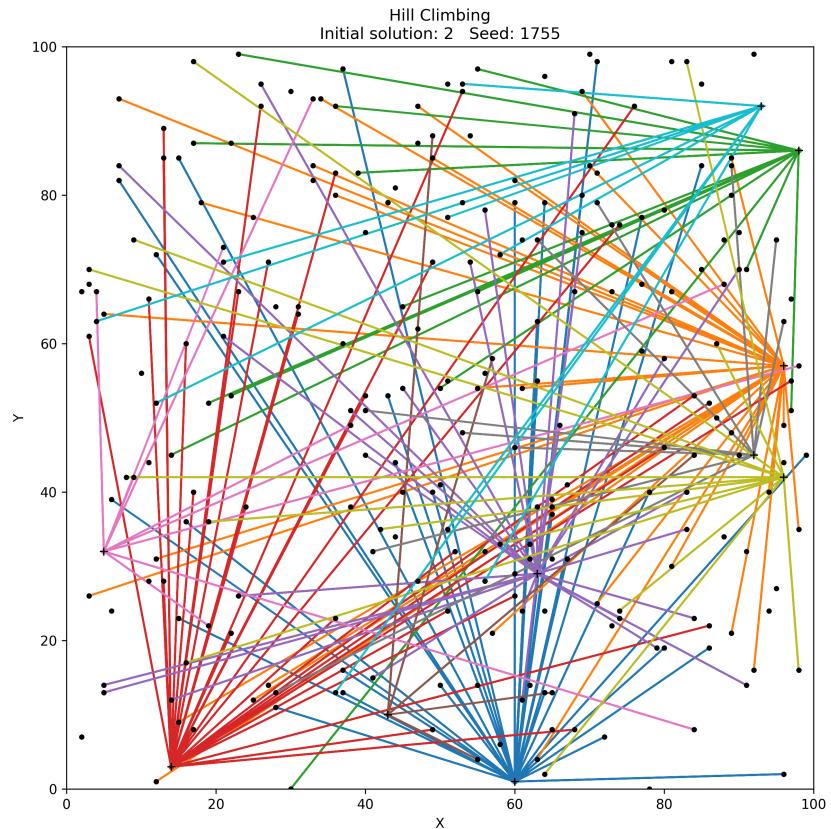


Figura 3: Solución inicial uniforme (todos los clientes)

4. Asignación ordenada: La cuarta solución inicial que hemos considerado ordena a todos los clientes de manera creciente por su consumo, de esta manera se asignan centrales primero a los clientes que más van a consumir, con la finalidad de maximizar el beneficio. Tiene un coste $O(n * \log(n) + n * m)$.

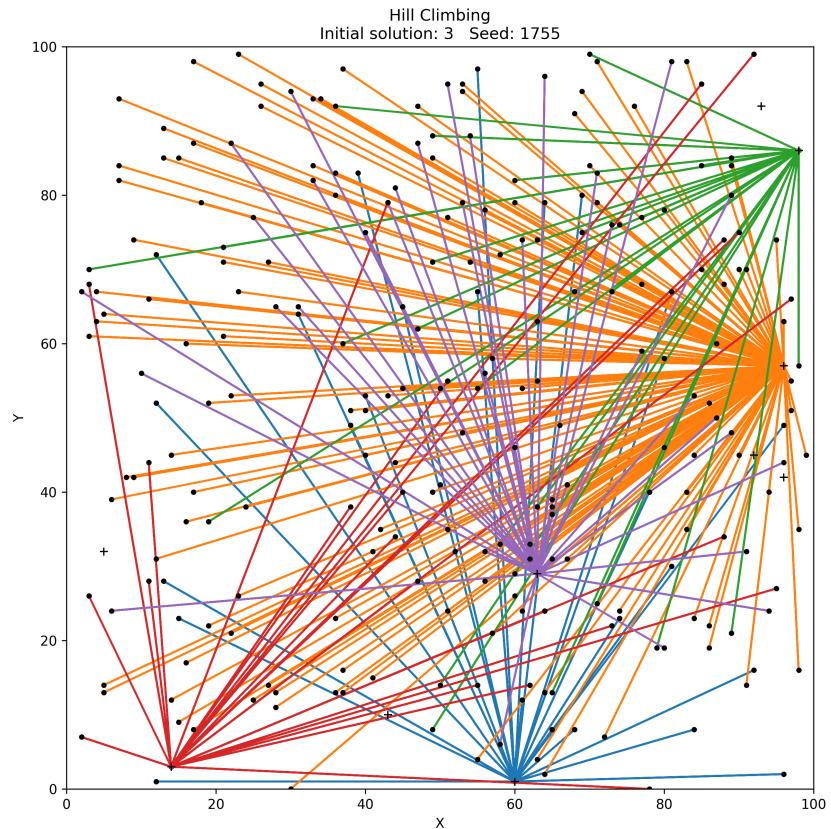


Figura 4: Solución inicial ordenada

5. Asignación según la distancia: La última solución inicial que hemos considerado estudiar asigna a los clientes de manera que la distancia entre la central y el cliente sea mínima, es decir una central C suministrará a todos los clientes posibles que se encuentren a su alrededor. Esta solución se enfoca más en perder la mínima cantidad de energía, priorizando a los clientes más cercanos. Tiene un coste de $O(n + n * m)$.

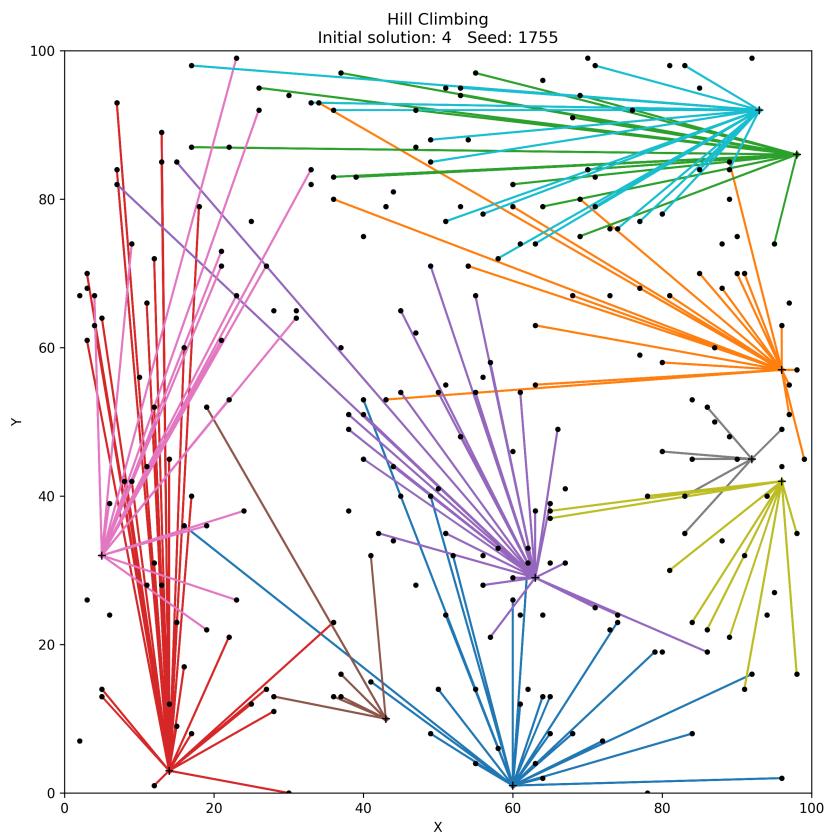


Figura 5: Solución inicial por distancias

1.5. Función heurística

La función heurística que usaremos para esta práctica es la que viene determinada por el enunciado, maximizar el beneficio. Esta función heurística es una función sencilla pero, como veremos en los experimentos, da resultados suficientemente buenos.

