Práctica de Búsqueda Local

Grau IA Q1 Curso 2022-2023

Miembros del grupo

Miró López-Feliu, Oriol

Cruz Rodríguez, Unai

Rivera Monsergas, Nico

Tabla de contenidos

Introducción

En esta práctica se han usado dos algoritmos distintos para resolver el problema propuesto

- Por un lado tenemos el algoritmo *Hill Climbing* (HC). Se empieza en una solución del problema y se va desplazando por el espacio de soluciones según varios operadores, tomando siempre el mejor camino inmediato que exista (Moverse al nodo vecino con mejor heurística).
- Por otro lado, tenemos el algoritmo *Simulated Annealing* (SA). Empieza de la misma forma que HC; pero al principio se mueve de forma aleatoria por el espacio de soluciones, ajustando el movimiento a medida que va avanzand. De modo que cada vez se realicen menos movimientos aleatorios (A medida que avanza, acepta menos nodos con heurísticas peores que el nodo actual).

Aclaraciones

Con tal de mejorar la legibilidad, durante el documento nos referiremos al número total de clientes como n, y al número total de centrales como m.

Algunas de las gráficas de este documento contienen ejes que no empiezan en cero. Se decidió hacer de esta forma para facilitar la legibilidad de estas, ya que los valores eran altos y tenían poco rango. Consecuentemente se pudieron comentar mejor los resultados obtenidos.

Descripción del problema

En esta práctica se nos plantea un problema de optimización de beneficio para una compañía que distribuye energía a diferentes clientes desde sus centrales en un tablero de 100x100. Se nos presentan una serie de propiedades para estos clientes y centrales, que detallaremos más adelante, y unas restricciones que seguir. Se nos proporcionan además diversas clases Java.

Elementos del problema

En el problema diferenciamos dos elementos principales: los clientes y las centrales. Podemos asumir que un estado del problema vendrá definido por la asignación de clientes a centrales.

Los clientes:

Hay 3 tipos de clientes, dependiendo del consumo de energía que tengan contratado. Estos son “G” si contratan entre 1 y 2 Mw, “MG” si son de 2 a 5 MW y finalmente “XG” si contratan entre 5 y 20 MW. Además, existen dos tipos diferentes de contrato: “Garantizado” y “No Garantizado”. A un cliente garantizado es obligatorio suministrarle energía, mientras que a uno no garantizado podemos escoger no hacerlo, a cambio de una penalización. Cada combinación de estos elementos supone un pago diferente por MW.

Las centrales:

Análogamente, hay 3 tipos de centrales, dependiendo de la energía que producen. Las de tipo “C” producen entre 10 y 100 MW, las de tipo “B” entre 100 y 250 MW y las de tipo “A” entre 250 y 750 MW. Las centrales pueden estar encendidas o apagadas. En caso de estar encendidas, producen toda la energía posible que pueden, y hay un coste por Mw y un coste para ponerla en marcha que varía según el tipo. En el caso de estar parada, también hemos de asumir un coste, que también varía según el tipo. La tabla que muestra estos datos es la siguiente:

Tipo	Producción	Coste marcha	Coste parada
A	250 a 750 Mw	Prod*50+20000	15000
B	100 a 250 Mw	Prod*80+10000	5000
C	10 a 100 Mw	Prod*150+5000	1500

Propiedades de una asignación

Una asignación de un cliente a una central tiene en cuenta la distancia entre estos. Si el cliente está muy lejos de la central, tendremos una pérdida de energía por la distancia, de la cual hemos de asumir nosotros el coste. La distancia se calcula de forma euclídea, según las coordenadas de cliente y central.

Análisis de los elementos:

Sobre los clientes

- No podemos fraccionar la demanda de un cliente. Es decir, si lo asignamos a una central, hemos de proporcionarle toda la energía que tiene contratada.

Además, hay varios elementos que se relacionan de forma intuitiva:

- La diferencia entre la energía contratada por un cliente y la energía que producimos para poder suministrarsela (debido a la pérdida por distancias) nos dará la energía perdida.
- Para calcular el beneficio total de una solución hemos de sumar todos los pagos de los clientes y restarles los costes de indemnizaciones y centrales.

Restricciones de la solución

Una solución se considera válida si y sólo si cumple las siguientes condiciones:

- Cada cliente está asignado a como mucho una central.
- La suma de energía suministrada por una central a todos sus clientes, incluyendo las pérdidas, no supera nunca la energía que puede producir.
- Todos los clientes garantizados reciben el suministro de energía que tienen contratado.

Análisis de las restricciones:

Una vez añadidas estas restricciones, empezamos a ver más relaciones entre los elementos del problema:

- Debemos comprobar si es posible suministrar a un cliente teniendo en cuenta las pérdidas antes de asignarlo a una central
- Podríamos asignar y designar clientes con contrato no garantizado, pero nunca podremos dejar a un cliente garantizado sin central asignada.

A partir de esto, podemos empezar a definir las restricciones de los operadores que vamos a usar.

Criterios para evaluar la solución

El enunciado sólo explicita la maximización del beneficio.

Análisis de los criterios:

Aún así, hay una serie de implicaciones que se dan con tal de poder maximizar el beneficio:

- Queremos minimizar las pérdidas de energía por distancia.
- A un cliente con contrato no asignado no nos saldrá a cuenta suministrarle energía si dicha pérdida por distancia es demasiado elevada
- Queremos que las centrales estén tan llenas como sea posible, ya que siempre producen energía al máximo.

- A raíz del criterio anterior, a veces queremos tener centrales apagadas, debido a que nos puede salir a cuenta.

Justificación de la correcta elección de búsqueda local

Una vez analizado el problema, analizamos si es correcto el uso de búsqueda local para solucionarlo. Nuestra conclusión ha sido que es correcto este método dado las características del enunciado.

La principal razón son los criterios para evaluar la solución. Al buscar optimizar propiedades inherentes del estado del problema es apto, ya que nuestras funciones heurísticas calcularán el valor heurístico a partir de los criterios nombrados anteriormente. Además, el problema no pide la solución óptima, sino una que se le acerque.

