UNIVERSITAT POLITÈCNICA DE CATALUNYA

Práctica 1

INTELIGENCIA ARTIFICAL

Búsqueda Local

Autores:

Marc González Vidal

Jeremy Comino Raigón

Joan Caballero Castro

Profesor:

Javier Béjar

Q1 2022/2023

IBERDROLA

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

ÍNDICE

1.Introducción

La finalidad de este documento es resolver un problema al que se podría enfrentar una empresa que se dedica a la suministración de energía a una población: la asignación de unos clientes a unas centrales de forma que genera el mayor beneficio posible a la empresa.

Para resolver este problema utilizaremos métodos de inteligencia artificial basados en búsqueda local, donde vamos a explorar un espacio de soluciones de nuestro problema a través de un conjunto de operadores y usando un heurístico para guiarnos.

2.Descripción del problema

2.1 Descripción formal del problema

2.1.1 Elementos de entrada

Trataremos con un problema que definiremos formalmente como el siguiente:

- El problema considera un área cuadrada de 100x100 Km^2
- Existen N centrales, con sus respectivas coordenadas (x,y) dentro del área cuadrada. Hay 3 tipos de centrales:

  Tipo A → 250 MW ≤ Prod ≤ 750MW ∧ Coste encendida=Prod*50+20000€ ∧ Coste parada=15000€

  Tipo B → 100 MW ≤ Prod ≤ 250MW ∧ Coste encendida=Prod*80+10000€ ∧ Coste parada=5000€

  Tipo C → 10 MW ≤ Prod ≤ 100MW ∧ Coste encendida=Prod*150+5000€ ∧ Coste parada=1500€

- Existen M clientes, con sus respectivas coordenadas (x,y) dentro del área cuadrada. Hay 3 tipos de clientes:

  Tipo XG → 5MW ≤ Consumo ≤ 20MW ∧ (Garantizado → Tarifa=400€/MW ˅ No Garantizado → Tarifa=300€/MW) ∧ Indemnización=50 euros/Mw

  Tipo XG → 2MW ≤ Consumo ≤ 5MW ∧ (Garantizado → Tarifa=500€/MW ˅ No Garantizado → Tarifa=400€/MW) ∧ Indemnización=50 euros/Mw

  Tipo XG → 5 MW ≤ Consumo ≤ 20MW ∧ (Garantizado → Tarifa=600€/MW ˅ No Garantizado → Tarifa=500€/MW) ∧ Indemnización=50 euros/Mw

- Un cliente puede ser:

  Garantizado → Se le debe suministrar energía obligatoriamente.

  No garantizado → Se le puede dejar sin suministro y pagar la indemnización.

- Dado un cliente C i una central C' tendremos pérdidas por el transporte de la electricidad como describimos a continuación:

  0KM < distancia_euclidiana(C,C') <= 10KM → Pérdida=0%

  10KM < distancia_euclidiana(C,C') <= 25KM → Pérdida=10%

  25KM < distancia_euclidiana(C,C') <= 50KM → Pérdida=20%

  50KM < distancia_euclidiana(C,C') <= 75KM → Pérdida=40%

  75KM < distancia_euclidiana(C,C') → Pérdida=60%

Durante la asignación, obviamente el sumatorio de todos los consumos de los clientes, una vez tenido en cuenta las pérdidas por distancia, debe ser menor o igual que la producción que tiene esa central.

2.1.2 Cálculo del balance de la empresa

Para calcular el balance de la empresa tendremos en cuenta los siguientes factores:

- **Ingresos por las tarifas** que los clientes suministrados nos van a pagar.
- **Costos de mantenimiento** de las centrales en caso de que no estén funcionando.
- **Costos de producción** de las centrales en caso de que estén activas.
- **Pérdidas por indemnizaciones** a los clientes no garantizados que no suministramos.

2.1.3 Objetivo del algoritmo

El objetivo del algoritmo va a ser dar un conjunto de asignaciones sobre los clientes a las centrales, donde se van a intentar optimizar los siguientes criterios:

- Maximizar el balance de la empresa. El balance de la empresa puede ser definido de la siguiente forma:

Balance = Ingresos tarifas – Costos mantenimiento – Costos producción - Indemnizaciones

- Minimizar las pérdidas de electricidad por el transporte de la misma desde la central al cliente. Para hacerlo vamos a asignar a clientes centrales que estén más cerca.

2.1.4 Descripción de la solución

Una asignación correcta tiene que satisfacer el siguiente conjunto de restricciones:

- Todos los clientes garantizados deben recibir suministro energético.
- Un cliente no garantizado sin suministro debe recibir una indemnización.
- Todos los clientes suministrados deben pagar su tarifa correspondiente.
- Una central no puede suministrar más energía que su capacidad.
- Una central activa genera toda su capacidad, independientemente de si toda la energía es usada o no.

- Cualquier central activa o inactiva debe reflejar los costes en el balance de la empresa.
- Un cliente solo puede estar suministrado por una única central.

2.2 Justificación del uso de algoritmos de Búsqueda Local

Vemos que este problema trata sobre asignar un conjunto de clientes a un conjunto de centrales, de forma que todo el espacio de búsqueda es inmensamente grande. Si intentamos buscar la solución óptima al problema con algoritmos más tradicionales, nos damos cuenta de que vamos a necesitar una cantidad de memoria y tiempo del que no disponemos.

Además, nos damos cuenta de que una vez encontrada una solución podemos explorar todo el espacio de soluciones usando los operadores correspondientes, y si además nos aseguramos de que se sigan respetando el conjunto de restricciones anteriormente descrito, podemos demostrar que no nos salimos del espacio de soluciones en ningún momento.

Es por esto que pensamos que aplicar algoritmos de Búsqueda Local, como es el caso de Hill Climbing o Simulated Annealing, son algoritmos que nos pueden llevar a una solución buena al problema. Obviamente estamos sacrificando el encontrar la más óptima por tiempo y memoria, pero de hecho con estos algoritmos seremos capaces de encontrar una solución óptima lo suficientemente buena, de hecho, la mejor localmente.

3. Diseño de los elementos clave de la búsqueda

3.1 Representación del estado

El estado se considerará como el conjunto de asignaciones de los clientes a las centrales. Solo con esto ya podremos representar la solución inicial que tenemos para empezar a explorar el estado de soluciones, siempre y cuando se satisfacen las restricciones que debe tener una solución descrita anteriormente.

Cada estado tendrá un heurístico que se tratará de maximizar, posteriormente profundizaremos sobre el heurístico escogido para llegar a la mejor solución local del problema.

3.1.1 Espacio de soluciones

Vamos a examinar ahora, dando una cota superior, el tamaño del espacio de soluciones de nuestro problema. Sabemos que nuestras soluciones están formadas por un conjunto N de clientes, que a su vez están formados por un conjunto X de clientes garantizados y un conjunto Y de clientes no garantizados. Sabemos que un cliente garantizado tiene que tener una central asignada, por lo que en el estado no puede existir un cliente garantizado que no tenga central, así el número de combinaciones posibles entre clientes garantizados y las centrales es $O(|\text{Centrales}|^{|\text{clientes garantizados}|})$. Por otro lado, examinemos al conjunto de clientes no garantizados. Si nos damos cuenta, su análisis es el mismo que el de los clientes garantizados, pero teniendo en cuenta de que puede ser que no tengan central. Esto se puede ver como que existe otra central que es central inexistente a la que solo pueden ir los clientes no garantizados que no van a tener asignación a central, con lo cual el número de combinaciones posibles entre clientes no garantizados y centrales es $O((|\text{Centrales}|+1)^{|\text{clientes no garantizados}|})$.

En conclusión, si tratamos de hacer la unión entre los dos conjuntos, nos sale que el espacio de soluciones de nuestro problema es el siguiente:

$$O((|Centrales|^{|clientes\ garantizados|}) + (|Centrales|+1)^{|clientes\ no\ garantizados|}))$$

3.1.2 Representación considerada

Para poder representar el estado correctamente hemos tenido en cuenta los siguientes atributos que lo definen:

- asignacion_clientes: es un vector de enteros que relaciona a los clientes y las centrales que les va a suministrar la energía. asignación_clientes[*i*], tiene por *i* el identificador del cliente y el valor que devuelve es el identificador de la central o -1 en caso de que al cliente no se le esté suministrando energía de ninguna central.

- numero_clientes_central: es un vector de enteros que guarda cuántos clientes hay asignados a cada central. numero_clientes_central[*i*], tiene por *i* el identificador de la central y el valor que hay en el vector es el número de clientes asignados que tiene una central en el estado actual. A priori puede parecer que este vector es prescindible, pero nos va a hacer falta para saber cuando podemos parar una central porque se ha quedado sin clientes al quitar un cliente de una central o la tenemos que activar porque anteriormente no tenía ningún cliente y hemos asignado un cliente a esa central.

- consumo_centrales: es un vector de doubles que guarda el consumo real de una central que está teniendo por parte de los clientes, es decir, es el sumatorio del consumo que están haciendo todos los clientes de esa central teniendo en cuenta las pérdidas de energía debido a la distancia en la que están. consumo_centrales[i], tiene por *i* el identificador de la central y devuelve el consumo en MW que se están usando actualmente de una central. Nótese que para estar en una representación de una solución correcta el valor que se guarda en este vector para cada central tiene que ser menor o igual que la producción que tiene por estar activa. Este vector se usará posteriormente para comprobar si a una central aún le queda energía suficiente como para suministrar a un cliente que estamos interesados en asignar a esa central.

- dinero: es un double que guarda el dinero total que esté ganando (en caso de que sea positivo) o gastando la empresa (en caso de que sea negativo) teniendo en cuenta todos los costes que tienen las centrales y los ingresos o los costes (en caso de que les tengamos que pagar indemnización por dejarlos sin suministro si son no garantizados) que nos den los clientes. Este atributo nos va a ser muy útil para el heurístico, ya que el objetivo de la práctica es maximizar los beneficios de la empresa.

Hemos utilizado esta representación del estado, ya que a nuestro parecer era lo mínimo que necesitábamos para poder representar todos los elementos del problema, y también que nos permitiese poder hacer todos los cálculos eficientemente. De hecho, gracias a esta representación, todos los cálculos que se hacen en los diferentes métodos del Estado una vez ya hemos generado la solución inicial (para generar las soluciones iniciales sí se usan métodos con O(n)) y estamos explorando el espacio de soluciones

es O(1), tratando así que la exploración que hacen nuestros algoritmos de búsqueda local sean los más rápidos posibles.

3.2 Solución inicial

Para la creación de la solución inicial hemos optado por la implementación de 2 estrategias:

- Asignar 1: Esta estrategia consiste en asignar a cada central clientes garantizados aleatoriamente hasta que la central no pueda dar suministro a ningún otro cliente. Dejando a todos los clientes no garantizados sin una central asignada.
- Asignar 2: Ordenamos de manera aleatoria todas las centrales y por cada central vamos asignándole los clientes garantizados que menos energía tendrían que suministrar y aún no han estado asignados. Por último, se intenta asignar al máximo número de clientes no garantizados utilizando el mismo criterio mencionado anteriormente.

Como se puede observar, el objetivo principal de Asignar 1 es la creación de una solución completamente aleatoria para asegurar que, con varias ejecuciones del algoritmo, llegaremos a tener diferentes resultados. Es decir, tener una solución inicial lo más indeterminista.
Por otra parte, Asignar 2 lo que intenta es asignar la mayor cantidad de clientes a las centrales. Nótese que, al hacer una ordenación aleatoria de las centrales, aún obtenemos una solución inicial completamente aleatoria. Todo debido a que un cliente no asignado C puede ser el que menos consumo real necesite en una central X y una central Y y todo dependerá de qué central esté primera en el orden dado.
Es más, sabemos que si cada ordenación nos da una solución inicial diferente al resto, podemos decir que como mucho tiene O(|Centrales|!) soluciones iniciales posibles.