2. Parte Experimental

2.1. Experimento 1

En este experimento se nos plantea comprobar qué operadores dan mejores resultados con el siguiente escenario: el número de centrales de cada tipo es 5 (A), 10 (B) y 25 (C), los clientes son 1000, tienen una proporción de 25 % (XG), 30 % (MG) y 45 % (G) según su tipo y una proporción del 75 % con suministro garantizado. Este experimento se hará con el algoritmo de Hill Climbing. Para este experimento, como aún no hemos experimentado con las diferentes soluciones iniciales para decidir cuál es la mejor, lo haremos con las 5 que planteado.

1. **Observaciones** Es lógico pensar que, dependiendo de la solución inicial con la que empieces, no todos los operadores se usarán de la misma manera.
 2. **Planteamiento** Para estudiar y realizar este experimento cuantificaremos el número de veces que se usa cada operador, de esta manera podremos sacar las conclusiones adecuadas con más comodidad.
 3. **Hipótesis** Dependiendo de el estado inicial se usarán más unos operadores que otros pero lo que si creemos es que rara vez se utilizará el operador de **Eliminar** ya que esto supondría perder un beneficio a cambio de pagar menos electricidad pero si nos fijamos en como está distribuido el beneficio que nos aporta cada cliente y el coste de producción vemos que lo que aporta el cliente es mucho mayor que el hecho de eliminar un cliente de una central.
4. **Metodología**
- Usaremos el escenario comentado anteriormente para generar las soluciones iniciales.
 - Usaremos el algoritmo Hill Climbing.
 - Ejecutaremos el programa 10 veces con cada solución inicial con diferentes **seed**.
 - Calculamos el número de veces en promedio que se usa cada operador con cada solución inicial.

5. Resultados

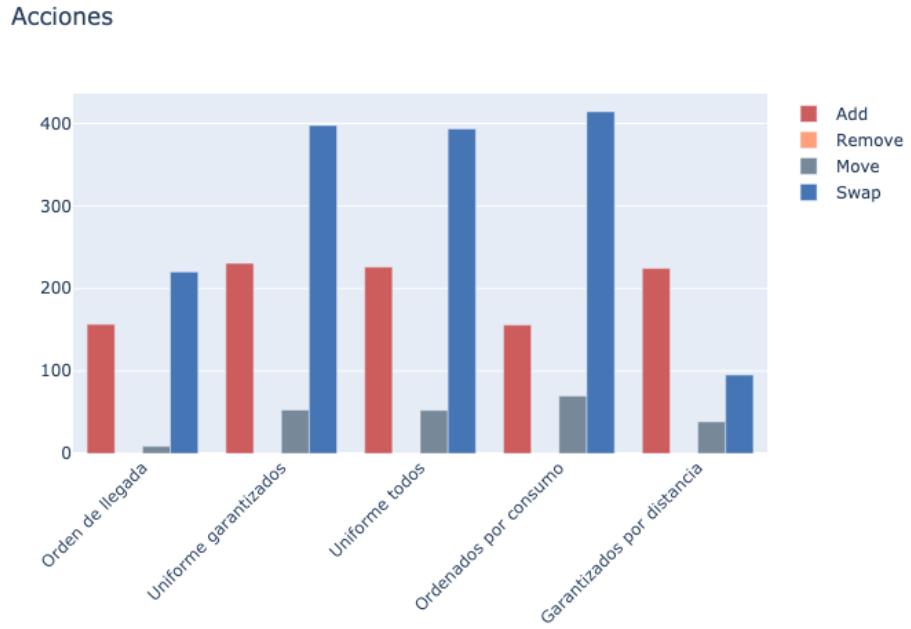


Figura 6: Número de usos de cada operador dependiendo de la solución inicial

6. **Conclusión** Como podemos observar, nuestras hipótesis iban por buen camino aunque no nos esperábamos que el operador de **Eliminar** no se usaría ninguna vez. La razón de este fenómeno es la que se comenta en la hipótesis, nunca habrá más beneficio quitando un cliente y así teniendo menos cosecha de producción que teniendo ese cliente y por ende tener un mayor coste de producción. Podemos ver como el **Swap** es el operador más usado como era de esperar junto con el **Añadir**. Con estas conclusiones en los experimentos de ahora en adelante no usaremos el operador de **Eliminar**, puesto que este no mejora el heurístico y además gasta recursos.

2.2. Experimento 2

Para el segundo experimento se nos propone determinar qué estrategia de generación de la solución inicial genera mejores resultados para la función heurística y escenario utilizados en el anterior experimento, y de nuevo, utilizando el algoritmo de Hill Climbing.

1. **Observaciones** Visto que tenemos muchas soluciones iniciales, habrán algunas que repitan resultados o que sean muy parecidos, por lo tanto, podremos eliminarlas para más tarde seleccionar aquellas que nos van mejor.
2. **Planteamiento** Para este experimento compararemos los tiempos, nodos expandidos, coste y beneficio total de cada solución para poder sacar conclusiones coherentes y adecuadas.
3. **Hipótesis** Partimos de la hipótesis que los estados iniciales que se generan de una manera más *greedy* empezarán con mejores soluciones, pero expandirán poco y darán peores resultados que las demás.

4. Metodología

- Usaremos el mismo escenario del experimento anterior
- Aplicaremos el algoritmo de Hill Climbing
- Ejecutaremos el programa 10 veces con diferentes `seed` para cada solución inicial.
- Compararemos la heurística, el tiempo y los nodos expandidos de cada solución inicial para poder sacar conclusiones con un amplio conocimiento de las capacidades de cada estado inicial.