All tener n clientes y cada uno de estos poder ir a m centrales, se puede ver que existen m^n combinaciones distintas. Este es nuestro espacio de búsqueda. Considerar un espacio de búsqueda con todas las posibles asignaciones sería demasiado grande para pretender modelar el problema con algoritmos de búsqueda básicos o backtracking.

Implementación del proyecto

Descripción y justificación de la implementación del estado:

El estado queda representado con los siguientes atributos

- ***asignación***: Vector de enteros donde cada posición corresponde a un cliente. asignación[i] indica la central a la que está asignado el cliente *i*-ésimo, y si su valor es -1 indica que no estás asignado a ninguna central.
- ***capacidad***: Vector de reales donde cada posición corresponde a una central. capacidad[i] indica la energía en MW que la central *i*-ésima genera y no está suministrando. Es decir, la energía restante por asignar (libre).
- ***beneficio***: Número real que nos indica el beneficio del estado. Calculado como la diferencia entre el dinero cobrado a los clientes y los gastos de las centrales, teniendo en cuenta la penalización al no suministrar energía a un cliente.
- ***lostBen***: Número real que nos indica las pérdidas que tiene el conjunto de centrales debido a la distancia entre estas y sus clientes.

Hemos escogido esta implementación ya que nos facilita mucho representar cambios en el estado.

El vector *asignación* nos permite operar rápidamente al asignar o desasignar un cliente a una central. El vector *capacidad* tiene un efecto similar: nos ayuda a representar cuánta energía existe aún disponible en cada central, con lo que se puede extrapolar información, cómo si está vacía o llena. De no tener este vector, calcularla sería mucho más costoso. El coste de operar sobre ambas de estas estructuras de datos es $O(1)$.

A la hora de implementar las diferentes funciones heurísticas, nos dimos cuenta que, siendo n el número de clientes, nos suponía un coste de $O(n)$ calcularla cada vez. Guardarnos los valores de *beneficio* y *lostBen* nos permite traducir este coste a $O(1)$. Las funciones heurísticas que explotan esto se explicarán más adelante.

Finalmente, queda aclarar que el estado quedaría representado con tan solo el vector *asignación*, y los demás elementos son auxiliares. Como se ha justificado antes, al tener el vector n elementos, y cada uno de estos poder tener m valores, se puede ver que el vector tiene n^m combinaciones distintas; y de aquí viene el tamaño de nuestro espacio de búsqueda.

Descripción y justificación de los operadores

Los operadores usados son:

- **swap(cliente1, cliente2)**: Sean *cliente1* y *cliente2* clientes distintos, intercambia las centrales en las que está asignado cada uno. Han de partir de centrales diferentes, y respetar el límite de capacidad de las centrales. Uno de los clientes puede no estar

asignado, siempre respetando que el otro no sea garantizado. Ramificación: $O(\frac{n*(n-1)}{2})$.

- **move(cliente, central)**: Asigna el cliente a la central respetando el límite de capacidad de esta. Puede ser que el cliente originalmente no esté asignado, o que esté en otra central. Ramificación: $O(n * m)$.

Decidimos usar únicamente estos dos operadores ya que nos permiten movernos por todo el espacio de solucionen y siempre generan estados válidos. El *move* solo no nos permitiría explorar más cuando todos los clientes están asignados. El *swap* solo no nos permite explorar clientes que no han sido asignados a paquetes en el estado inicial.

Aún así, durante el proceso de modelaje salieron otros operadores, aunque decidimos descartarlos. Son los siguientes:

- **empty(central)**: Reparte todos los clientes de la central entre las demás. Decidimos descartarla debido a que su ramificación es demasiado elevada, y pese a ser útil, se puede obtener el mismo resultado usando el operador de move, por ejemplo. Ramificación: $O(m * (k * (m - 1)) = O(km^2 - km)$, siendo *k* el número de clientes de la central.

- **add(cliente, central)**: Asignar un cliente no asignado a una central. Descartado debido a que modificamos el *move* original para englobarlo. Ramificación: $O(n * m)$.

- **remove(cliente)**: Quita un cliente de una central, si este no es garantizado. Descartado debido a que experimentalmente, como se verá más adelante, no aportaba mucho y aumentaba significativamente el coste en algunos casos. Ramificación: $O(n)$

Descripción y justificación de las estrategias para hallar la solución inicial

Las estrategias implementadas son las siguientes:

1. **genSolIniOrdenada()**: Asigna únicamente centrales a los clientes garantizados, intentando que estos estén a la mínima distancia posible de la central que les suministra la energía, para así minimizar pérdidas. Coste: $O(n * m)$

2. **genSolIniOrdenadaTodos()**: Análoga, a la estrategia 1, pero después de asignar los garantizados, intenta hacer lo mismo con los clientes no garantizados, al menos hasta que los que queden sin asignar no quepan en ninguna central. Intenta minimizar las pérdidas y aproximarse a una solución buena. Coste: $O(n * m)$

3. **genSolIniAsignadosTodos()**: Coloca todos los clientes garantizados en la primera central que puede y procede a hacer lo mismo con los no garantizados. Es decir, asigna todos los clientes que pueda pero sin buscar una buena solución para intentar evitar picos locales). Coste: $O(n * m)$

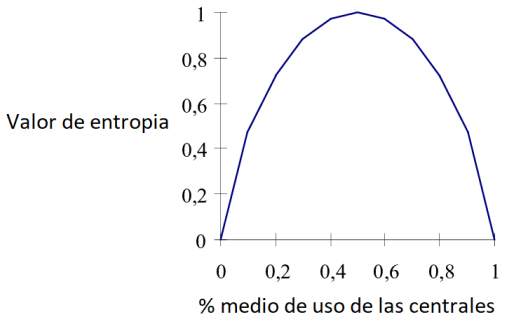
4. **genSolIniAleatoriaTodos()**: Análoga a la estrategia 4, aunque de forma aleatoria. Coste: $O(n * m)$

Descripción y justificación de las funciones heurísticas

En cuanto a heurísticas, hemos probado varias hasta quedarnos con la definitiva. Aquí mostramos las diferentes heurísticas que hemos diseñado junto con un resumen de su resultado y nuestra valoración.