3.3 Conjunto de operadores y condiciones de aplicabilidad

A continuación, vamos a definir el conjunto de operadores que nos van a permitir explorar nuestro espacio de búsqueda.

- asignar_cliente_a_central(c, cent): asigna al cliente *c* a la central *cent*. Si *cent* es -1 quiere decir que a *c* se le va a dejar sin suministro.
  Ramificación: O(|Clientes|*(|Centrales|+1)). Para cada cliente que haya en nuestro conjunto le podemos asignar una central incluyendo también la central con identificador -1 para dejar a un cliente sin suministro. Por lo que si hacemos esta combinatoria nos sale ese factor de ramificación.
- swap(c1, c2): intercambia la central que tiene el cliente *c1* y se la asigna a *c2* y viceversa.
  Ramificación: O(|Clientes|²). De hecho, sería $\frac{(|Clientes|)(|Clientes|-1)}{2}$, ya que cada cliente puede ser intercambiado por alguien que no sea él, y si representamos esta relación con un grafo donde los vértices son los clientes y las aristas son los intercambios posibles, nos sale un grafo completo, por lo que el factor de ramificación es el número de aristas de un grafo completo.
- cent_to_cent(ce1, ce2): asigna a todos los clientes que tiene la central *ce1* a la central *ce2*.
  Ramificación: O(|Centrales|*(|Centrales|-1). Podemos escoger cualquier central a la que podemos aplicar el operador y luego pasamos los clientes a cualquier otra central que no sea esa.

Decidimos escoger el primer operador porque básicamente es la primera operación que se te puede ocurrir a la hora de dar unas asignaciones sobre algún tipo de conjunto. Además, solo con el primer operador ya somos capaces de explorar todo el espacio de soluciones, y no solo el espacio de soluciones sino también todo el espacio de estados, aunque eso no nos interesa realmente. Es importante destacar que este operador no hace solo una “operación básica”, que en este caso sería la de asignar, sino que también es capaz de quitar un cliente de una central, por lo que realmente este operador se podría desglosar en dos operadores, un asignar que solo asigne y otro de quitar a un cliente de una central. Hemos decidido dejarlo así porque nos parecía interesante tener un operador que nos diese la capacidad de explorar todo el espacio de soluciones.

Como ya teníamos un operador que nos permitía explorar todo el espacio de soluciones, nos costó decidirnos por elegir otro operador que nos pudiese ser interesante. Al final optamos por escoger un swap por una simple razón: el primer operador sí que nos permitía bastante bien el ir mejorando el dinero obtenido por la empresa, pero era bastante ineficiente para hacer que las elecciones hechas por el algoritmo fuesen eficientes a la hora de intentar que la energía perdida fuese la mínima posible, y para esto el swap nos ofrecía una posibilidad muy buena. De hecho, a priori puede parecer que un swap entre 2 clientes garantizados no tiene sentido, pero no es así, ya que aunque los dos van a seguir pagando la tarifa, si conseguimos que se pierda menos energía durante el transporte, estamos ganando capacidad libre en la central, por lo que, aunque el dinero sea el mismo, sí que estamos mejorando el estado. Ya se han hecho algunos avances del heurístico, pero se explicará con detalle más adelante.

También nos resultó interesante un operador que nos permitiese mover muchos clientes solo con un uso del operador, y para eso hicimos el cent_to_cent, aunque ya veremos que con la condición de aplicabilidad realmente va a ser un operador que se podrá usar muy pocas veces.

Obviamente, tal y como están descritos los operadores, nos podemos salir del estado de soluciones, cosa que no nos interesa. Es por eso que a la hora de comprobar sus condiciones de aplicabilidad nos aseguramos de que al aplicar el operador no nos salgamos del espacio de soluciones. Para hacer esto hemos añadido 3 métodos a la clase Estado.

El primero es move_efectivo, dónde nos aseguramos que dado un cliente y una central podemos hacer esa asignación asegurándonos de que no dejamos a ningún cliente garantizado sin suministro. También nos aseguramos de no dejar al cliente en la misma central, ya que eso nos volvería a generar el mismo estado. Finalmente nos aseguramos de que la demanda de energía del cliente pueda ser suplida por la central sin que esa se exceda de su producción.

El segundo es el swap_efectivo, dónde primero nos aseguramos de que el swap se va a hacer con clientes del mismo tipo de contrato. Posteriormente, miramos que el swap genere algún cambio al estado, es decir, que las centrales no sean las mismas. Finalmente nos aseguramos de que las centrales sean capaces de suplir la demanda de los clientes, teniendo en cuenta que dejan de servir al cliente anterior.

El tercero es el cent_to_cent_efectivo, dónde se comprueba que la segunda central sea capaz de absorber toda la demanda energética sumando las pérdidas que hay en el transporte de todos los clientes de la primera central.

Con los operadores y métodos descritos anteriormente nos aseguramos de que todas las restricciones de nuestro problema se siguen cumpliendo y podemos explorar todo el espacio de soluciones para encontrar una solución óptima.

3.4 Función Heurística

El enunciado de la práctica trata de una empresa que quiere maximizar su beneficio, por lo que está claro que, de un modo u otro, dentro de la función heurística tendremos que darle un peso al balance de la empresa que tenga en el estado correspondiente, intentando maximizarlo positivamente.

Inicialmente y solo con el primer operador, se nos ocurrió que hacer un heurístico que solo tuviera en cuenta el balance de la empresa sería suficiente, y de hecho no funcionaba mal, ya que terminaba llegando a una solución con beneficios. De hecho, es obvio que llegará a una buena solución, ya que siempre está escogiendo el hijo de mayor beneficio (en caso de Hill Climbing).

Posteriormente, al introducir el swap estuvimos pensando si realmente era efectivo el hacer swap con garantizados, ya que de la forma que hemos definido el heurístico hasta ahora no tiene sentido, ya que no lo modifica (los clientes pagan la misma tarifa y al ser garantizados se asegura de que van a tener central). Por lo que fue en este momento en el que se nos ocurrió tener en cuenta otro factor en el heurístico, que sí se veía modificado con el swap de garantizados. Este factor es tener en cuenta el dinero que nos ahorramos al no perder energía durante el transporte, haciendo que el cliente esté cerca de la central.

De ese modo un swap entre garantizados sí que era interesante, ya que nos dejaba un espacio de energía no consumido en una central que no era perdido por el transporte, pudiendo ser aprovechado para suministrar a otro cliente. Además, nos dimos cuenta de que este factor también era interesante para los clientes no garantizados y para el otro operador, por lo que decidimos añadirlo al heurístico, obteniendo así mejores resultados.

Por lo que el heurístico lo podemos definir de la siguiente forma:

$$h(x) = A - BC$$

Donde:

A = Balance de la empresa (€).

B = Energía Perdida en el Transporte (MW).

C = Coste de producción de energía en la central (€/MW).

Es importante destacar que el heurístico es consistente, ya que estamos operando con las mismas unidades, en este caso euros.

4. Experimentación

4.1. Entorno de experimentación

Los experimentos se han realizado en una máquina virtual de las siguientes características:

- **Máquina virtual**
  - **Procesador:** AMD Ryzen 7 5800H with Radeon Graphics 3.20 GHz.
  - **Número de procesadores:** 4.
  - **Memoria RAM:** 10 GB.
  - **Sistema Operativo:** Ubuntu 22.04.1 LTS.

- **Otros datos**
  - **IDE:** Apache Netbeans IDE 15.
  - **Memoria asignada:** 9216 MB.
  - **Cálculo del tiempo de ejecución:** Librería System de Java (método currentTimeMillis)

4.2 Experimento 1: Elección de los Operadores

Para la elección de los operadores hemos tenido en cuenta 2 factores:

- El tiempo de ejecución del programa.
- El número de veces de utilización de estos.

Para nuestro experimento hemos utilizado la siguiente configuración:

- 40 centrales en total de las cuales 5 son de clase A, 10 de clase B y 25 de clase C.
- 1000 clientes de los cuales el 25% son XG, 30% es MG y 45% son G. Todo esto combinado a que hay un 75% de clientes garantizados.
- Semillas aleatorias con valores posibles [0,500).
- Se utiliza la estrategia de asignación Asignar2.

Viendo este escenario, optamos por evaluar los operadores en grupos de dos y ver el rendimiento que tienen en conjunto. Por cada conjunto de 2 operadores haremos la evaluación con 5 seeds diferentes y obtendremos los tiempos. Por otra parte, miraremos la proporción de utilización de los diferentes operadores para llegar a la solución final.

Ahora podemos ver los siguientes resultados:

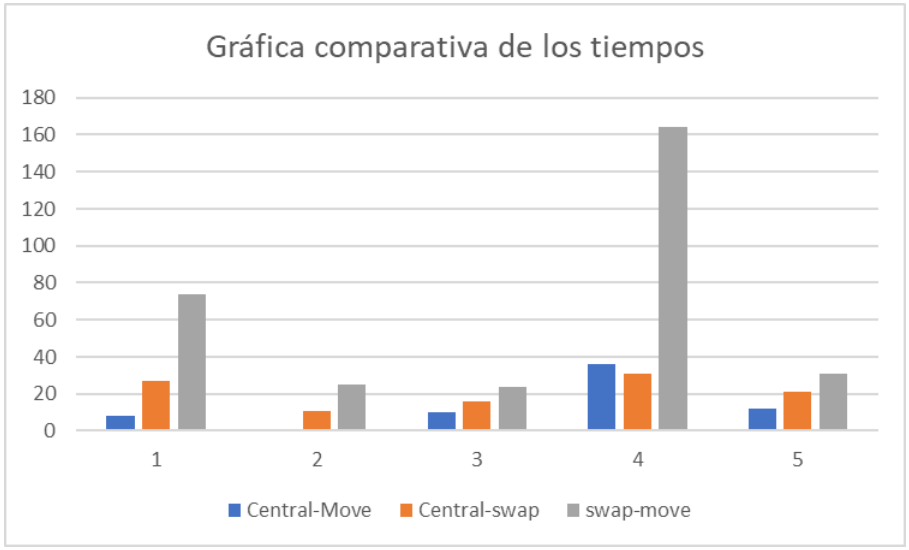

Figura 1: En esta tabla se pueden ver los tiempos de cada ejecución del algoritmo con cada configuración de operadores. Aclaramos también que el eje Y al ser la representación de los tiempos, tenemos como unidades los segundos.