5. Resultados

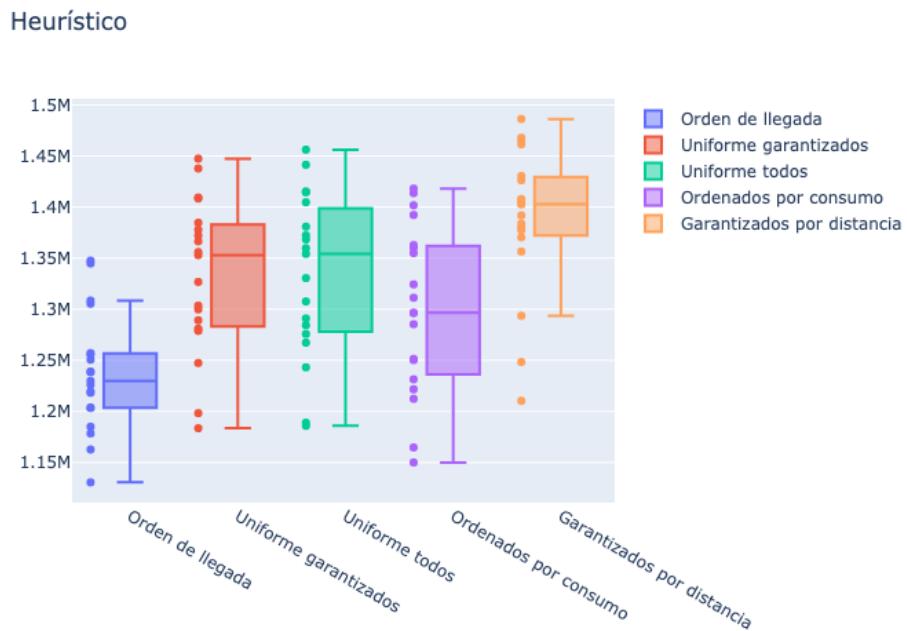


Figura 7: Resultados de las heurísticas con las diferentes soluciones iniciales

Nodos expandidos

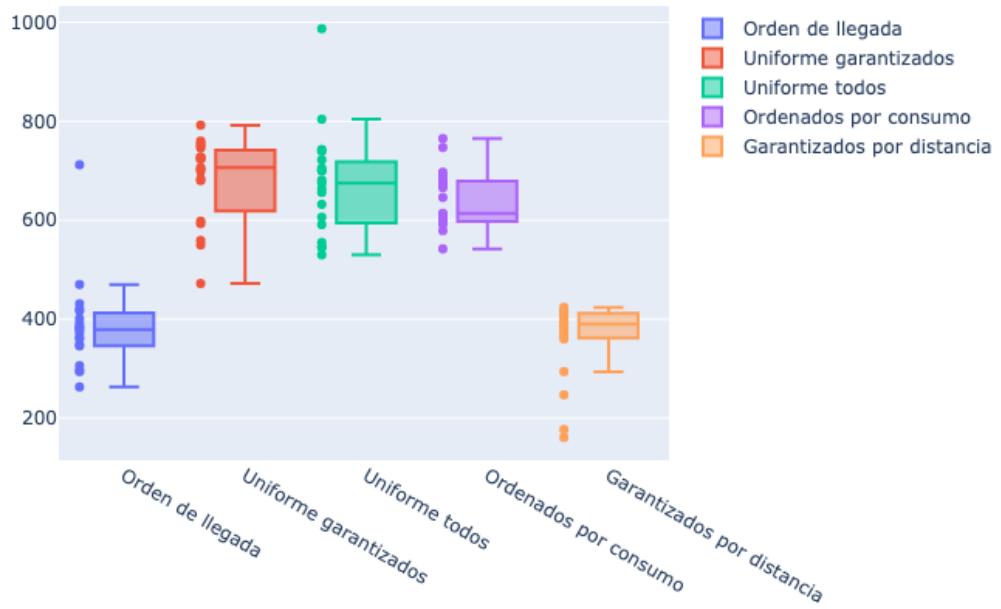


Figura 8: Resultados de los nodos expandidos con las diferentes soluciones iniciales

Tiempos

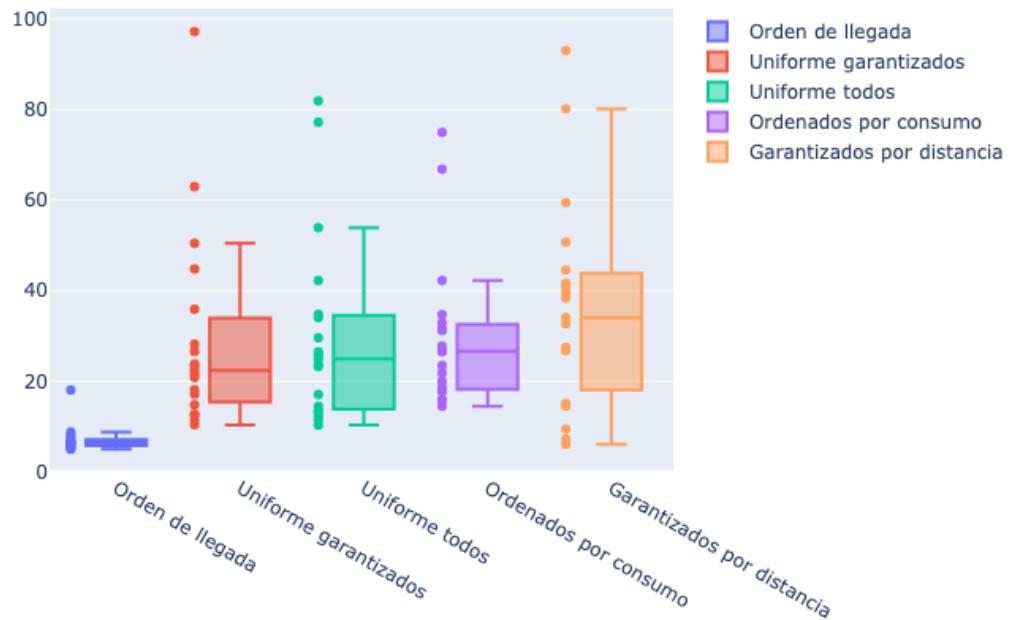


Figura 9: Resultados de los tiempos con las diferentes soluciones iniciales

Paired t-test		
Alpha	0.05	
Hypothesized Mean Difference	0	
	Variable 1	Variable 2
Mean	1299977.53842105	1387923.50473684
Variance	6750551870.07894	5189807586.80804
Observations	19	19
Pearson Correlation	0.750430250180109	
Observed Mean Difference	-87945.9663157894	
Variance of the Differences	3056829000.62207	
df	18	
t Stat	-6.93357397013218	
P (T<=t) one-tail	8.82349538317682E-07	
t Critical one-tail	1.73406360661754	
P (T<=t) two-tail	1.76469907663536E-06	
t Critical two-tail	2.10092204024104	

6. **Conclusión** Como podemos ver en los resultados de la heurística hay 3 soluciones iniciales que son mejores que las demás. Estas son las dos de asignación uniforme y la de distancias. La primera solución propuesta, la de asignar a los clientes garantizados por orden de llegada, condiciona al algoritmo de Hill Climbing, como esta llena las centrales al máximo antes de pasar a la siguiente, se concentran un gran número de clientes en pocas centrales. En el escenario explicado y con los operadores que usamos nosotros no se consigue aprovechar al máximo el potencial de las centrales y, en consecuencia, no conseguimos mejorar la heurística. Las siguientes dos soluciones son muy parecidas entre sí, pues una reparte los clientes garantizados uniformemente por las centrales y la otra hace lo mismo pero con todos los clientes. Estas dos soluciones forman parte de las soluciones con mejores heurística. Las dos soluciones se construyen de manera similar y sus resultados son muy parecidos. La solución que ordena a los clientes por consumo no destaca en sus resultados. Por último, la solución que trabaja con el criterio de las distancias es lógicamente la que mayor heurística consigue, dado que sirve a los clientes garantizados cercanos a cada central, minimizando la pérdida de energía, maximizando así el beneficio. Si observamos las figuras que nos muestran los tiempos y nodos expandidos de cada solución inicial, cabe destacar que la solución de usar la distancia entre cliente y central parte de un buen beneficio inicial, aunque no es la que más expande, y tarda poco tiempo en encontrar una solución. Antes de escoger una solución inicial definitiva hemos decidido hacer un test t-Student para asegurarnos de que las medias de heurístico son diferentes. Para ello hemos escogido las dos que dan mejores resultados de heurístico que son la de *Uniformes todos* y la de *garantizados por distancia*. Podemos ver que haciendo el test con una confianza del 95 % podemos ver como el p-value nos da $\sim 8,82 * 10^{-7}$. Con este resultado podemos ver como las medias son realmente diferentes. Considerando los tres aspectos de las gráficas y el test de t-Student, hemos decidido escoger como mejor solución inicial a la de *Garantizados por distancia*.