1. **Beneficio:** La misma heurística que función de calidad. Bastante simple y además se puede añadir como parámetro de la representación del estado e ir modificando el beneficio cada vez que se ejecuta un operador. Esto nos evita tener que calcularlo desde cero cada vez. La acabamos rechazando porque nos dimos cuenta que no nos encaminaba correctamente, ya que era demasiado ciega. Por ejemplo, el operador swap entre dos clientes que ya tienen central asignada no cambia nunca el beneficio, ya que los clientes seguirán pagando lo mismo. De esta forma no era posible vaciar/llenar las centrales que es lo que pensamos que sería óptimo. Consecuentemente, pensamos en la siguiente heurística.

2. **Beneficio - entropía**: Esta heurística busca empujar las centrales a estar vacías o llenas. Se calcula como la entropía de Shannon, sobre el porcentaje de energía usada por las centrales. Esta función se ilustra abajo. Como se puede observar, es máxima cuando todas las centrales están siendo usadas al 50% de su capacidad, y mínima cuando están al 0% (apagadas), o al 100% (llenas). Decidimos descartarla debido a que, pese a probar varios valores k de ponderación, la heurística siguiente nos daba en media mejores resultados, con un menor tiempo de ejecución.



3. **Beneficio - lostBen**: Heurística que busca maximizar el beneficio, aunque de forma más inteligente, intentando acercar a los clientes a las centrales. Ambos parámetros han sido explicados en el apartado sobre la implementación del estado, y no requieren ninguna ponderación entre ellos debido a que las unidades de ambos ya son €. Hemos escogido definitivamente esta heurística, debido a que inteligentemente buscaba maximizar el beneficio, e ir acercando los clientes a las centrales liberaba espacio para poder insertar otros clientes. Se ha implementado de manera que su cálculo sea constante, ya que solo requiere calcularse la primera vez y sus valores se modifican correctamente con cada operador aplicado.

Experimentos

Experimento 1

Descripción: Determinar qué conjunto de operadores da mejores resultados para una función heurística que optimice el criterio de calidad del problema (3.3) con un escenario en el que 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. Deberéis usar el algoritmo de Hill Climbing. Escoged una de las estrategias de inicialización de entre las que proponéis. A partir de estos resultados deberéis fijar los operadores para el resto de experimentos.

Observación: Puede haber un conjunto de operadores mejores que otros.

Planteamiento: Probamos todas las combinaciones de operadores y comparamos sus resultados.

Hipótesis: Todos los conjuntos de operadores son iguales (H_o) o los hay que producen mejores resultados.

Método:

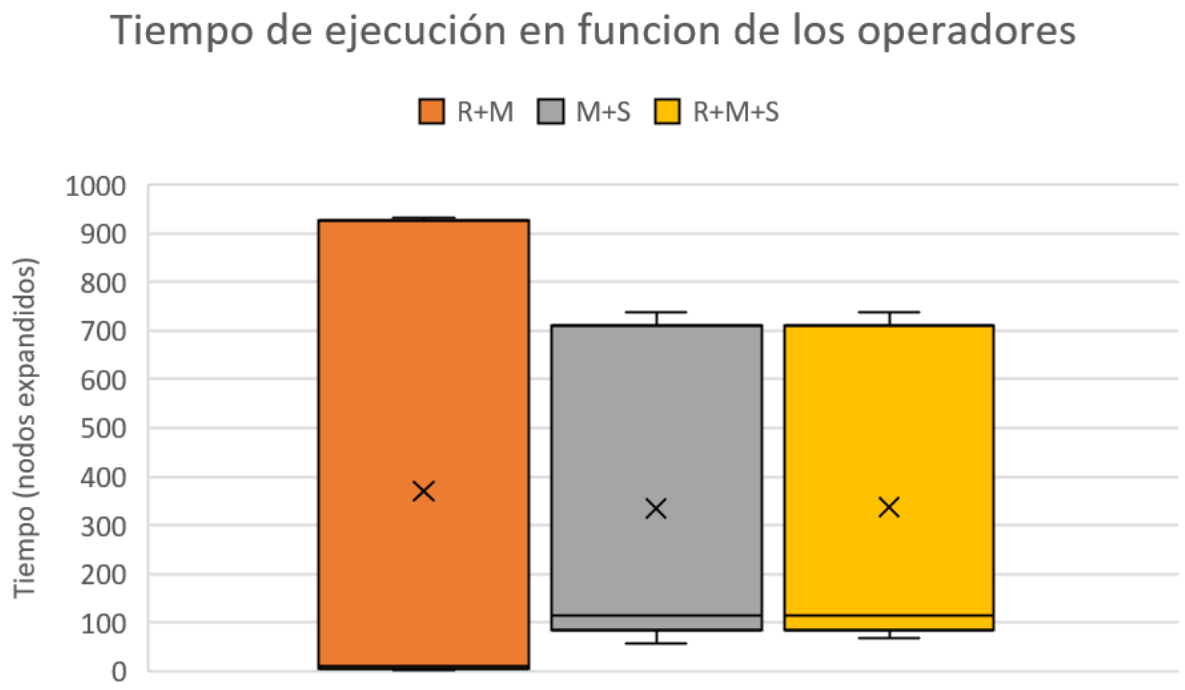
- Se parte de la solución inicial 3 y usamos 10 semillas distintas para generar el tablero.
- Ejecutamos un experimento por cada semilla y conjunto posible de operadores.
- Usamos los parámetros dados en la descripción del experimento.
- Usamos el algoritmo *Hill Climbing*.
- Usamos la heurística 3.
- Al final de cada experimento medimos:
    - Beneficio.
    - Nodos expandidos (Medida de tiempo, no tomamos milisegundos ya que esto depende de la máquina en la que se ejecute)