Como podemos observar en la gráfica mostrada, la aplicación del operador cent_to_cent, el cual lo que hace es asignar todos los clientes de una central A a otra Central B, si esta puede dar suministro a todos, ayuda a los tiempos de ejecución del programa, pero si contrastamos esta información con la proporción del uso de este operador para llegar a la solución podemos ver el siguiente resultado:

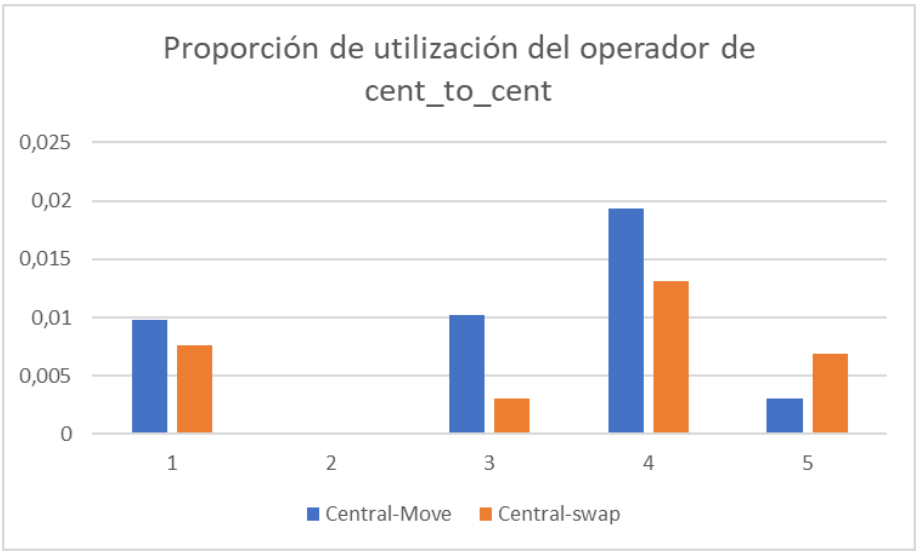

Figura 2: Proporción en tanto por uno de la proporción de la utilización del operador cent_to_cent.

Como se puede ver, la proporción de este operador es minoritaria. Es decir, no tiene ninguna relevancia en la construcción de la solución final.
A diferencia del conjunto Move-Swap donde por ejemplo el operador de move tiene mayor proporción. Esto es demostrado empíricamente con la siguiente prueba de hipótesis:

```
        Paired t-test

data:  prop1$V1 and prop2$V1
t = -4.0888, df = 4, p-value = 0.007495
alternative hypothesis: true mean difference is less than 0
95 percent confidence interval:
       -Inf -0.1188154
sample estimates:
mean difference
     -0.2482487
```

Figura 3: Prueba donde se mira si la prop1 (proporción de cent_to_cent en el caso de la combinación con el operador swap) es menor que la de prop2 (proporción del move en el caso la combinación con el operador swap).
Se puede observar que podemos afirmar que se utiliza en mayor medida el operador de move que el operador de cent_to_cent en la creación de las soluciones finales.

Basándonos en estos datos, hemos optado por la utilización de los operadores de Move y Swap para el resto de experimentos. Todo esto debido a que son operadores que pueden llegar a ser utilizados en mayor medida en la creación de soluciones y por lo tanto, a cambio de un cierto tiempo de ejecución superior a los casos con el operador cent_to_cent, podemos llegar a explorar de mejor manera el espacio de soluciones, y en consecuencia llegar a una solución mejor.

4.3 Experimento 2: Generación de la Solución Inicial

Este experimento tiene como finalidad encontrar el mejor generador de solución inicial que hemos implementado. Queremos un algoritmo eficiente que nos encuentre una solución óptima en un tiempo razonable.

Este tipo de problemas tiene una gran cantidad de espacio de búsqueda y de soluciones, y no queremos que visite todas y cada una de ellas, sino que se sepa guiar entre las mejores y evite las que tengan peores resultados heurísticos. Por tanto, soluciones que nos tarden mucho tiempo y encuentren soluciones muy óptimas no son nuestro mayor objetivo, ya que también estamos intentando minimizar el tiempo de ejecución. En consecuencia, nuestros datos para comparar será el beneficio obtenido por el heurístico dividido por el tiempo que tarda en ejecutar el algoritmo.

Para hacer nuestro experimento hemos utilizado como plantilla la configuración de clientes y centrales del experimento 1.

Nuestro experimento consiste en medir el tiempo de ejecución y el valor del heurístico con 10 semillas diferentes y utilizando las dos asignaciones mencionadas anteriormente.

A continuación, se pueden ver los resultados obtenidos:

Semilla	Tiempo 1	Heurístico	Heurístico1 /	Tiempo 2	Heurístico	Heurístico2 /

	(s)	1 (M)	Tiempo1	(s)	2 (M)	Tiempo2
351716896	164	1,333817	0,008133030488	98	1,316395	0,01343260204
26711345	80	1,290823	0,0161352875	32	1,202483	0,03757759375
-831157754	196	1,299006	0,006627581633	42	1,24866	0,02973
726036	200	1,299808	0,00649904	156	1,261193	0,008084570513
174081775	228	1,406866	0,006170464912	214	1,345588	0,006287794393
-1800468193	151	1,364235	0,009034668874	98	1,317344	0,01344228571
44569704	49	1,24836	0,02547673469	20	1,150224	0,0575112
-1894025117	61	1,246514	0,02043465574	14	1,081702	0,07726442857
-1184119069	258	1,360676	0,005273937984	225	1,306621	0,005807204444
-1600185119	158	1,401772	0,008871974684	77	1,378077	0,0178971039
	154,5	**1,3251877**	**0,01126573765**	**97,6**	**1,2608287**	**0,02670347833**

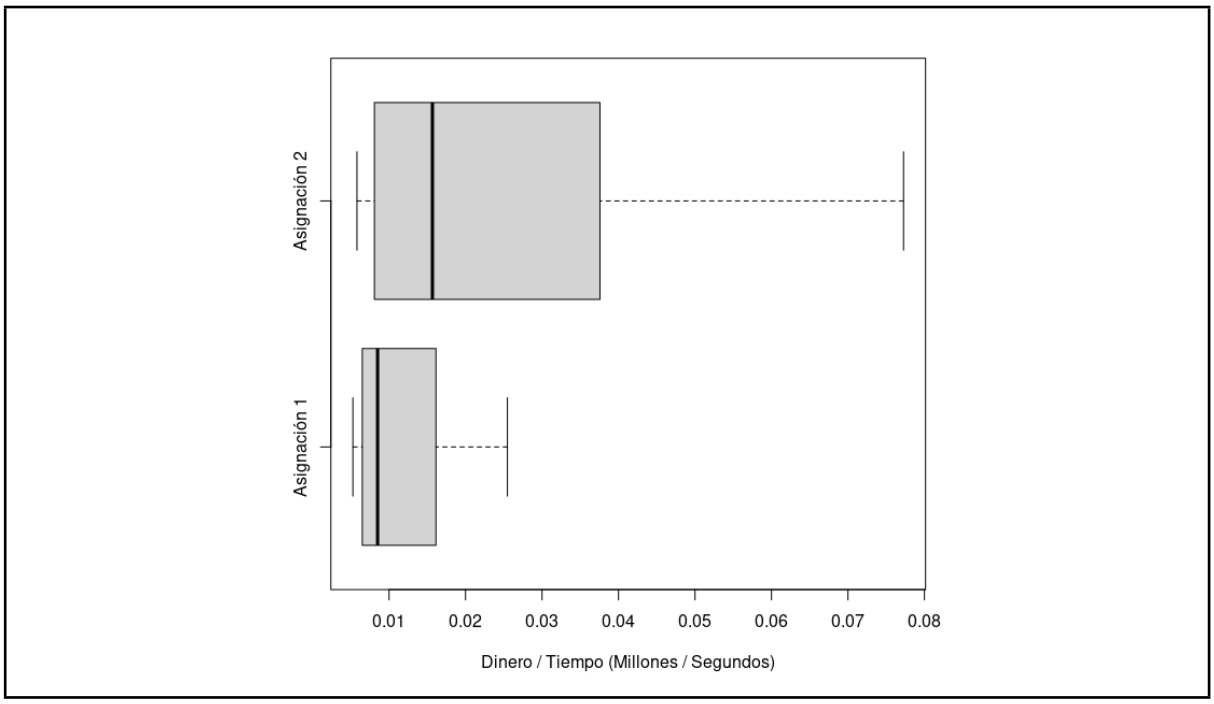

Figura 4: Comparativa de los valores de heurístico/tiempo obtenidos por la Asignación 1 y la Asignación 2.

Podemos observar por el Boxplot mostrado en la parte superior que Asignación 1 no tiene una gran variabilidad de valores, a la vez que no contiene unos valores mayores comparados por Asignación 2. En cambio, hemos visto que Asignación 2 ha obtenido un número variado de valores y se puede ver como de media, Asignación 2 obtiene una mayor cantidad de dinero por cada segundo que el algoritmo ha sido ejecutado. Para ver si es correcto hemos hecho una prueba de hipótesis donde probaremos si Asignación 2 nos ofrece mayor dinero por unidad de tiempo comparado con Asignación 1:

```
        Paired t-test

data:  profit2$V1 and profit1$V1
t = 2.6748, df = 9, p-value = 0.01271
alternative hypothesis: true mean difference is greater than 0
95 percent confidence interval:
 0.004858028         Inf
sample estimates:
mean difference
     0.01543774
```

Figura 5: Prueba de p-valor donde probamos si Asignación 2 (profit2) es mayor que Asignación 1 (profit1) con un 95% de confianza.

Como podemos observar, el valor obtenido *p-value* es menor al valor de confianza (en este caso 0.05). Por lo tanto, llegamos a la conclusión que Asignación 2 nos da mayores valores de heurístico/tiempo comparado con Asignación 1.
Debido a todo lo anteriormente mencionado, escogemos la Asignación 2 porque nos puede llegar a obtener resultados cercanos a los que podríamos tener con Asignación 1 pero ejecutando el algoritmo de búsqueda en menos tiempo.

4.4 Experimento 3: Simulated Annealing

En este experimento vamos a ajustar los parámetros de la búsqueda del Simulated Annealing para obtener mejores resultados.

Nuestro objetivo será optimizar el Simulated Annealing buscando unos valores óptimos para los siguientes 4 parámetros:

- steps: número total de iteraciones.
- stiter: número de iteraciones por cada cambio de temperatura (ha de ser un divisor de steps).
- k: determina el comportamiento de la función de temperatura.
- lambda: determina el comportamiento de la función de temperatura.

Debemos de ser cuidadosos a la hora de determinar los valores de estos parámetros, ya que tanto su asignación individual como la combinación entre los 4 podrá variar sustancialmente los beneficios obtenidos de los resultados.

- Los valores de steps y stiter nos determinan el número de iteraciones y su distribución.
- Los valores de k y lambda influyen en la función de energía y en la probabilidad de aceptar estados peores.

Para este experimento partiremos del mismo escenario y condiciones que en los experimentos anteriores:
- 5 centrales de tipo A, 10 de tipo B y 25 de tipo C.
- 1000 clientes con proporción de 25% (XG), 30% (MG) y 45% (G).
- Proporción del 75% de clientes con suministro garantizado.

Con los resultados obtenidos de los experimentos 1 y 2, usaremos la combinación de los operadores swap y move junto con el segundo método de generación de soluciones iniciales.

El primer paso será determinar los valores para k y lambda, ya que las variables steps y stiter dependrán de estos parámetros. Para este caso fijaremos unos valores predeterminados para steps y stiter.

Debemos destacar que el valor de steps deberá ser lo suficientemente grande como para que haya convergencia. Por este motivo, antes de empezar a experimentar con las variables k y lambda, nos queremos asegurar que con el valor de steps escogido la función converja.

Para esto haremos una experimentación previa donde realizaremos una serie de ejecuciones con valores distintos de steps y fijaremos el resto de variables al valor que tienen por defecto en la clase SimulatedAnnealingSearch, que son stiter = 100, k = 20 y lambda = 0.005.