2.3. Experimento 3

En este tercer experimento nuestro objetivo es determinar los parámetros que mejor resultado ofrecen para el algoritmo Simulated Annealing, de nuevo con la misma heurística, operadores y estrategia de generación de la solución inicial escogidos en los anteriores experimentos.

1. **Observaciones** Diferentes valores en los hiperparámetros de Simulated Annealing darán resultados diferentes.
2. **Planteamiento** Para cada conjunto de hiperparámetros nos quedaremos con la heurística para después poder compararlos.
3. **Hipótesis** Los valores extremos no funcionarán de la mejor manera.
4. **Metodología**
 - Usaremos el mismo escenario que en el experimento 1.
 - Usaremos el algoritmo de Simulated Annealing con $1e6$ de pasos.
 - Exploraremos los siguientes conjuntos de hiperparámetros.
 - $\text{stiter} = [1, 50, 100, 150, 200]$
 - $k = [0.5, 1, 25, 400, 8000]$
 - $\lambda = [1, 1e-1, 1e-2, 1e-3, 1e-4]$
 - Para cada conjunto de hiperparámetros ejecutaremos el programa 10 veces y haremos el promedio de la heurística.
 - Nos quedaremos con el promedio máximo entre todas las combinaciones.

5. Resultados

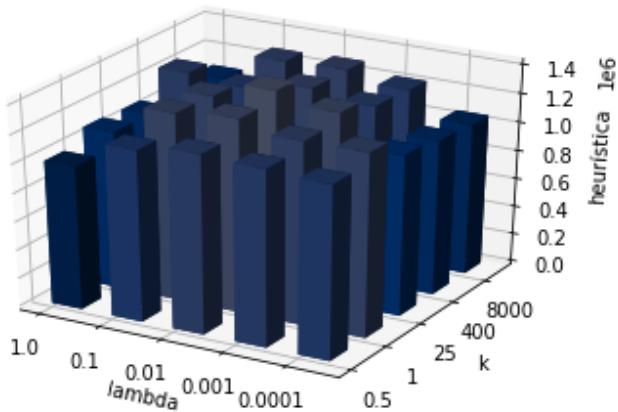


Figura 10: Resultados con los `steps = 1e6` y el `stiter = 100`

6. **Conclusión** Después de realizar el experimento podemos concluir que los parámetros que nos dan mejores resultados son $\text{stiter} = 100$, $\lambda = 0,01$ y $k = 25$. En la Figura 10 podemos ver que cuanto más al extremo nos vayamos más oscuro es el color lo que significa que la heurística es más baja. En el centro están los valores

más altos lo que es razonable sabiendo como funciona el algoritmo de Simulated Annealing.

2.4. Experimento 4

Este experimento consiste en, dado el escenario de los experimentos previos, estudiar la evolución del tiempo de ejecución para valores crecientes de los siguientes parámetros:

- Número de centrales, manteniendo las proporciones de los tipos del escenario inicial.
- Número de clientes, manteniendo las proporciones de los tipos del escenario inicial.

Para 40 centrales, empezaremos con 500 clientes e incrementaremos de 500 en 500 hasta notar la tendencia. Para 1000 clientes, comenzaremos con 40 centrales e incrementaremos de 40 en 40. De nuevo, utilizaremos Hill Climbing y la misma función heurística hasta ahora.

1. **Observaciones** Incrementando los parámetros del problema se vería un crecimiento en el tiempo de ejecución.
2. **Planteamiento** En cada ejecución y en cada caso, aumentamos el valor de un parámetro específico de manera uniforme.
3. **Hipótesis** El crecimiento del tiempo de ejecución será casi lineal en ambos casos.
4. **Metodología**
 - Empezando con el número de centrales, inicialmente tendremos 40 centrales y 1000 clientes (escenario inicial).
 - Aplicaremos el algoritmo del Hill Climbing.
 - Volveremos a ejecutar el algoritmo del Hill Climbing aumentando el número de centrales en 40.
 - Repetiremos el paso anterior hasta observar una tendencia, en nuestro caso hasta las 160 centrales.
 - Seguiremos la misma metodología para estudiar el crecimiento del número de clientes, empezando con 500 clientes y incrementando en 500 en cada ejecución.