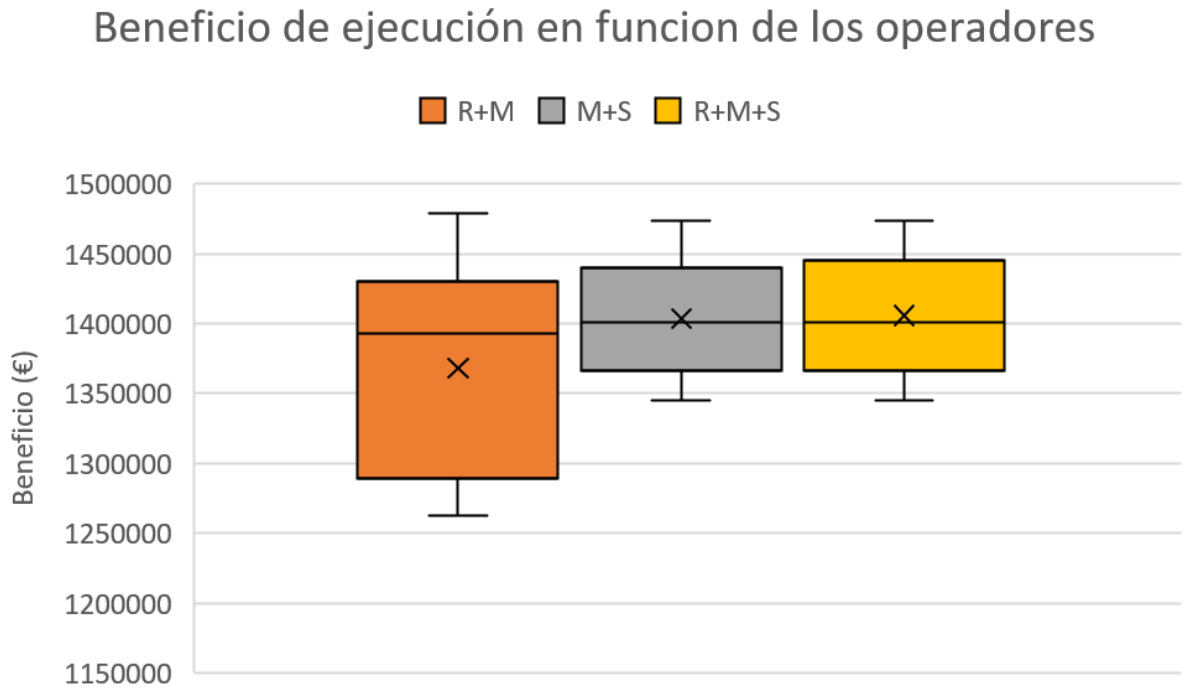
Resultados:

Se experimenta con las combinaciones R+M, M+S y R+M+S, ya que son las únicas combinaciones que permiten explorar todo el espacio de soluciones.

Tiempo de ejecucion en nodos expandidos			
Seed	**R+M**	**M+S**	**R+M+S**
1234	4	58	68
1	871	634	634
2	931	739	739
3	12	70	70
4	927	722	722
5	12	106	107
6	10	91	91
7	6	111	111
8	3	120	120
9	928	705	705
Media	370,4	335,6	336,7
Desv Est.	468,3780999	315,2801928	314,2348909

Beneficio en euros			
Seed	**R+M**	**M+S**	**R+M+S**
1234	1399738	1441767	1462029
1	1478873	1473729	1473729
2	1419520	1405671	1405671
3	1385198	1427194	1427194
4	1461971	1439182	1439182
5	1291528	1370578	1370578
6	1281349	1352897	1352897
7	1295032	1384570	1384570
8	1262488	1344642	1344642
9	1406802	1396602	1396602
Media	1368249,9	1403683,2	1405709,4
Desv Est.	79161,71751	41833,23675	44300,71351





En cuanto a resultados, todos han generado beneficios y nodos parecidos, con lo que no podemos basarnos simplemente en resultados para escoger unos operadores definitivos.

En la gráfica de cantidad de nodos generados podemos ver como la combinación de R+M da soluciones muy variadas cuando cambiamos la seed, y además tiene valores máximos de muchos nodos. Por este motivo lo descartamos para los siguientes experimentos.

Para la decisión final hemos tenido en cuenta el tiempo en ejecución en milisegundos. Aunque no lo hemos mostrado gráficamente, lógicamente ha sido mayor con la combinación

R+M+S debido al mayor factor de ramificación. Además, en nuestras pruebas hemos notado que el operador R prácticamente no se ejecuta ya que no modifica el beneficio.

Las expectativas no han encajado con lo obtenido, al esperar que hubiera una combinación mejor y los resultados mostrar que todos se comportan parecido, con lo que rechazamos H_o

Por este motivo hemos decidido usar la combinación de operadores *move* y *swap*.

Experimento 2

Descripción: Determinar qué estrategia de generación de la solución inicial da mejores resultados para la función heurística usada en el apartado anterior, con el escenario del apartado anterior y usando el algoritmo de Hill Climbing. A partir de estos resultados deberéis fijar también la estrategia de generación de la solución inicial para el resto de experimentos.

Observación: Uno de los generadores de soluciones iniciales puede ser mejor que otros.

Planteamiento: Generamos soluciones con todos y comparamos sus resultados.

Hipótesis: Todos las generaciones iniciales son iguales (H_o) o una produce mejores resultados.