Para ejecutar este primer experimento llamaremos a la función experimento3_steps(). En este experimento hemos explorado unos valores de steps entre 5.000 y 150.000. Escogimos un total de 6 semillas aleatorias distintas y para cada una de ellas hicimos un total de 6 ejecuciones con el algoritmo de SA, una para cada valor de steps.

Decidimos ejecutar también el algoritmo de Hill Climbing con las mismas condiciones que el Simulated Annealing para así tener una referencia del beneficio y el tiempo obtenido de los dos. De esta manera fijaremos un steps inicial para el resto de experimentos, y una vez obtenidos los valores del resto de variables volveremos a calcular el valor óptimo de steps.

Esta será la manera en la que haremos este y el resto de experimentos:
- Elegiremos aproximadamente 6 semillas aleatorias, una para cada réplica. Dependiendo del tiempo que tarde en hacer el experimento hemos realizado más o menos repeticiones.
- Para cada experimento, todas las réplicas tendrán un número de ejecuciones fijado. Esto dependerá del número de valores que queramos explorar para cada experimento. En este primero, por ejemplo, haremos 6 ejecuciones por réplica, ya que queremos explorar 6 valores diferentes de steps.
- Para cada experimento guardaremos los valores y parámetros que nos sean necesarios para su estudio posterior. Unos valores que guardaremos siempre independientemente del experimento serán la semilla usada, el beneficio obtenido (nuestra heurística) y el tiempo de ejecución.

steps	beneficioSA	beneficio HC
5.000	1.139.135,5	1.250.695
10.000	1.170.508,0	1.250.695
50.000	1.263.820,2	1.250.695
80.000	1.281.475,8	1.250.695
100.000	1.294.906,2	1.250.695
150.000	1.296.791,2	1.250.695

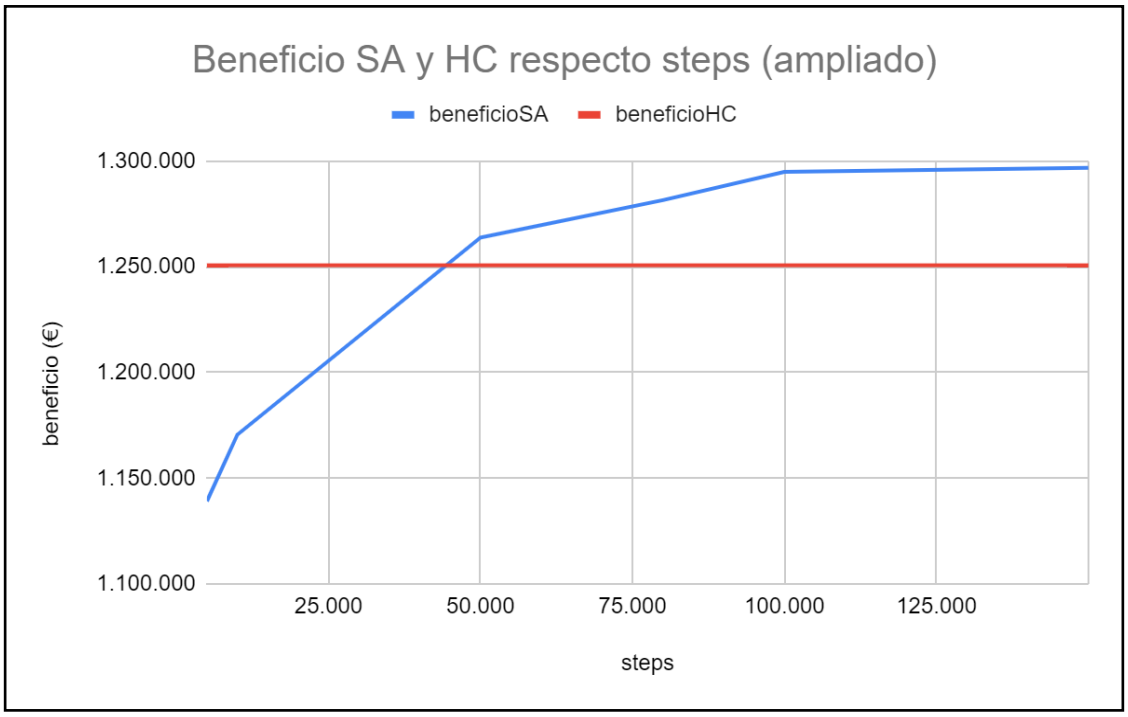

Figura 6: Gráfico ampliado de la comparación previa del valor del heurístico de SA y HC respecto al número de iteraciones totales

Observando la tabla y el gráfico llegamos a la conclusión que 100.000 es un valor adecuado para steps, ya que a partir de ese número el heurístico prácticamente no varía y, en cambio, aumenta el tiempo de ejecución. Por tanto, para los siguientes experimentos utilizaremos un valor de steps de 100.000.

El siguiente paso es encontrar unos valores óptimos para k y lambda. Procederemos a ejecutar la función experimento3_k_lambda1() con el valor de stiter por defecto.
Para la primera versión utilizamos unos valores extremos de k = 1, 5, 25, 125 y lambda = 1, 0.01, 0.0001. Observando la tabla y el gráfico detectamos que el valor máximo se encuentra con k = 0.0001 y lambda = 125.

Para todo lambda = 1 parece haber malas soluciones. Con 0.01 parece encontrar buenos resultados y a medida que vamos aumentando k parece ir maximizándose lentamente.

Cabe destacar que las diferencias en el gráfico son muy significativas (el máximo es 1.326.381,60 y el mínimo es 1.119.856,80).

lambda	k	beneficio
1	1	1.125.231,60
1	5	1.123.856,40
1	25	1.121.907,20
1	125	1.119.856,80
0,01	1	1.318.208,20
0,01	5	1.320.732,80
0,01	25	1.321.162,60
0,01	125	1.319.702,20
0,0001	1	1.320.398,60
0,0001	5	1.324.219,40
0,0001	25	1.326.264,40
0,0001	125	**1.326.381,60**

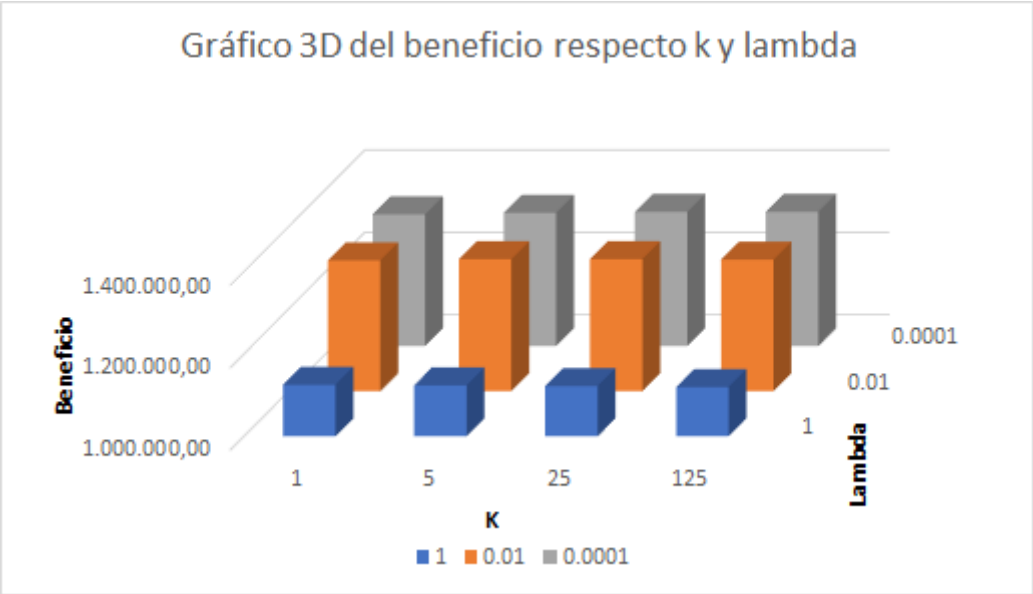

Figura 7: Gráfico 3D del beneficio obtenido respecto a los parámetros k y lambda (1)

Repetiremos el procedimiento anterior acotando valores alrededor del último caso, ahora con la función experimento3_k_lambda2(). Dado que a medida que disminuía lambda parecía encontrar mejores soluciones, ahora utilizaremos valores en torno al mejor resultado encontrado anteriormente. Por esto ahora asignaremos a lambda los valores 0.001, 0.0001 y 0.00001. De la misma manera, investigaremos los valores de k cercanos a 125 que es donde encontramos el máximo anterior, por tanto, ahora k tendrá los valores 25, 125 y 625.

lambda	k	beneficio
0,001	25	**1.341.950,86**
0,001	125	1.337.337,86
0,001	625	1.338.834,43
0,0001	25	**1.340.402,14**
0,0001	125	1.339.209,71
0,0001	625	1.339.037,71
0,00001	25	1.338.540,71
0,00001	125	1.338.229,43
0,00001	625	1.338.309,86

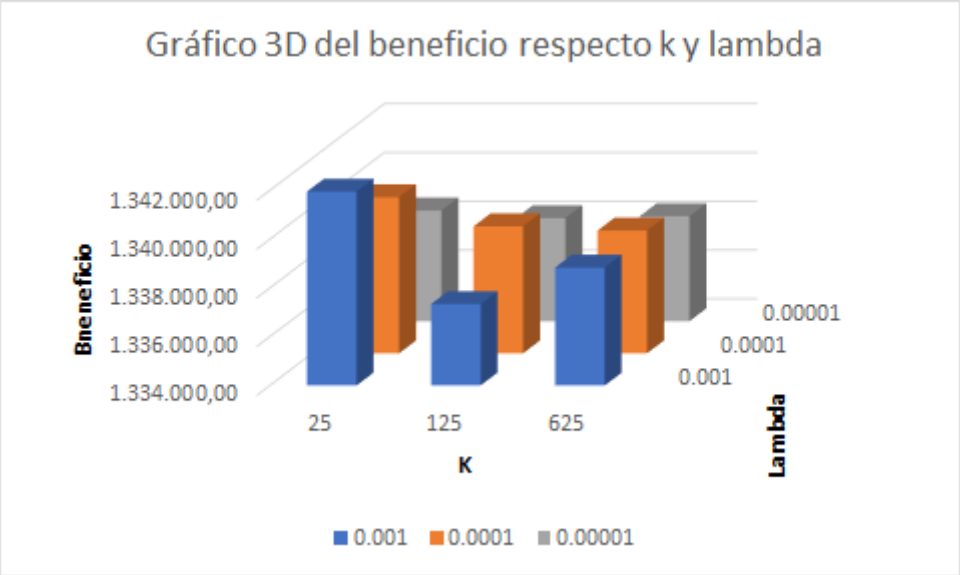

Figura 8: Gráfico 3D del beneficio obtenido respecto a los parámetros k y lambda (2)

Ahora notamos que los mejores resultados los encontramos con un valor de k = 25 en torno a una lambda de 0.001 y 0.0001. Notamos que aún hay cierta variabilidad (rango de 5.000 aproximadamente), pero menos que antes: la desviación estándar se ha reducido.