5. Resultados

Gráfico coste temporal en función del número de centrales

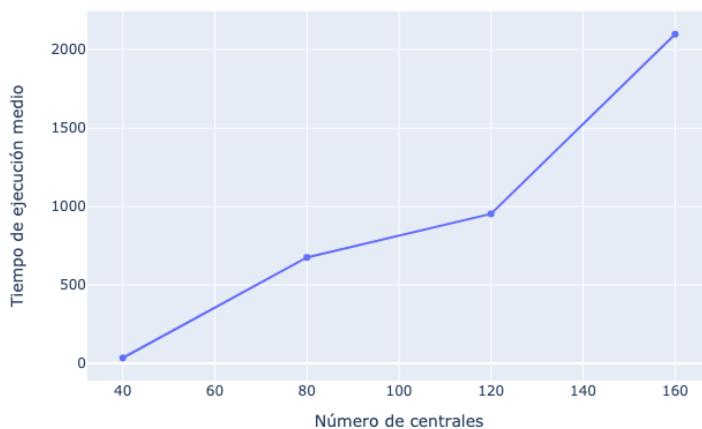


Figura 11: Resultados del coste temporal en función del número de centrales

Gráfico coste temporal en función del número de clientes

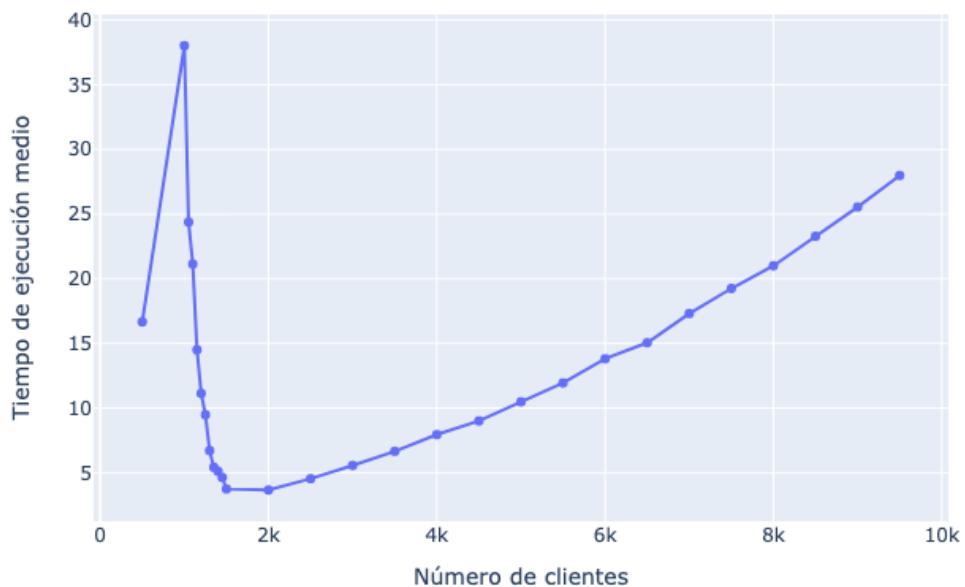


Figura 12: Resultados del coste temporal en función del número de clientes

Gráfico del heurístico aumentando el número de centrales

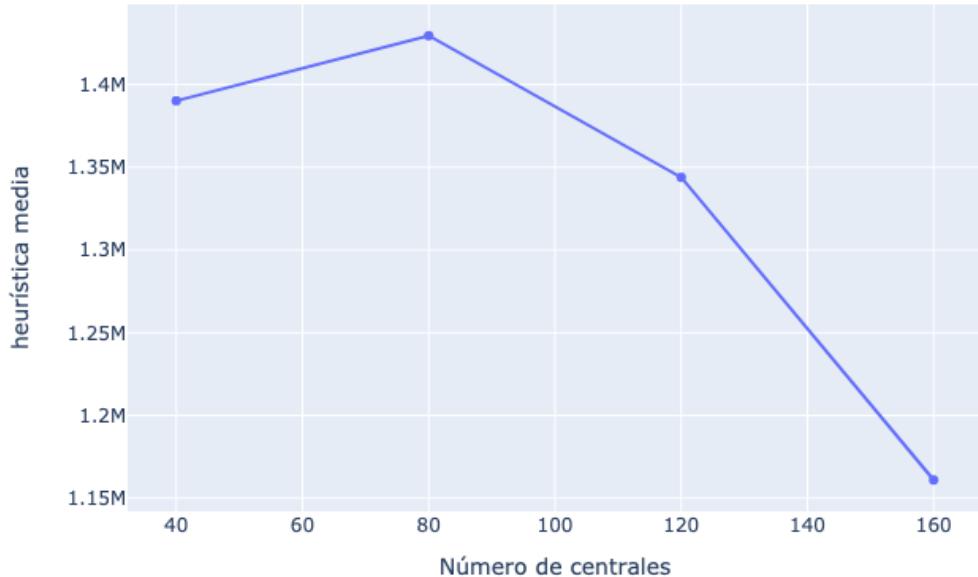


Figura 13: Resultados del heurístico en función del número de centrales

Gráfico del heurístico aumentando el número de cliente

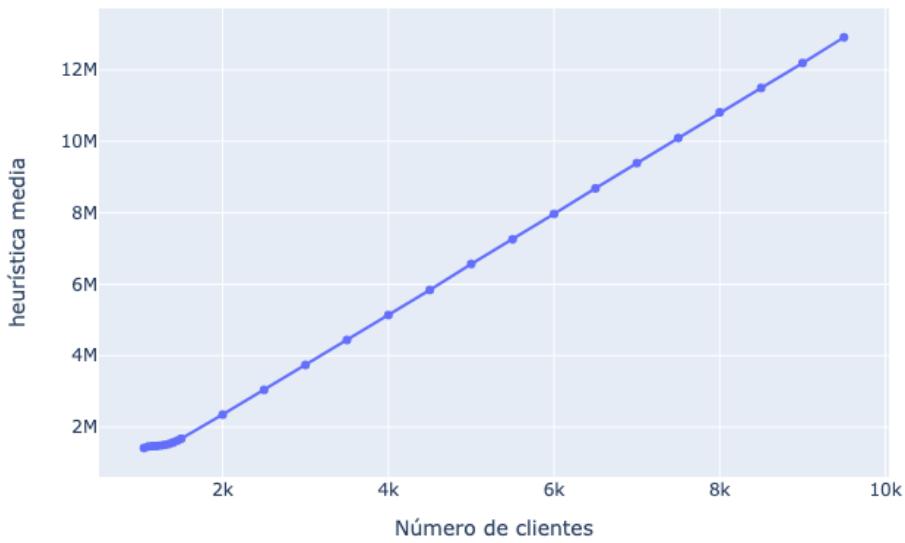


Figura 14: Resultados del heurístico en función del número de clientes

6. **Conclusión** Observando los resultados se aprecia que nuestra hipótesis se cumple en el caso del estudio del número de centrales. La gráfica de la Figura 11 muestra la relación entre el coste temporal y el número de centrales. En el escenario inicial empezamos con un tiempo medio negligible, este va creciendo a medida que crecen el número de centrales. El comportamiento del crecimiento es casi lineal. Por otro lado, los resultados de la relación entre el tiempo de ejecución y el número de clientes no sigue el comportamiento que habíamos pensado. Empieza con un valor alto y este va bajando en las siguientes dos ejecuciones, hasta llegar al escenario con 2000 clientes, donde consigue el mínimo coste temporal. A partir de este valor, la gráfica sigue un comportamiento de crecimiento lineal. Por lo que hemos concluido con los resultados de este experimento, podemos afirmar que nuestra hipótesis no es del todo incorrecta puesto que el crecimiento de números de centrales si que supone un crecimiento casi lineal del coste temporal y el número de clientes consigue esta linealidad a partir de un cierto valor.

A pesar de no comentarse en el enunciado, hemos decidido poner también unas gráficas de como evoluciona el heurístico en estos dos casos ya que nos parecía interesante. Como podemos ver, cada vez que aumentamos las centrales el beneficio es menor, esto se debe a que nosotros hemos escogido una solución inicial la cual se inicia tal que cada cliente garantizado es subministrado por su central más cercana. Esto provoca que hayan muchas centrales activas produciendo a pocos clientes y debido a que ya no tenemos el operador de **Eliminar**, que igual para este caso nos hubiese ido bien, el beneficio siempre es menor. La gráfica del beneficio según el número de clientes es bastante lógica ya que cuantos más clientes haya más clientes que aportan más beneficio se podrán asignar.

2.5. Experimento 5

Este experimento consiste en generar soluciones iniciales sin distinguir entre si son prioritarios o no prioritarios, pero dando una penalización a aquellos estados que no cumplan la restricción de dar electricidad a todos los clientes prioritarios. Usando los algoritmos de Hill Climbing y Simulated Annealing. También tenemos que determinar para que valor de penalización se cumple que siempre la solución final es válida.

1. **Observaciones** La nueva heurística y los cambios aplicados pueden mostrar un comportamiento diferente.
2. **Planteamiento** Partimos de una solución vacía y observamos a partir de que valores generamos soluciones válidas aplicando las modificaciones adecuadas.
3. **Hipótesis**
4. **Metodología**
 - Modificamos la función heurística para poder penalizar las soluciones que no suministren a clientes con contratos garantizados.
 - Partimos de una solución inicial vacía.
 - Aplicamos el algoritmo de Hill Climbing.
 - Aplicamos el algoritmo de Simulated Annealing.
 - Observamos los resultados y los valores que nos empiezan a dar soluciones válidas.