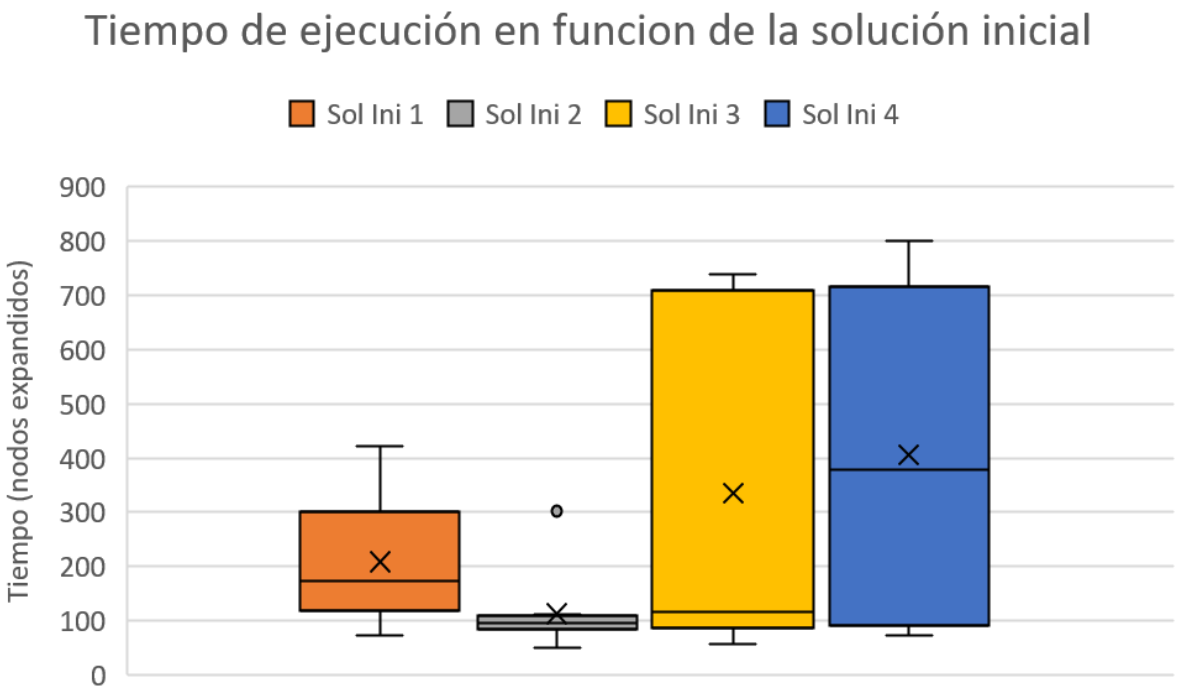
Método:

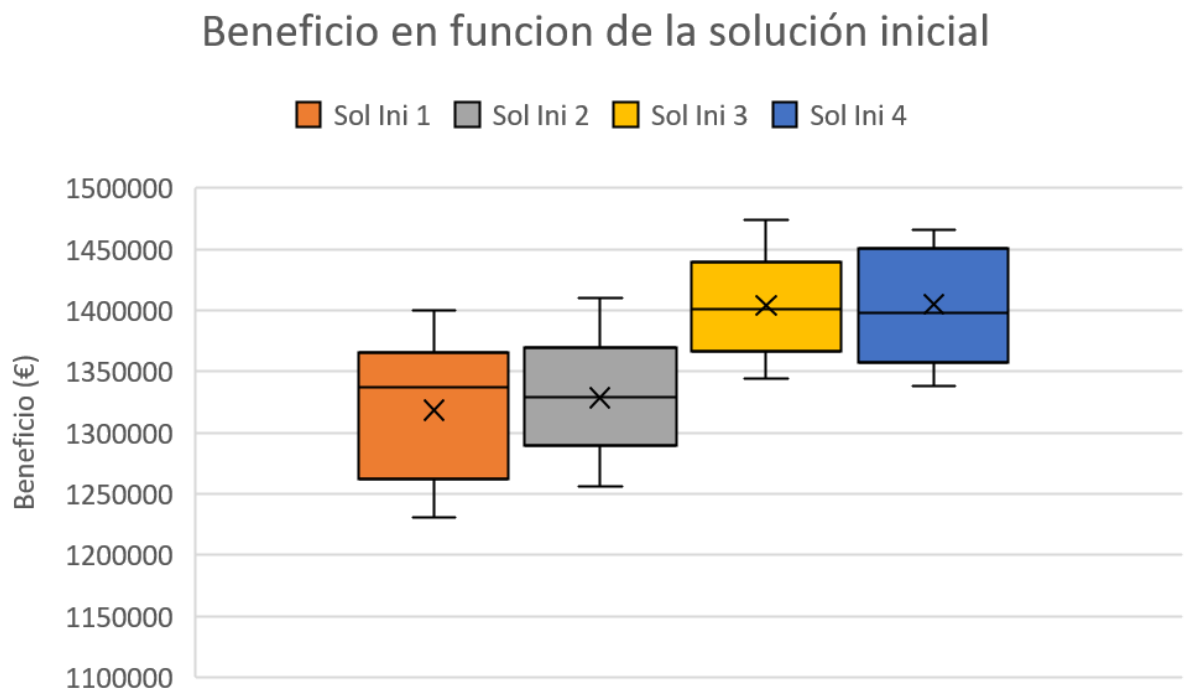
- Cuando probamos la generación inicial 4, al ser una solución con factor de aleatoriedad, se realizan 10 ejecuciones y se coge la media para cada parámetro.
- Experimentamos con los parámetros dados en la descripción del experimento 1.
- Usamos el algoritmo *Hill Climbing.*
- Usamos la heurística 3.
- Al final de cada experimento medimos:
    - Beneficio.
    - Nodos expandidos (Medida de tiempo, no tomamos milisegundos ya que esto depende de la máquina en la que se ejecute)

Resultados:

Beneficio en euros				
Seed	**Sol Ini 1**	**Sol Ini 2**	**Sol Ini 3**	**Sol Ini 4**
1234	1345053	1379269	1441767	1449206
1	1349270	1349270	1473729	1465691
2	1377346	1333714	1405671	1405062
3	1232742	1366189	1427194	1447114
4	1400243	1409672	1439182	1454517
5	1283263	1274385	1370578	1356650
6	1361023	1296663	1352897	1357745
7	1231123	1293915	1384570	1383249
8	1272134	1256466	1344642	1337675
9	1328991	1323859	1396602	1390138
Media	1318118,8	1328340,2	1403683,2	1404704,7
Desv Est.	59716,1	48755,3	41833,2	46737,6