Hemos decidido ajustar más los valores y hacer una última experimentación con los valores medios de estos parámetros para encontrar unos valores más óptimos. Estos son los valores obtenidos después de llamar a la función experimento3_k_lambda3():

lambda	k	beneficio
0,001	15	1.300.098,67
0,001	30	1.295.914,33
0,001	60	1.297.305,33
0,001	90	1.298.560,17
0,0005	15	1.295.654,17
0,0005	30	1.299.844,50
0,0005	60	1.297.193,83
0,0005	90	1.297.068,67
0,0001	15	1.299.404,67
0,0001	30	**1.303.604,00**
0,0001	60	1.301.230,83
0,0001	90	1.300.216,17

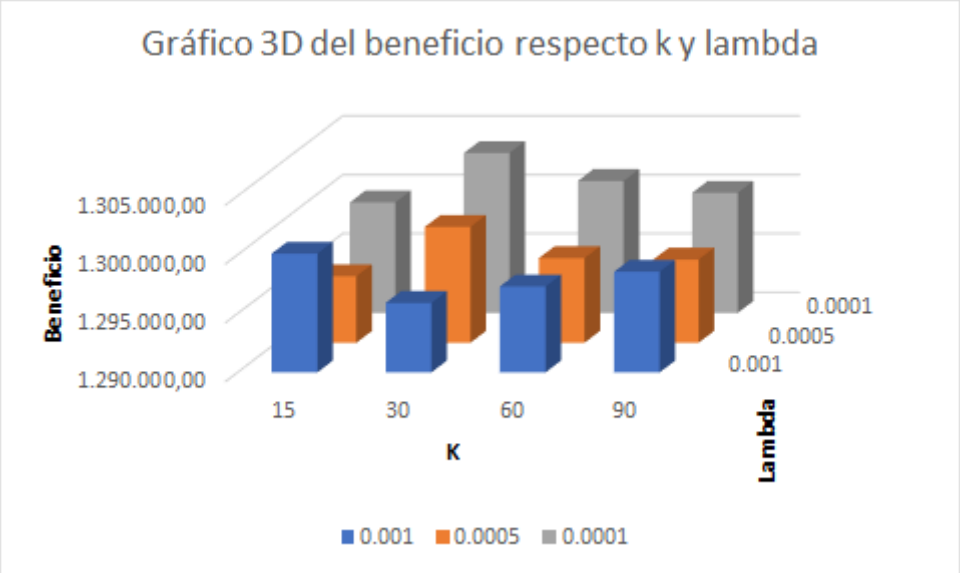

Figura 9: Gráfico 3D del beneficio obtenido respecto a los parámetros k y lambda (3)

Aquí vemos como tenemos un máximo con lambda = 0,0001 y k = 30. Por tanto, hemos encontrado unos valores para estos dos parámetros que usaremos en el resto de experimentos.

Puesto que ya tenemos los valores de k y lambda, ahora procederemos a realizar ejecuciones para encontrar el valor óptimo para stiter. Para eso llamaremos a la función experimento3_stiter() y probaremos un cierto rango de valores para stiter (todos ellos divisores de steps) entre 50 y 2.500. Como ya advertía la documentación, podría pasar que dependiendo de los parámetros k y lambda escogidos, stiter fuera poco relevante. De hecho, nos ha sido difícil detectar alguna tendencia, ya que las medias daban valores muy parecidos. Para una media de 1.315.972,426, la desviación estándar es de 2.280,92.

Mirando el gráfico nos pudimos dar cuenta de un máximo en stiter = 100 y otro en stiter = 250. Para valores de stiter mayores a 500 hasta 2.000 parecía no haber mucha diferencia, así que decidimos escoger el valor del máximo mayor en stiter = 250.

stiter	beneficio
50	1.316.204,2
100	1.318.219,7
200	1.315.904,5
250	**1.320.888,0**
500	1.314.138,5
1.000	1.315.320,3
1.500	1.314.539,5
2.000	1.313.676,3
2.500	1.314.860,8

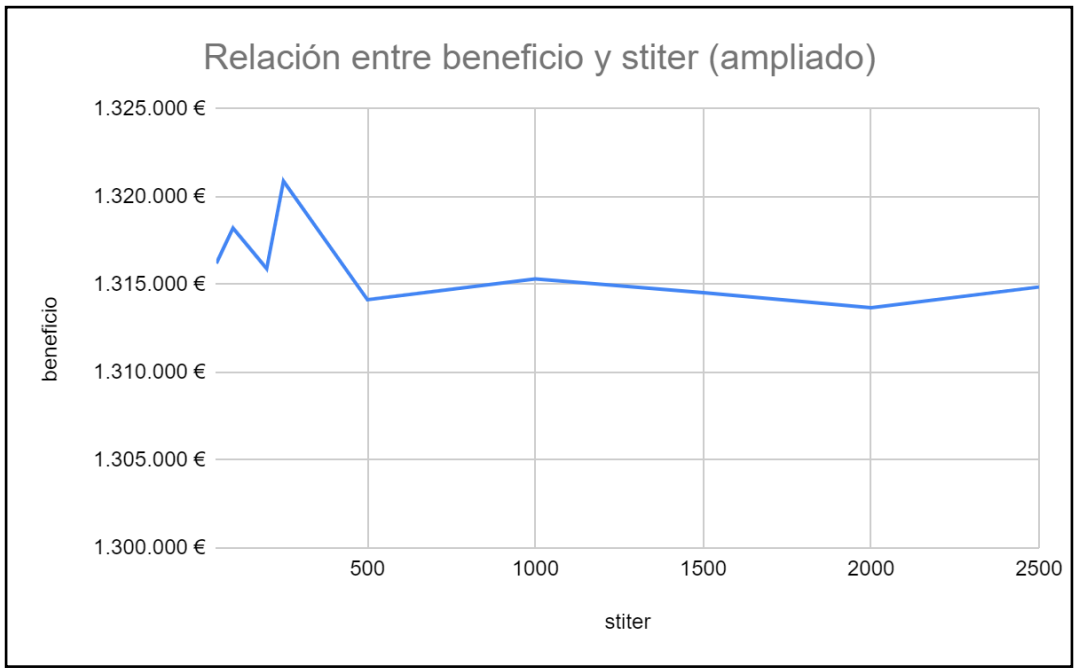

Figura 10: Gráfico ampliado entre el beneficio y el parámetro stiter

Finalmente, volvemos a steps. Esencialmente repetiremos el primer experimento, con Simulated Annealing y Hill Climbing, para asegurarnos de que converja correctamente y sea mejor que Hill Climbing con los nuevos valores. Ahora volveremos a usar la función experimento3_steps() fijando los parámetros ya encontrados y con posibles valores de steps múltiplos de stiter.

steps	beneficioSA	beneficioHC
10.000	1.206.961,6	1.286.215,5
16.000	1.235.835,5	1.286.215,5
32.000	1.275.567,2	1.286.215,5
64.000	1.291.459,60	1.286.215,50
128.000	1.307.352,0	1.286.215,5
256.000	1.311.990,1	1.286.215,5

512.000	1.319.244,2	1.286.215,5

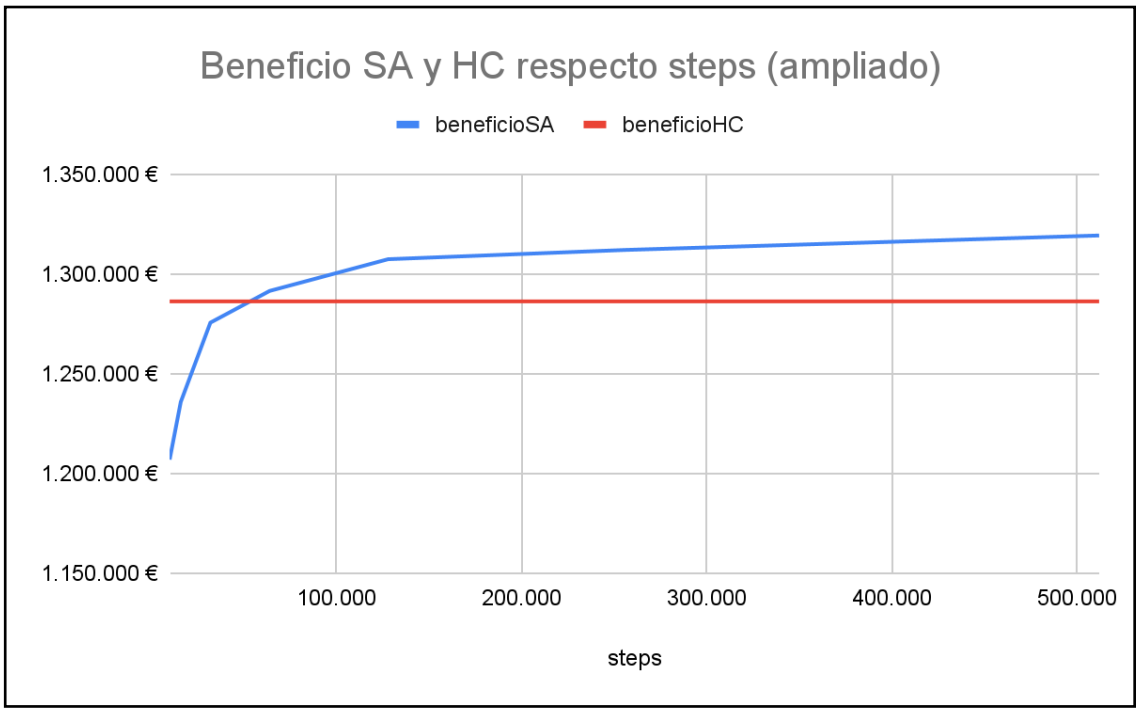

Figura 11: Gráfico ampliado de la comparación final del valor final del heurístico de SA y HC respecto al número de iteraciones totales

Observando la tabla de medias y el gráfico de la Figura 11, notamos que al principio el algoritmo de Hill Climbing obtiene mejores resultados que el Simulated Annealing, pero a partir de un valor alrededor de 64.000 no hay ninguna duda que Simulated Annealing se comporta mejor. Además, notamos como a valores de steps muy altos el Simulated Annealing se va normalizando: vemos como de 128.000 a 256.000 el beneficio varía del orden de 4.000€, mientras que de 32.000 a 64.000 o de 64.000 a 128.000 varía del orden de 20.000€.

Aunque en este experimento no es nuestro objetivo, cabe mencionar que el factor tiempo de ejecución es siempre una limitación a tener en mente para poder ejecutar estos algoritmos, y no es razonable realizar un gran número de pasos para obtener mejoras de beneficios tan pequeñas sacrificando el tiempo de ejecución. Por tanto, hemos decidido escoger como steps el valor de 64.000 porque la diferencia de beneficios con los demás valores ya era más pequeña y, en cambio, el tiempo de ejecución se duplicaba entre dos steps consecutivos.

Experimentalmente, hemos optimizado la función de búsqueda de SA a los parámetros: steps = 64.000, stiter = 250, k = 30 y lambda = 0,0001.

4.5 Experimento 4: Aumentar centrales y clientes

Este experimento tiene dos apartados: uno en el que estudiaremos la evolución del tiempo de ejecución según vayamos aumentando el número de centrales y otro en el que vayamos aumentando el número de clientes.

a) Evolución del tiempo según el número de centrales

En este primer experimento hemos fijado el número de clientes a 1.000 con las mismas configuraciones iniciales que el resto de experimentos. Hemos fijado el número inicial de centrales a 40 y las hemos ido aumentando en 40 unidades hasta que el tiempo de ejecución ya no fuese razonable para seguir con la experimentación.
Nos hemos dado cuenta de que a partir de un número de centrales mayor a 120 el tiempo por ejecución superaba los 8 minutos. Por tanto, debido al gran número de repeticiones que debíamos hacer, decidimos realizar este experimento desde 40 hasta 120 centrales.

Para obtener los datos hemos hecho la experimentación con 6 semillas distintas, donde en cada una de ellas realizamos una ejecución para cada número de centrales, para un total de 3 ejecuciones por semilla.