5. Resultados

Gráfico clientes garantizados en función de la penalización con Hill Climbing



Figura 15: Resultados del número de clientes garantizados en función de la penalización con Hill Climbing

Gráfico coste temporal en función de la penalización con Hill Climbing

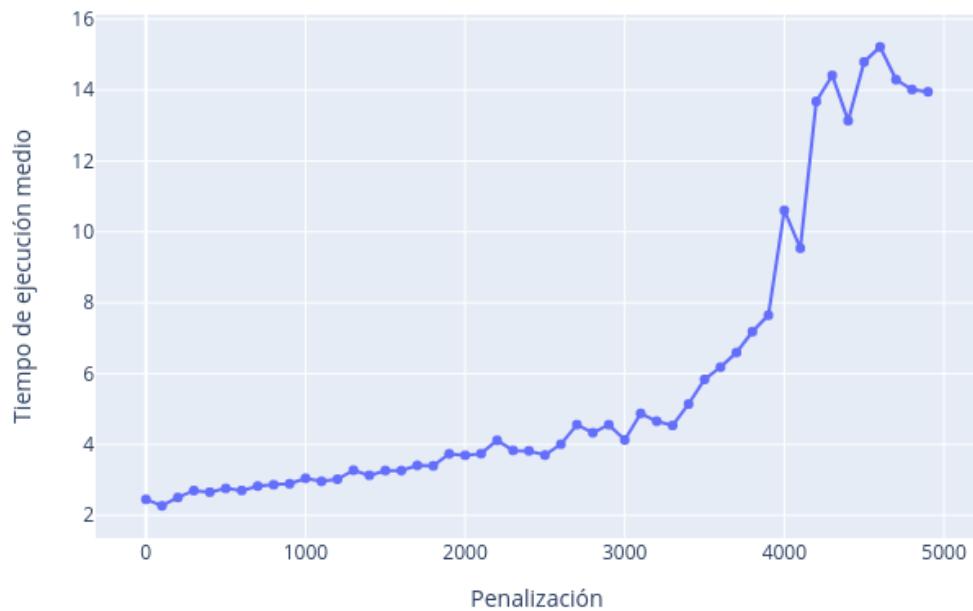


Figura 16: Resultados del coste temporal en función de la penalización con Hill Climbing

Gráfico heurística en función de la penalización con Hill Climbing

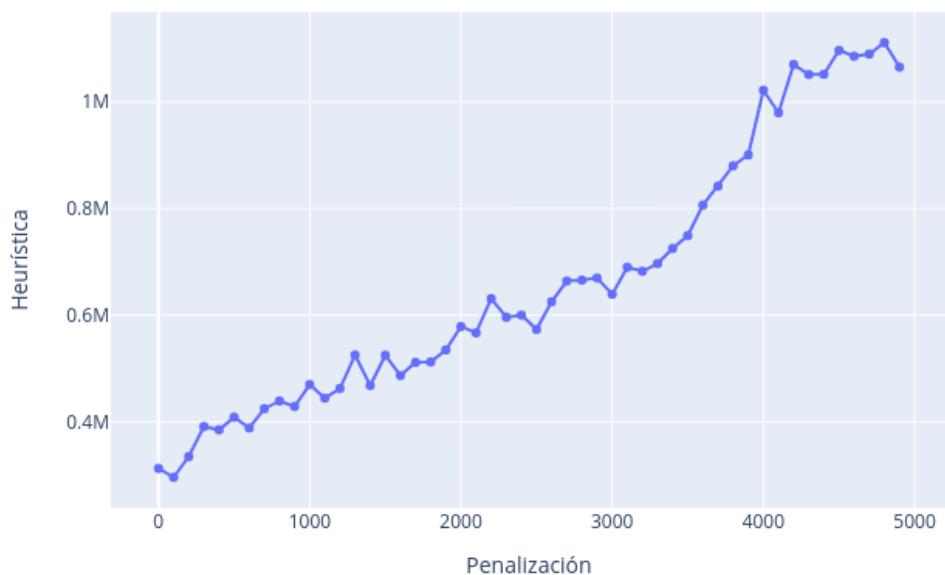


Figura 17: Resultados del heurístico en función de la penalización con Hill Climbing

Gráfico clientes garantizados en función de la penalización con Simulated Annealing

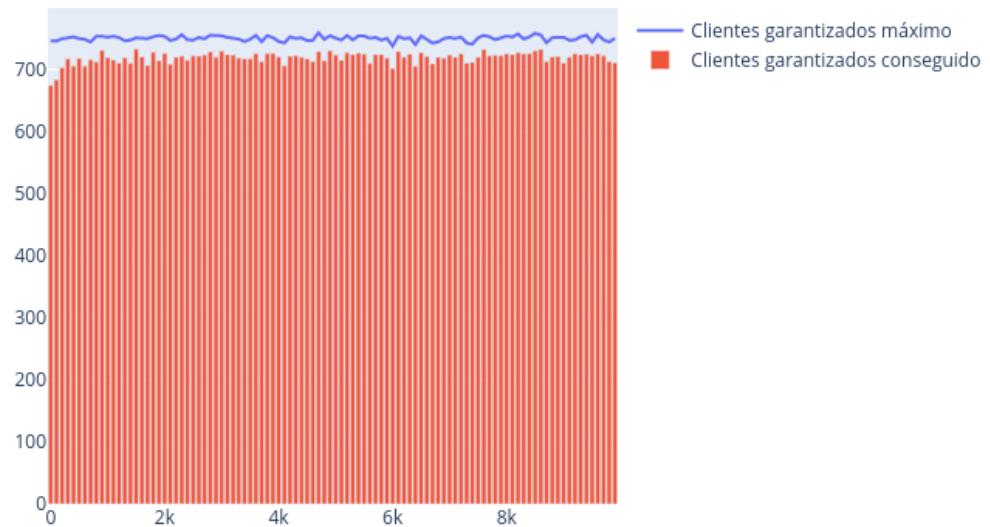


Figura 18: Resultados del número de clientes garantizados en función de la penalización con Simulated Annealing

Gráfico coste temporal en función de la penalización con Simulated Annealing

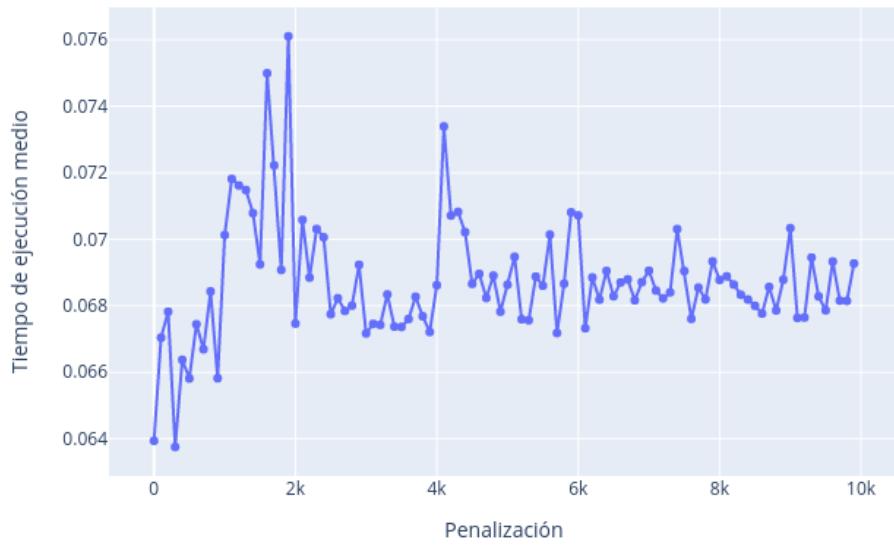


Figura 19: Resultados del coste temporal en función de la penalización con Simulated Annealing