Nodos expandidos				
Seed	**Sol Ini 1**	**Sol Ini 2**	**Sol Ini 3**	**Sol Ini 4**
1234	164	82	58	74
1	422	302	634	637
2	377	88	739	714
3	74	110	70	799
4	171	50	722	697
5	175	99	106	96
6	274	106	91	77
7	105	112	111	114
8	122	85	120	120
9	201	94	705	725
Media	208,5	112,8	335,6	405,3
Desv Est.	115,0	68,9	315,3	328,4





Lo primero que se puede destacar es la diferencia entre las dos primeras soluciones y las dos últimas. En las dos primeras, los clientes se asignan intentando minimizar las distancias, por lo que generan soluciones iniciales con un *lostBen* bajo. Por esta misma razón tiene valores bajos de beneficio y nodos expandidos: es difícil para los operadores generar un estado que no empeore los beneficios perdidos y añada a más clientes.

También añadimos la cuarta solución inicial para ver cómo se comportaría el programa al añadir un factor aleatorio. El resultado es que el rango de soluciones no varía demasiado, a veces llegando a picos más altos que las demás. Sin embargo, la media sigue siendo muy parecida a la de la tercera solución.

En base a los resultados obtenidos, tal y como esperábamos existen soluciones iniciales que son mejores que otras, con lo que aceptamos H_o.

Además, se puede entender claramente la razón por la que acabamos optando por la tercera solución inicial: ya que es una de las que más maximiza el beneficio. Además la cuarta solución inicial (el segundo candidato) tiene un factor de aleatoriedad que conlleva a una varianza en los resultados con la que es más difícil experimentar.

Experimento 3

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

Observación: El algoritmo *Simulated Annealing*, al principio se moverá aleatoriamente por el espació de soluciones, y cada vez irá acotando su movimiento hasta que solo pueda moverse si mejora el resultado. De esta forma, usaremos el algoritmo de generación aleatoria 4 (Genera una solución aleatoria, de forma que siga la filosofía del algoritmo de búsqueda, para no crear tendencias deterministas).

Planteamiento: Se realizan distintas ejecuciones con variaciones en los parámetros iniciales. Al principio estas variaciones son grandes, y a medida que vamos viendo qué valores nos dan mejores resultados, acotamos el área de búsqueda para mejorar la solución.

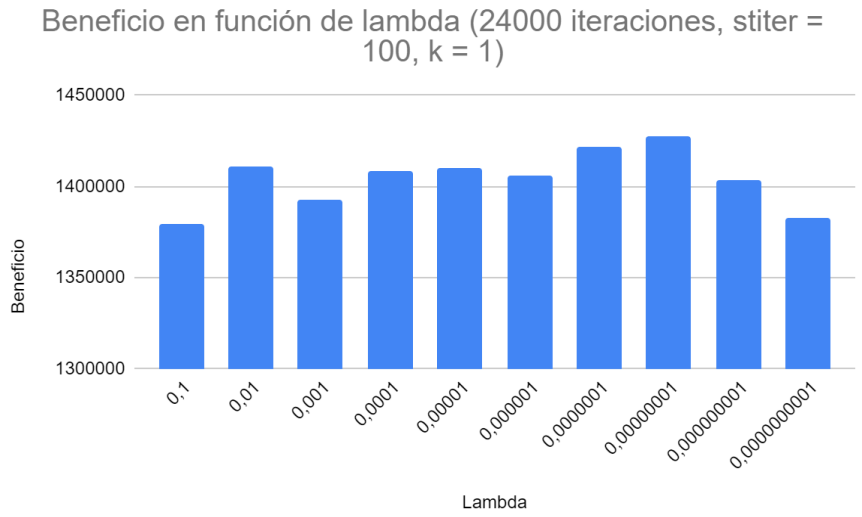
Hipótesis: Existe una combinación de parámetros que nos dé una soluciones mejores que el algoritmo *Hill Climbing* (H_o).

Método:

Al tener un algoritmo de búsqueda basado en la aleatoriedad, hemos supuesto que, por muy ajustados que estén los parámetros, puede darse una ejecución del algoritmo de búsqueda que nos dé un resultado atascado en un óptimo local. Para solventar este problema, hemos decidido que para cada variación del experimento se realizan 15 ejecuciones y se calculará la media de todas (Por lo que cada valor que se ve en este experimento es una media sacada de 15 ejecuciones con distintas semillas).

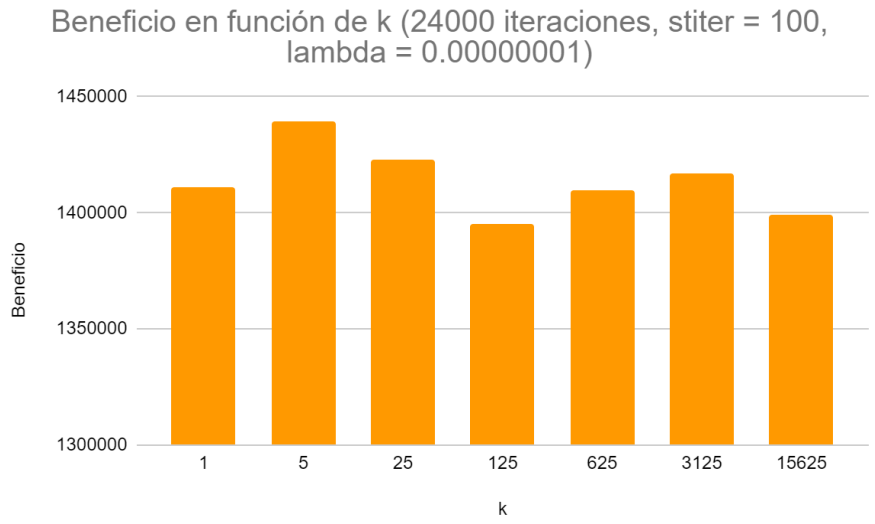
Para los juegos de pruebas, hemos usado los operadores *swap* y *move* (que fueron los que dieron los mejores resultados en el experimento 1) y la función heurística 3.