Centrales	tiempo (ms)
40	44.445,8
80	188.096,4
120	296.190,0

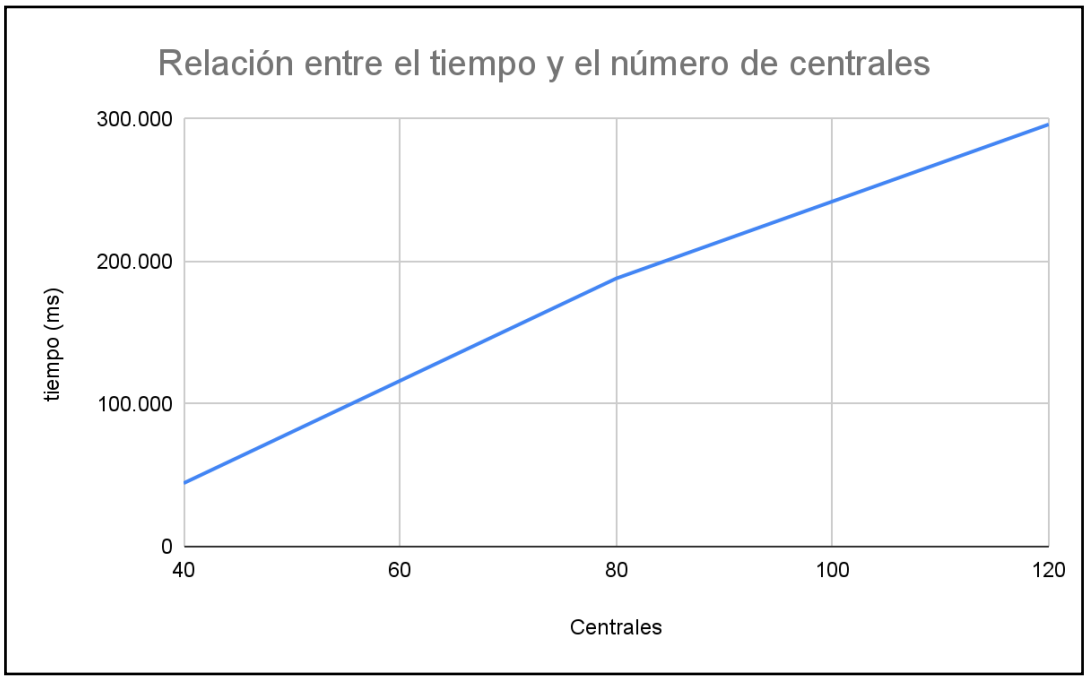

Figura 12: Gráfico entre el tiempo de ejecución y el número de clientes

Como podemos ver en la tabla de medias y en el gráfico anterior, el tiempo de ejecución es linealmente proporcional al número de centrales. Empezamos con un tiempo aproximado de 44 segundos y vamos aumentando unos 2 minutos por cada 40 centrales que añadimos, con un valor aproximado de 5 minutos para 120 centrales.

Este resultado era de esperar, ya que cuantas más centrales tengamos en nuestro mapa, más posibilidades tendrá un cliente para escoger una central que le suministre energía y, por tanto, tendremos que explorar más estados sucesores, factor que aumenta el tiempo de ejecución.

Centrales	Beneficio
40	1.335.916,6
80	1.120.377,6
120	954.567,8

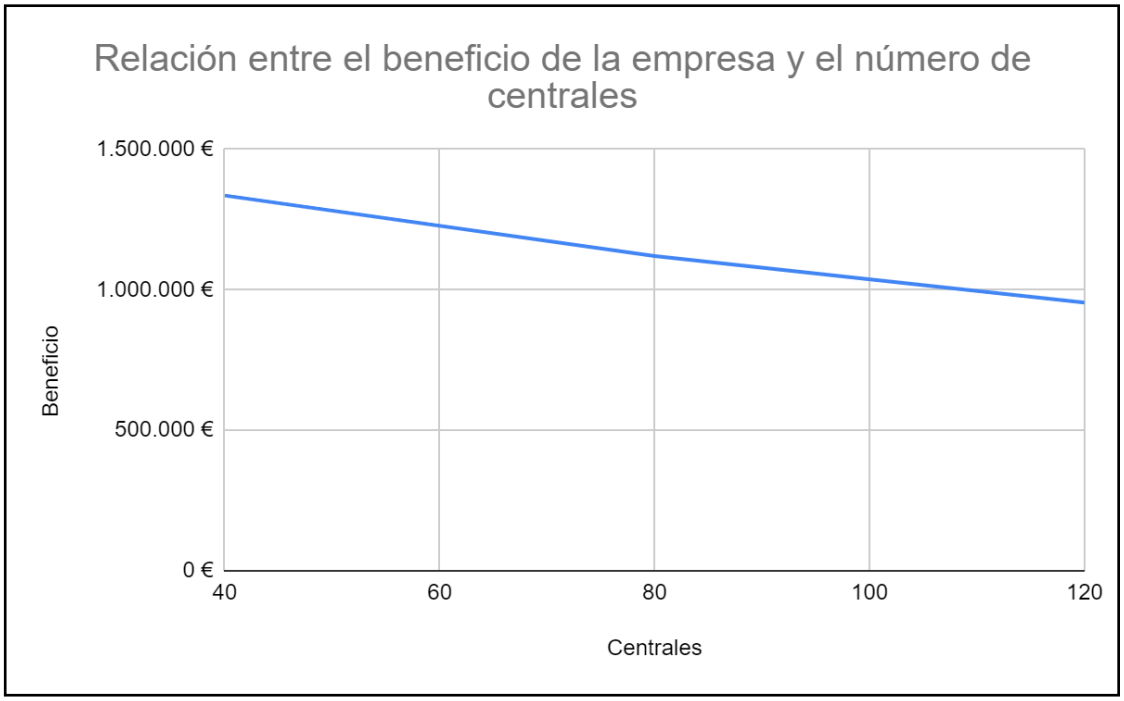

Figura 13: Gráfico entre el beneficio y el número de centrales

Cabe mencionar que al contrario que el tiempo de ejecución, el beneficio que generamos disminuye a medida que aumentamos el número de centrales, esto se puede ver por la pendiente negativa que tiene el gráfico.

Vamos a razonar esto: cada central tiene un coste de mantenimiento para dejarla activa o parada. Si aumentamos el número de centrales, entonces deberemos de gastar más dinero para mantener a las centrales nuevas. Como el número de clientes sigue siendo el mismo, la cantidad de energía que piden no variará, y, por tanto, lo único que estaremos generando son más pérdidas por tener a más centrales activas o paradas.

b) Evolución del tiempo según el número de centrales

En este segundo experimento hemos fijado el número de centrales a 40 con las mismas configuraciones iniciales que el resto de experimentos. Hemos fijado el número inicial de clientes a 1.000 y los hemos ido aumentando en 500 unidades hasta que nos era imposible generar una solución inicial donde todos los clientes garantizados estaban asignados en alguna central.

A diferencia del experimento anterior en el cual decidimos parar de incrementar las centrales porque llegábamos a tiempos muy elevados, en este nuestro problema era el generar una solución inicial válida. Al mantener fijado el número de centrales y solo estar aumentando el número de clientes, llegamos a un punto donde el consumo total que

piden los clientes garantizados es superior a la producción total de las centrales, por lo cual es imposible poder generar una solución inicial.
Esto en el experimento anterior no pasaba, ya que al aumentar el número de centrales estábamos incrementando la producción total y, por tanto, si con 40 centrales podíamos generar una solución inicial, entonces con más centrales también la podremos generar.

Para solucionar este problema decidimos disminuir el porcentaje de clientes garantizados del 75% a un 25%, para que así nos fuese más fácil generar soluciones iniciales. Dado que en los experimentos anteriores con 750 clientes garantizados y 40 centrales nunca se nos había dado este problema, hemos realizado los experimentos hasta un total de 3.000 clientes, para que así el número de clientes garantizados con 3.000 clientes sea 750, el cual ya sabíamos que no nos daría problemas.
También probamos de hacer ejecuciones con 3.500 clientes, pero vimos que ya no podíamos generar soluciones iniciales, así que encontramos que nuestro valor límite era 3.000 clientes.

Para obtener los datos hemos hecho la experimentación con 10 semillas distintas, donde en cada una de ellas realizamos una ejecución para cada número de clientes, desde 1.000 hasta 3.000, para un total de 5 ejecuciones por semilla.

Clientes	tiempo (ms)
1.000	33.873,2
1.500	45.212,5
2.000	42.755,8
2.500	43.346,6
3.000	31.638,9
Media	**39.365,4**

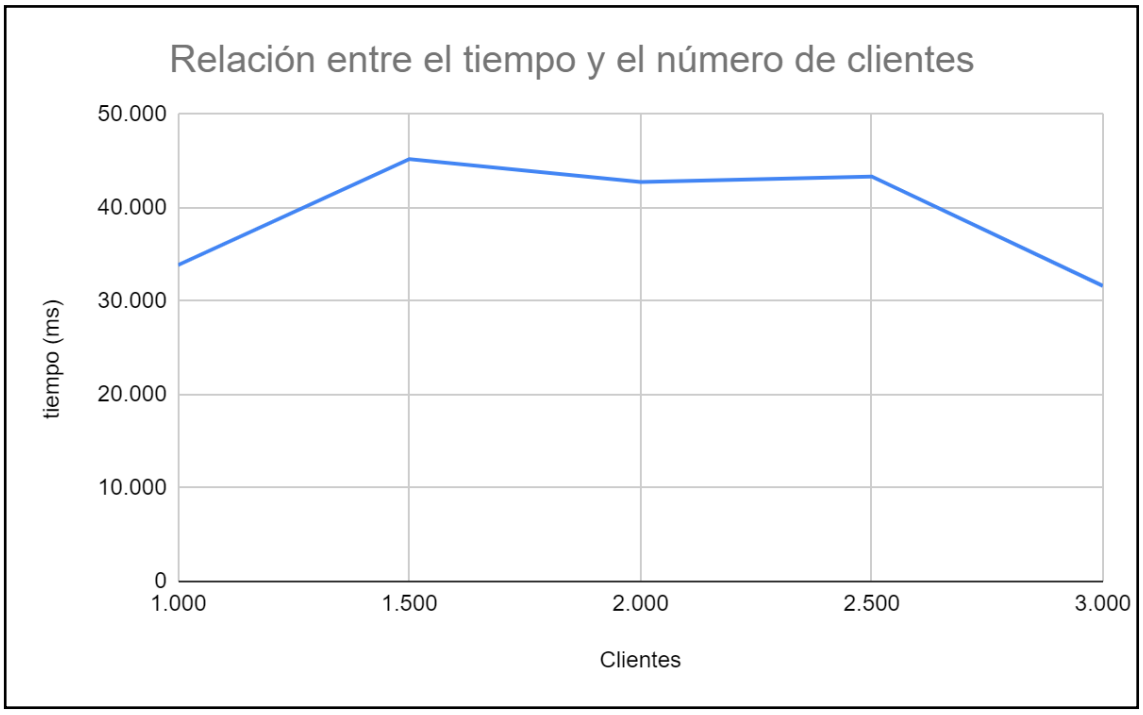

Figura 14: Gráfico entre el tiempo y el número de clientes

En este experimento nos encontramos con unos resultados que nos sorprendieron: el número de clientes parecía no afectar al tiempo de ejecución. Esto era extraño, ya que cuando hicimos pruebas en experimentos anteriores siempre obteníamos tiempos mayores cuando decidíamos aumentar el número de clientes. Debido a esto decidimos realizar el experimento de nuevo y ahora guardarnos un dato que nos ayudaría a resolver este misterio: el número de clientes que faltaban por asignarse.