Gráfico heurística en función de la penalización con Simulated Annealing

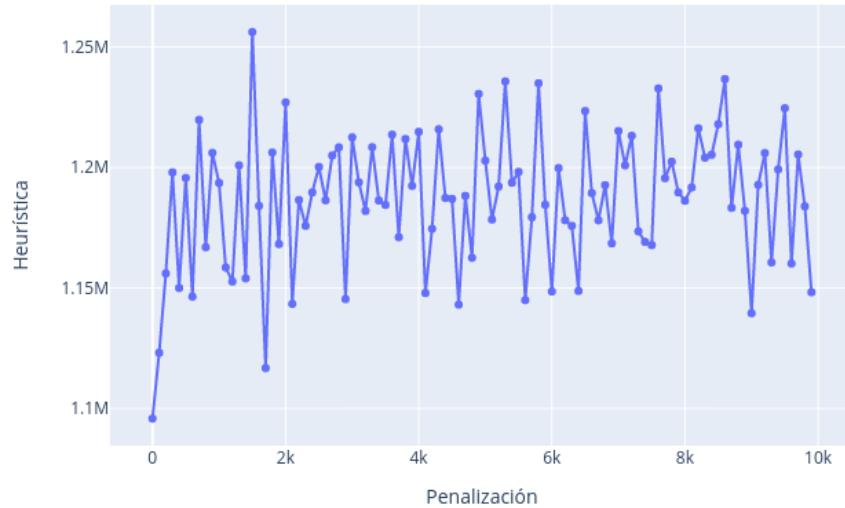


Figura 20: Resultados del heurístico en función de la penalización con Simulated Annealing

Estos gráficos han sido generados con la nueva heurística, esta consiste en aplicar una penalización mayor o menor dependiendo del número de clientes garantizados que tenga el estado. Es decir, la heurística tiene la siguiente formula:

$$\text{heurística} = \min(-\text{beneficio} + \text{penalización} * (\text{CLG_MAX} - \text{clg_estado}))$$

Dónde $CLG = \text{Clientes Garantizados}$. Con esta heurística un estado con más clientes garantizados asignados tendrá una mejor heurística.

6. **Conclusión** Observando los gráficos del Hill Climbing podemos ver cómo con poca penalización el algoritmo es muy rápido aunque no tiene buenas heurísticas. A medida que vamos subiendo la penalización por soluciones no válidas el algoritmo va tardando más aunque también termina con mejores heurísticos. A partir de una **penalización = 4400** se asegura que la solución final siempre sea válida. De primeras puede parecer que no tiene mucha lógica que con una menor penalización haga que la heurística sea tan mala. Esto tiene todo el sentido del mundo, ya que los operadores elegidos en los primeros experimentos no están capacitados para mejorar soluciones no válidas lo que hace que aunque ahora haya un mayor espacio de soluciones, esto no significa que consigamos un mayor heurístico. Esto puede ser porque igual el espacio de soluciones tiene muchos mínimos locales y al no penalizar las soluciones no válidas, nos vamos directamente hacia mínimos locales. Para que sea más fácil ver esta idea representamos la siguiente figura:

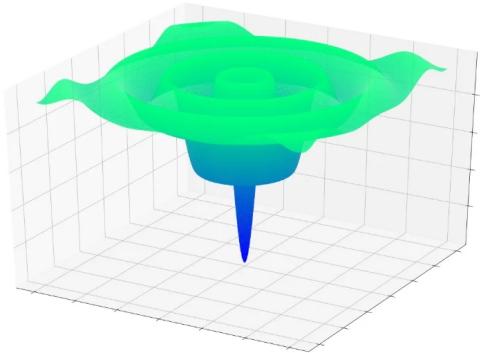


Figura 21: Figura con mínimos locales y un mínimo global

Supongamos que esta figura representa el espacio de soluciones de nuestro problema, sin una alta penalización nuestro algoritmo se puede estar quedando en los mínimos locales (en color verde) en vez de ir hacia el mínimo global (en color azul).

Comentando el Simulated Annealing podemos ver como actúa de una manera totalmente diferente, la penalización que aplicamos no afecta ni a la heurística, ni al tiempo ni al número de clientes garantizados asignados.

2.6. Experimento 6

En este experimento se estudia como afecta el aumento de centrales tipo C con el uso de las demás centrales. También se estudia como varia el coste en tiempo en las diferentes ejecuciones.

1. **Observaciones** Por las características de cada tipo de central, es posible que añadiendo más centrales de tipo C se podrían usar menos las centrales de tipo A y B.
2. **Planteamiento** Para realizar este experimento comparamos 3 escenarios: el original (experimento 1), duplicando y triplicando las centrales de tipo C.
3. **Hipótesis** Partimos de la hipótesis que el uso de más centrales de tipo C reducirá el uso de las otras centrales, independientemente del algoritmo que apliquemos. Esto es porque las centrales C estarán más repartidas y permitirán ahorrar pérdidas energéticas, y, en consecuencia, darán más beneficios.
4. **Metodología**
 - Usaremos el mismo escenario que el experimento 1
 - Aplicaremos el algoritmo del Hill Climbing y el de Simulated Annealing
 - Duplicamos el número de centrales de tipo C
 - Aplicaremos el algoritmo del Hill Climbing y el de Simulated Annealing
 - Triplicamos el número de centrales de tipo C
 - Aplicaremos el algoritmo del Hill Climbing y el de Simulated Annealing
 - Comparemos para cada algoritmo como varian los resultados para cada escenario y el coste temporal de cada uno de ellos.