Para determinar los parámetros iniciales del *Simulated Annealing*, decidimos empezar los experimentos con 24000 ejecuciones (suficientes para dar tiempo a que la probabilidad de aceptación llegue a 0 y el algoritmo de búsqueda pueda escalar hasta el pico de la función heurística que pueda llegar). Empezamos por buscar el valor *lambda*, probando con los valores (0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001, 0.0000001).



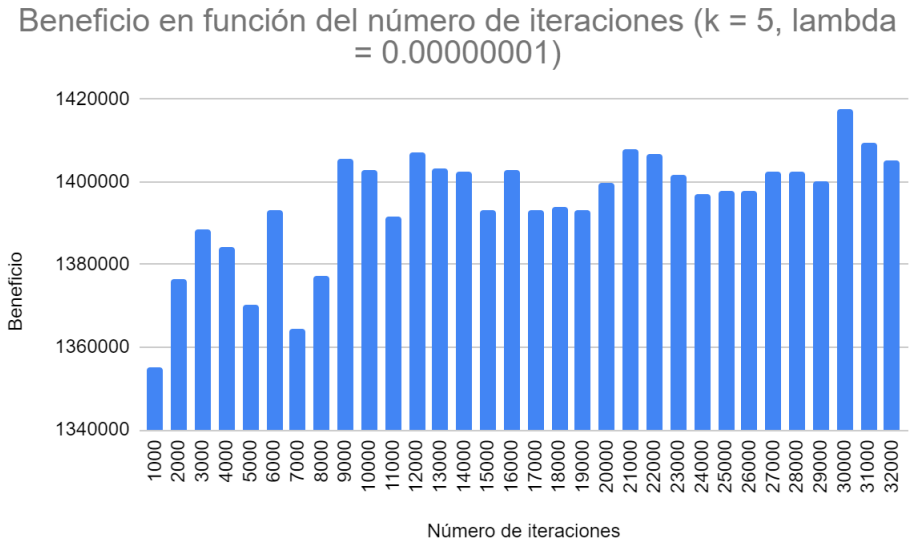
Como se puede observar, el valor de lambda que nos da mejores resultados es 0.00000001.

Partiendo de esta lambda, buscamos el valor de k entre los valores (1, 5, 125, 625, 3125,15625).



Se puede observar que nos da el mejor beneficio en *k* = *5*.

Teniendo *k* y *lambda* definidos, el siguiente paso es minimizar la cantidad de iteraciones sin que afecte significativamente al resultado del algoritmo. Todo ello con el objetivo de reducir el coste temporal de ejecución. Repetimos el experimento variando el número de iteraciones en el rango [1000, 32000]:



Como se puede ver, al reducir la cantidad de iteraciones, el beneficio no se ve afectado significativamente hasta llegar a 9000. Esto se debe a que al llegar a una solución que no puede mejorar (O dónde la cantidad de mejoras que puede hacer es muy limitada), se queda buscando mejores nodos sin mucho éxito hasta que terminan las iteraciones. De esta forma, al reducirlas a 22000 aumentamos la eficiencia de este algoritmo, sin afectar significativamente al beneficio de las soluciones que da.

El beneficio que nos da *Simulated Annealing* con esta combinación de parámetros (iteraciones = 9000, stiter = 100, k = 5, lambda = 0.00000001) varia entre [1,38 M y 1,42 M].

Por otro lado, el algoritmo *Hill Climbing*, nos da resultados entre los valores [1,35 M y 1,45 M]. Debido a esto vemos que los resultados concuerdan con lo esperado, ya que ambos algoritmos nos dan valores parecidos.

Experimento 4

Descripción: Dado el escenario de los apartados anteriores, estudiad cómo evoluciona el tiempo de ejecución para hallar la solución para valores crecientes de los 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, comenzad con 500 clientes e incrementad el número de 500 en 500 hasta que se vea la tendencia. Para 1000 clientes, comenzad con las 40 centrales iniciales e incrementad el número de 40 en 40 hasta que se vea la tendencia. Usad el algoritmo de Hill Climbing y la misma función heurística que antes

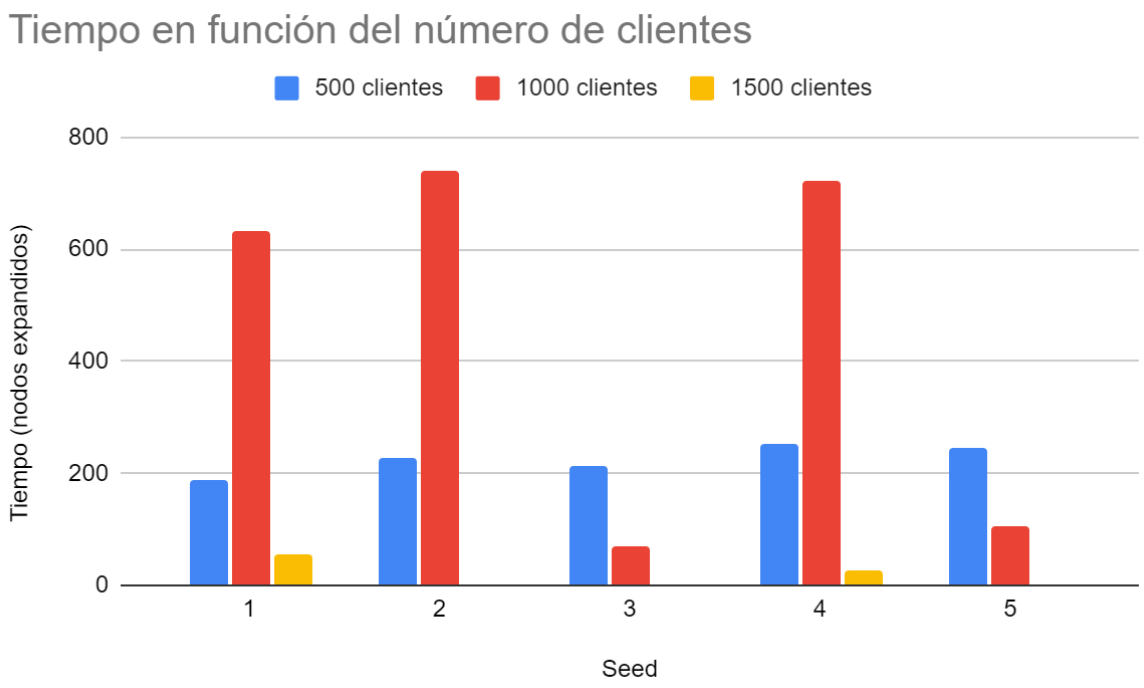
Planteamiento: Generamos experimentos con todos los valores y sacamos resultados y conclusiones.