Clientes	Clientes Sin Asignar
1.000	15,1
1.500	216,2
2.000	605,3
2.500	1141,7
3.000	1787,3

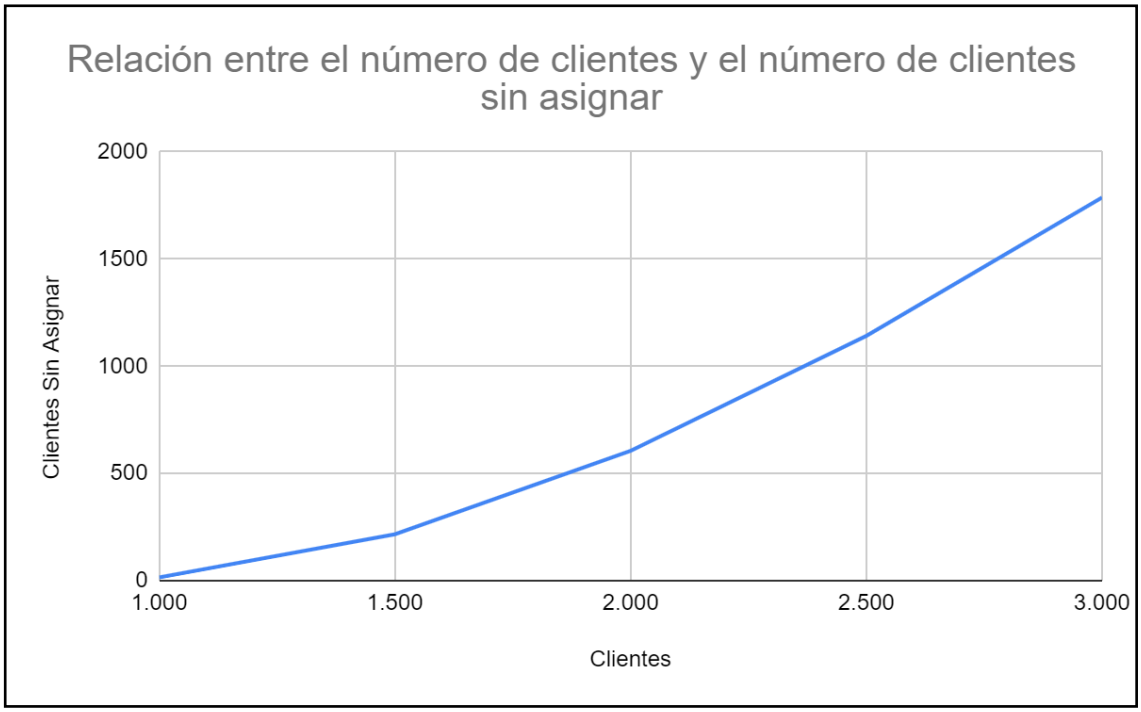

Figura 15: Gráfico entre el número de clientes y el número de clientes sin asignar

En el gráfico de la Figura 15 podemos observar como a medida que el número de clientes aumenta, también lo hace el número de clientes sin asignar, y parece que lo hace de forma lineal.

Vamos a razonar esto: debido a que el número de centrales no varía y, por tanto, tampoco varía la producción total, estamos aumentando cada vez más el consumo total de los clientes cuando la producción total se mantiene igual. Hemos de recordar que los clientes garantizados tienen prioridad a la hora de asignarse una central, esto provoca que llegue un momento donde los clientes garantizados ocupen una gran parte de la producción total y no dejen a los clientes no garantizados asignarse a una central.

Esto lo podemos ver en la Figura 16, donde el número de clientes asignados se mantiene constante en un rango de valores entre 1.000 y 1.500. Esto significa que para una configuración de 40 centrales con la proporción por defecto de 5 centrales de clase A, 10 de clase B y 25 de clase C, el número de clientes que podamos asignar esté siempre en ese rango.

Figura 16: Gráfico entre el número de clientes y el número de clientes asignados

Lo que provoca esto es que la función getSuccessors() del Hill Climbing genere cada vez menos sucesores válidos a medida que aumentemos el número de clientes, ya que al estar la gran mayoría de centrales llenas, los operadores de swap y move tendrán menos posibles combinaciones correctas y, por tanto, no podrán explorar tantos estados sucesores como lo hacían en otros casos.

Como ahora hemos limitado bastante el espacio por el que se pueden mover los operadores, es normal que el tiempo de ejecución no aumente junto con los clientes, ya que cada vez estamos reduciendo más el espacio libre de las centrales, lo que provoca una reducción del tiempo de ejecución al no explorar tantos estados.

4.6 Experimento 5: Generando soluciones iniciales a través del heurístico

En este experimento nos piden empezar desde un estado inicial que no sea solución, de hecho, un estado vacío donde todos los clientes están sin central, incluso los garantizados, y guiar a través del heurístico hacia una solución, es decir, que tenga todos los garantizados asignados. De hecho, este es un experimento bastante interesante, porque si lo conseguimos no haría realmente falta que implementásemos algoritmos de soluciones iniciales, ya que sería el propio heurístico quien nos ayudaría a generarla. Para hacer esto tendremos que modificar el heurístico y añadir una penalización por cada cliente garantizado que no tenga central. Tenemos que tener en cuenta que esta penalización tiene que ser lo suficientemente grande como para que sea notoria y el algoritmo modifique el camino en el espacio de búsqueda. El heurístico quedaría de la siguiente forma:

$$h(x) = A - BC - K * nº\ clientes\ garantizados\ sin\ central$$

Tenemos que darnos cuenta de que ahora vamos a tener que navegar por el espacio de estados y no de soluciones, por lo que las condiciones de aplicabilidad de nuestros operadores van a cambiar de modo que puedan generar un estado que no sea solución.

Por lo que para hacerlo vamos a permitir a nuestras funciones que comprueban la aplicabilidad que permitan que un garantizado pueda no tener central. En cuanto a los atributos, vamos a tener que añadir el número de clientes garantizados sin central, que vamos a ir actualizando de forma constante en cada estado, y también un factor de penalización K que a continuación vamos a analizar el valor que le pusimos.

A priori se nos ocurrió que la penalización tenía que ser enorme. Para hacerlo calculamos cuál era el peor heurístico que se podía obtener en un estado y lo pusimos como la penalización por cada cliente garantizado no asignado. Para hacer el cálculo de este heurístico simplemente pensamos el peor estado de todos, que sería no tener a ningún cliente asignado (tenemos que pagar a todos indemnización) y todas las centrales estaban activas (pagamos el coste de generar toda la producción y luego no la usamos). De hecho, este heurístico nos funcionó bien y nos permitía generar soluciones, y en realidad es lógico, ya que el asignar un cliente garantizado a una central nos mejora en una cantidad ingente de unidades el heurístico. Pero en ese momento nos planteamos si había una penalización más acotada que nos funcionase. Para intentar reducir el factor de penalización pensamos en cuál era el movimiento que más mejoraba el primer factor del heurístico (el balance de la empresa) y era el apagar la central más grande que hubiese en ese problema. De hecho, si lo queremos hacer general para todos los problemas podemos usar la central más grande posible con los datos que nos daban en el enunciado, es decir, apagar una central de 750 MW, esto nos mejoraría el heurístico de la siguiente forma:

$$Mejora = 750 * 50 + 20000 - 15000 = 42500$$

Y como en el peor de los casos podemos asignar a un garantizado a una central tenemos que K > 42500, en el caso de que sea de 750 MW la mayor central, si lo queremos hacer para el problema específico tendríamos que hacer el cálculo anterior con la central más grande.

Cuando hicimos los experimentos con esta modificación en el heurístico, el Hill Climbing funcionaba perfectamente y conseguimos llegar a una solución. Pero ese no era el caso del Simulated Annealing porque nos dejaba a unos pocos clientes garantizados sin asignar. Al pensar qué podía estar pasando nos dimos cuenta de que el Simulated Annealing al inicio escoge sucesores de una forma aleatoria, por lo que no está teniendo en cuenta esta modificación en el heurístico y para cuando empieza a tenerla en cuenta ya es demasiado tarde para encontrar una solución, y encuentra un máximo local en el espacio de búsqueda, pero que este no cumple las restricciones de ser solución. Por lo que usar este método no es demasiado eficaz en Simulated Annealing, en cambio, sí lo es para el Hill Climbing.

4.7 Experimento 6: Proporción Tipo Centrales

En este experimento nos piden duplicar y triplicar el número de centrales de tipo C y ver si al hacer esto utilizamos menos las centrales de tipo A y B.

Para este experimento hemos seleccionado aleatoriamente 10 semillas distintas y hemos hecho para cada una de ellas 6 ejecuciones: 3 para Simulated Annealing y 3 para Hill Climbing. Para cada algoritmo hemos ejecutado el experimento con la configuración inicial, con las centrales de tipo C duplicadas y con las centrales de tipo C triplicadas.

Para cada ejecución nos hemos guardado cuál era el número de clientes asignados a cada tipo de central, el beneficio obtenido y su tiempo de ejecución.
Resultados experimentales: (NOTA: todos los datos se encuentran adjuntos al proyecto en los ficheros de datos correspondientes)

Centrales C	Cent A SA (%)	Cent B SA (%)	Cent C SA (%)
25	38,39	31,96	28,73
50	34,5	24,88	40,62
75	28,43	18,95	

ACTUALITZA

Centrales C	Cent A HC (%)	Cent B HC (%)	Cent C HC (%)
25	41,54	31,47	25,93
50	36,73	24,39	38,88
75	30,08	19,19	50,73

En estas tablas podemos ver el porcentaje de clientes que están asignados a cada tipo de central. Vemos cómo a medida que aumentamos el número de centrales de tipo C, cada vez el porcentaje de clientes asignados a centrales de tipo C es mayor.

Nuestro objetivo era ver si al aumentar las centrales de tipo C, los otros dos tipos de central tendrían menos clientes asignados. Con las tablas de medias hemos visto que esto es cierto, tanto para Simulated Annealing como para Hill Climbing. Ahora vamos a hacer un estudio sobre los datos para verificar si realmente es cierto.

Vamos a comentar los datos para los dos casos del algoritmo de Hill Climbing con la proporción de centrales de tipo A y de tipo B, y para el algoritmo de Simulated Annealing se podrán sacar las mismas conclusiones viendo los datos que pondremos a continuación:

```
Paired t-test
data:  pA1 and pA2
t = 4.4003, df = 9, p-value = 0.0008598
alternative hypothesis: true mean difference is greater than 0
95 percent confidence interval:
 0.03053807       Inf
sample estimates: mean difference 0.0523438
```

```
Paired t-test
data:  pA1 and pA3
t = 5.304, df = 9, p-value = 0.0002456
alternative hypothesis: true mean difference is greater than 0
95 percent confidence interval:
 0.07777047       Inf
sample estimates: mean difference 0.1188438
```

Vemos empíricamente que hay diferencia entre los datos. pA1 (proporción de clientes asignados a centrales de tipo A) es un 5.2% mayor a pA2 (proporción de clientes asignados a centrales de tipo A con tipo C duplicadas), y notablemente mayor con un 11.8% a pA3 (proporción de clientes asignados a centrales de tipo A con tipo C triplicadas). Por tanto, podemos argumentar que al aumentar la proporción de centrales de tipo C, la proporción de clientes asignados a centrales de tipo A disminuye.