5. Resultados

Gráfico de aumento de centrales de tipo C con usando Hill Climbing

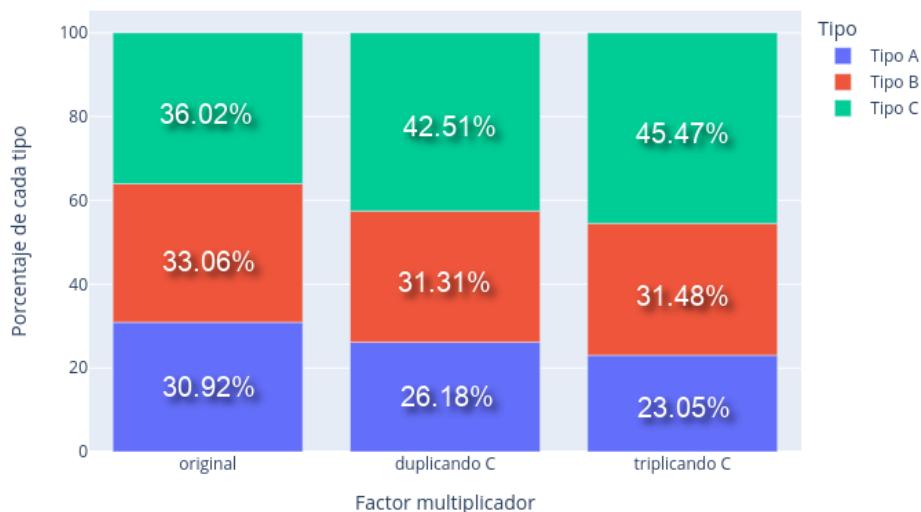


Figura 22: Resultados de usar más centrales de tipo C en Hill Climbing

Gráfico de aumento de centrales de tipo C usando Simulated Annealing

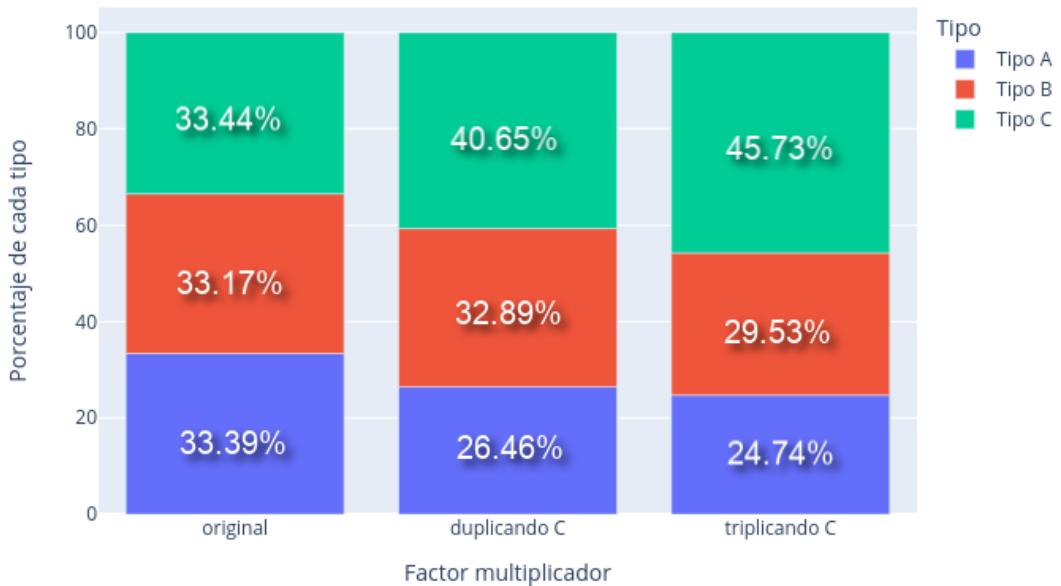


Figura 23: Resultados de usar más centrales de tipo C en Simmulated Annealing

Gráfico coste temporal usando Hill Climbing



Figura 24: Coste temporal del aumento de centrales tipo C en Hill Climbing

Gráfico coste temporal usando Simulated Annealing



Figura 25: Coste temporal del aumento de centrales de tipo C en Simmulated Annealing

6. Conclusión Observando la [Fifura 22](#) y la [Figura 23](#) podemos empezar por destacar que los resultados en ambos casos son muy parecidos, y, se podrían comentar a la vez. En el caso del escenario original el uso de cada central es más o menos equitativo, sobretodo cuando se usa el algoritmo de Simulated Annealing. En el segundo escenario, usando el doble de centrales de tipo C, se observa que la proporción cambia un poco. Las centrales de tipo A, las que más producen, se usan menos que las centrales de tipo B y mucho menos que las centrales de tipo C, que se usan mucho más. En el último caso, trabajando con el triple de centrales C, se aprecia más la diferencia entre el porcentaje de cada tipo de central. Las centrales de tipo C son las más usadas con diferencia, casi por igual con los dos algoritmos. Tanto las centrales de tipo B como las de tipo A, sobretodo estas últimas, se usan cada vez menos. Si comentamos la [Figura 24](#) y la [Figura 25](#), podemos ver como el tiempo crece de manera lineal dependiendo de cuantas centrales de tipo C haya. En conclusión, los resultados confirman nuestra hipótesis y nos demuestran un claro beneficio en usar más las centrales de tipo C, ahorrando así posibles pérdidas de energía.

3. Conclusiones

En este trabajo hemos podido obtener un conocimiento mucho más profundo de como funcionan los diferentes algoritmos de búsqueda local, en este caso Hill Climbing y Simulated Annealing. Hemos podido ver que aunque este problema sea una simplificación de un problema que podría ser de la vida real estos algoritmos con una buena especificación e implementación funcionan bien. En especial hemos podido ver como el Simulated Annealing llega a unas heurísticas muy parecidas a las del Hill Climbing en mucho menos tiempo.

Otro aspecto positivo que sacamos de esta práctica son las diferentes tecnologías usadas que nos ha hecho aprender más de ellas:

- **Java**: Lenguaje de programación usado para la implementación de toda la práctica.
- **Python**: Lenguaje de programación que hemos usado para generar todas las gráficas y que también hemos usado para hacer diferentes scripts que nos han facilitado el trabajo.
- **Google Colab**: Usado para generar algunas otras gráficas.
- **R**: Lenguaje de programación usado para hacer alguna prueba estadística.
- **GitHub**: Usado para tener un control de versiones y así poder trabajar varias personas en el mismo proyecto.
- **LATEX**: Usado para escribir y documentar todo el proceso y los diferentes experimentos de la práctica.

En conclusión, creemos que hemos hecho una buena práctica y estamos orgullosos del resultado final.

4. Trabajo de innovación: Alexa

4.1. Breve descripción

Alexa es un sistema de asistencia virtual desarrollado por *Amazon*. Inspirado por series de televisión y películas el proyecto se anunció el noviembre del 2014. Alexa utiliza el sistema de *Conversational AI*, basado en *NLP*, natural language processing, un proceso que analiza y interpreta el habla.

4.2. Reparto del trabajo

- **Mark Smithson** Se encargará de describir el funcionamiento a nivel técnico y como funciona *Alexa*.
- **Adrian Cristian Crisan** Se encargará de junto con Mark de explicar el funcionamiento y las bases de esta tecnología. También hará una breve introducción de la historia que hay detrás de *Alexa*.
- **Priyanka Amarnani** Se encargará de buscar toda la información y hacer un informe con datos estadísticos y numéricos del impacto medioambiental que tiene *Alexa*.
- **Javier Castaño** Se encargará de hacer un informe completo del impacto social que esta tecnología supone.

4.3. Referencias

- Introducción a *Alexa* [1]
- ¿ Cómo funciona *Alexa* ? [2]
- Coste de las AI's [4]
- Machine Learning detrás de *Alexa* [3]
- ¿ Cómo de inteligente es *Alexa* ? [7]
- Ética detrás de AI como *Alexa* [5]
- Cómo las AI's ayudan al medioambiente [6]

4.4. Dificultades

Las mayores dificultades de este trabajo las estamos encontrando cuando queremos ver datos y costes reales de los diferentes impactos, tanto sociales como medioambientales.

Referencias

- [1] Alexa. Alexa — amazon. <https://developer.amazon.com/en-US/alexa/alexa-skills-kit/conversational-ai>. [Online; accessed 08-October-2022].
- [2] Alexandre Gonfalonieri. How amazon alexa works? your guide to natural language processing (ai). <https://towardsdatascience.com/how-amazon-alexa-works-your-guide-to-natural-language-processing-ai-7506004709d3>. [Online; accessed 13-October-2022].
- [3] Data Science Foundation International. The machine learning behind alexa's ai systems. <https://www.youtube.com/watch?v=Dkg1ULBASNA>. [Online; accessed 08-October-2022].
- [4] Elsabet Jones and Baylee Easterday. Artificial intelligence's environmental costs and promise. <https://www.cfr.org/blog/artificial-intelligences-environmental-costs-and-promise>. [Online; accessed 14-October-2022].
- [5] Jai Kotia and Rishika Bharti. Ai ethics: Personal assistants like alexa, siri and google home. <https://chatbotslife.com/ai-ethics-personal-assistants-like-alexasiri-and-google-home-e41af2f6c315>. [Online; accessed 07-October-2022].
- [6] Jenny Medeiros. How voice assistants are helping the environment—and how they could do more. <https://www.voicesummit.ai/blog-old/how-voice-assistants-are-helping-the-environment-and-how-they-could-do-more>. [Online; accessed 14-October-2022].
- [7] Randrita Sarkar. How intelligent is alexa with conversational ai? <https://techfastly.com/how-intelligent-is-alexa-with-conversational-ai>. [Online; accessed 09-October-2022].