Podemos dar la misma explicación para las centrales de tipo B viendo los siguientes datos:

```
Paired t-test
data:  pB1 and pB2
t = 5.6437, df = 9, p-value = 0.0001581
alternative hypothesis: true mean difference is greater than 0
95 percent confidence interval:
 0.05000923        Inf
sample estimates: mean difference 0.07406682
```

```
Paired t-test
data:  pB1 and pB3
t = 9.5532, df = 9, p-value = 2.614e-06
alternative hypothesis: true mean difference is greater than 0
95 percent confidence interval:
 0.1018766       Inf
sample estimates: mean difference 0.1260668
```

En este caso, la proporción de clientes asignados a centrales de tipo B al duplicar el número de centrales de tipo C (pB2) es un 5% menor a la proporción de clientes asignados a centrales de tipo B original (pB1), y un 12.6% menor cuando triplicamos el número de centrales de tipo C.

Estos son los datos recogidos para Simulated Annealing, con los mismos nombres para las variables y la misma conclusión que en Hill Climbing:

```
Paired t-test
data:  pA1 and pA2
t = 3.9014, df = 9, p-value = 0.001806
alternative hypothesis: true mean difference is greater than 0
95 percent confidence interval:
 0.02243879        Inf
sample estimates: mean difference 0.04232581
```

```
Paired t-test
data:  pA1 and pA3
t = 4.4594, df = 9, p-value = 0.0007893
alternative hypothesis: true mean difference is greater than 0
95 percent confidence interval:
 0.06067574        Inf
sample estimates: mean difference 0.1030258
```

```
Paired t-test
data:  pB1 and pB2
t = 8.3064, df = 9, p-value = 8.187e-06
alternative hypothesis: true mean difference is greater than 0
95 percent confidence interval:
 0.05740523        Inf
sample estimates: mean difference 0.07366129
```

```
Paired t-test
data:  pB1 and pB3
t = 12.886, df = 9, p-value = 2.094e-07
alternative hypothesis: true mean difference is greater than 0
95 percent confidence interval:
 0.1140469       Inf
sample estimates: mean difference 0.1329613
```

Por tanto, podemos afirmar empíricamente que tanto para Hill Climbing como para Simulated Annealing, al duplicar y triplicar el número de centrales de tipo C, el número de clientes asignados a centrales de tipo A y tipo B disminuye.

Ahora vamos a hacer un estudio del tiempo de ejecución. Para eso vamos a observar la tabla de medias y los gráficos obtenidos a partir de la experimentación y sacaremos conclusiones.

Centrales C	tiempoSA (ms)	tiempoHC (ms)
25	199	18.537,50
50	58,33333333	120.171,00
75	39,33333333	186.816,00

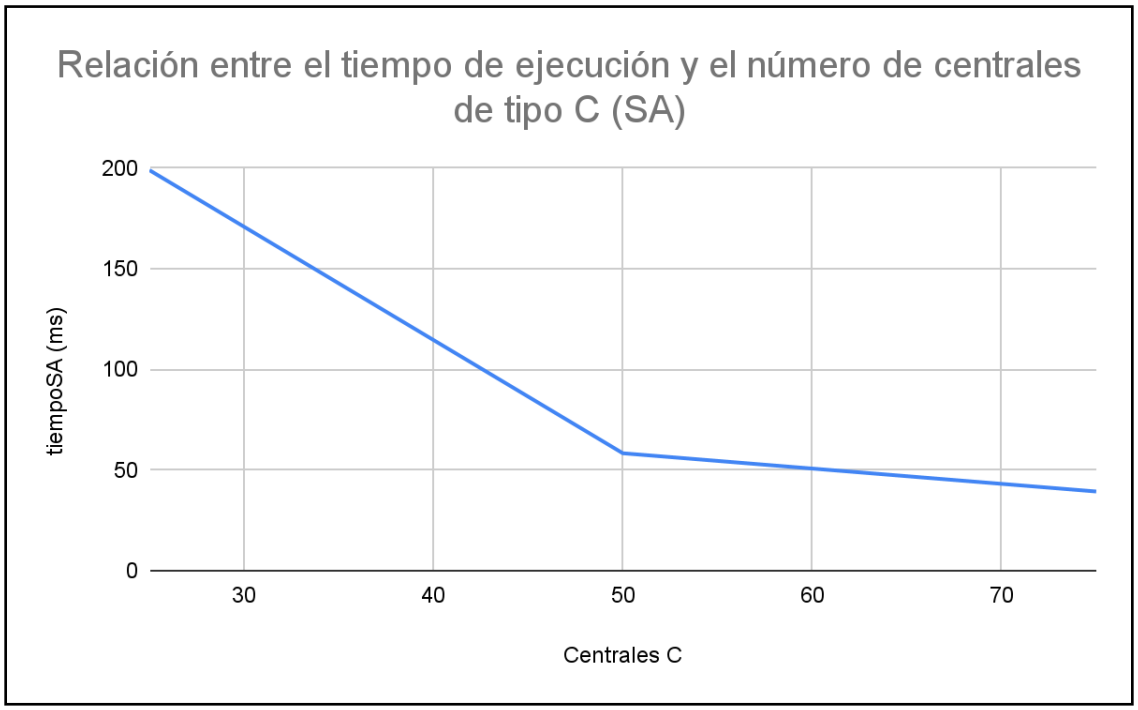

Figura 17: Gráfica del tiempo de ejecución respecto al número de centrales de tipo C (SA)

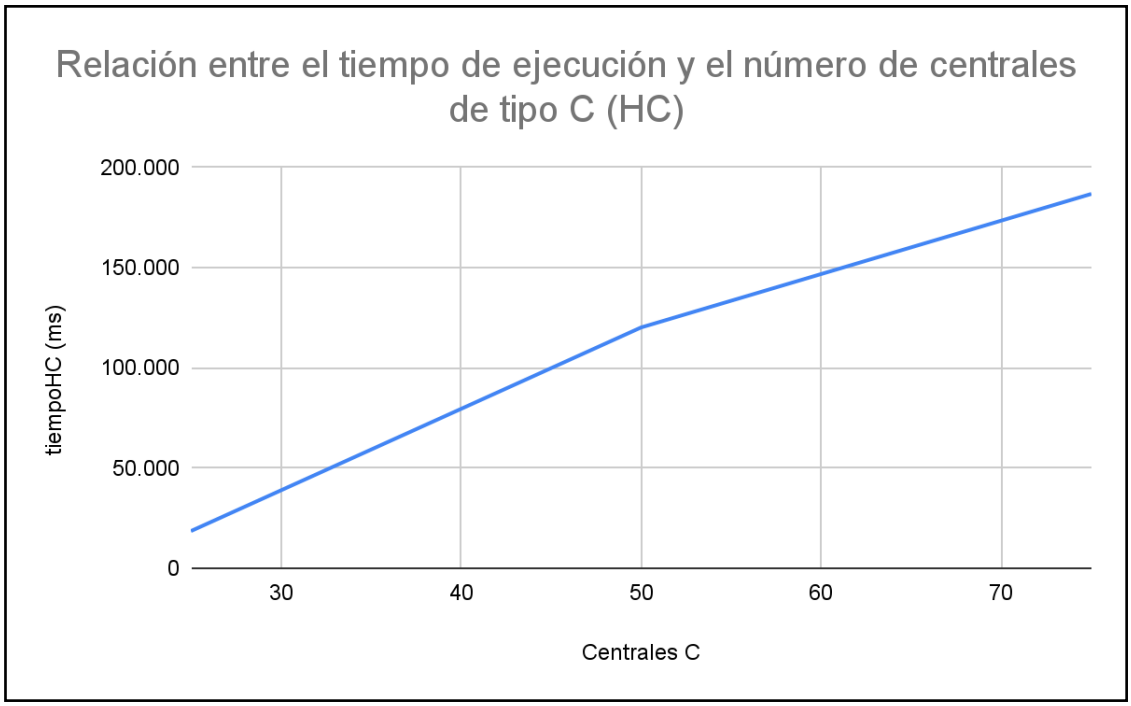

Figura 18: Gráfica del tiempo de ejecución respecto al número de centrales de tipo C (HC)

Vemos una cosa extraña, en el caso de Simulated Annealing el tiempo parece disminuir a medida que aumentamos el número de centrales de tipo C, mientras que en Hill Climbing, el tiempo aumenta de forma casi lineal.

Empezaremos argumentando esto para el caso del Simulated Annealing. Al haber menor número de centrales, es más complicado generar sucesores que cumplan las restricciones de consumo y producción: hay poco espacio libre en las centrales y, por tanto, hay menos posibilidades de generar un sucesor aleatorio que cumpla las restricciones. Debido a esto, tarda más tiempo en ejecutar el algoritmo.

En cambio, a medida que aumentamos el número de centrales vamos aumentando también la capacidad de producción total, y, por tanto, el Simulated Annealing es más libre de explorar más posibilidades, ya que no tantas estarán bloqueadas porque no cumplan las restricciones, permitiendo ser más rápido al tiempo de ejecutarlo.

Todo lo contrario pasa en el caso del Hill Climbing. Al haber menos centrales, el algoritmo explora menos posibles estados sucesores y, en consecuencia, tarda menos tiempo. A medida que vamos aumentando el número de centrales, este tiene que explorar muchos más estados sucesores y por eso su tiempo de ejecución aumenta.

4.8 Experimento 7: Experimento Especial

Para este experimento hemos realizado 10 ejecuciones mediante el algoritmo de Hill Climbing con la configuración inicial con la semilla 1234. Estos son los resultados obtenidos:

# Ejecución	Clientes asignados	tiempo (ms)
1	973	23.276
2	986	28.936
3	989	26.726
4	992	35.469
5	983	21.375
6	983	25.725
7	987	30.671
8	983	32.734
9	987	30.792
10	986	32.801
Media	**985**	**28.851**

5. Conclusiones

Finalmente, después de realizar los diferentes experimentos de los que se compone la práctica, podemos concluir en que los algoritmos de búsqueda local que hemos utilizado nos permiten solucionar el problema que inicialmente se nos ha planteado de una forma bastante óptima.

Debido a que el espacio de estados y soluciones es enorme para este problema, intentar resolverlo con algoritmos más tradicionales que no usaran ningún tipo de heurística se nos hubiese hecho imposible por la necesidad de tiempo, pero sobre todo de memoria para llegar a soluciones medianamente óptimas.

Además, nos gustaría destacar que a pesar de ser una idea muy sencilla, en ningún momento se nos ocurrió que podíamos usar el heurístico para generar una solución, y el experimento número 5 nos ha ayudado a ver eso.

También podemos ver que una vez programado y solucionado este problema no tendríamos gran dificultad en añadir más componentes a nuestro problema, obviamente sin modificar la naturaleza del problema, como podría ser el añadir más tipos de centrales o nuevos tipos de clientes. Así, una vez tenemos el código base, es relativamente fácil extenderlo a versiones más complejas del problema.

Algo que también nos gustaría destacar porque nos sorprendió muchísimo es la velocidad en la que el Simulated Annealing consigue llegar a la solución óptima. Pensábamos que no habría mucha diferencia entre el Hill Climbing y el Simulated Annealing, pero estábamos bastante equivocados. A pesar de que normalmente el Hill Climbing nos ofrecía más beneficio, el beneficio extra que nos daba es prácticamente despreciable por el precio que tenemos que pagar en tiempo y memoria, y para problemas reales creemos que lo más inteligente es pagar ese precio.