Hipótesis: Existe una relación entre el tiempo de ejecución y los parámetros(H_o).

Método:

- Ejecutamos el programa con 40 centrales fijas y variamos los clientes de 500 en 500 para 5 semillas de tablero distintas.
- Repetimos el proceso, pero esta vez fijando los clientes a 1000 y variando las centrales de 40 en 40.

Resultados:

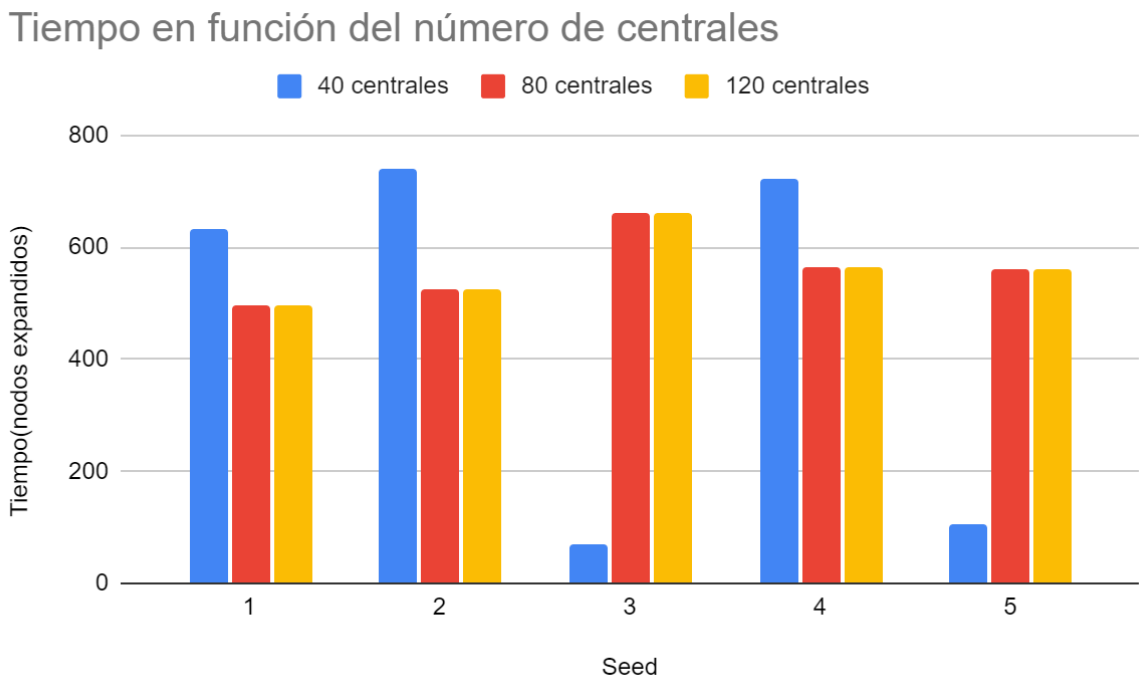


Resultados con 40 centrales y variando el número de clientes

En la primera tabla se pueden observar los resultados fijando el número de centrales a 40 y aumentando de 500 en 500 el número de clientes. En nuestro caso, por la forma en la que está hecha nuestra solución inicial, para asegurarnos de que todas las soluciones son válidas tenemos puesto un control en el que si la solución inicial no asigna a todos los clientes garantizados; el sistema acaba automáticamente la ejecución. Por esa misma razón, el programa no se ejecuta a partir de los 1500 clientes.

Por otra parte, podemos observar que en el caso de 1000 clientes, los resultados distan mucho entre ellos dependiendo de la semilla usada para generar la solución inicial. Esto se debe a que, al tener 1000 clientes, está relativamente cerca de no poder colocar a todos los clientes garantizados dentro de las centrales (Como se puede ver al tener 1500, que ya no es capaz de generar una solución inicial). Esto causa que, con algunas semillas (como la 3 o la 5) se generen soluciones iniciales muy cerradas, con poco margen de acción, lo que se acaba traduciendo en poca propagación del algoritmo.

Por último, tenemos el caso de los 500 clientes. A diferencia de los otros dos, con esta cantidad de clientes reducida, siempre es capaz de crear una solución inicial. Además, deja un gran margen de acción al no dejar casi todas las centrales cerca de su límite.



Resultados con 40 centrales y variando el número de clientes

En esta parte, tenemos un caso parecido al anterior. En el caso de tener 40 centrales, tenemos poco espacio de maniobra, por lo que la cantidad de nodos generados varía *mucho entre* sí de *una semilla a otra. Sin embargo en el caso de 80 y 120 centrales, tenemos mucho espacio libre en las centrales, lo que hace que la cantidad de nodos generados entre una semilla y otra sean parecidos.*

Los resultados encajan con lo esperado, al encontrar una relación entre los parámetros y el tiempo de ejecución, con lo que aceptamos H_o.

Experimento 5

Descripción: Habéis implementado el asegurar el servicio de los clientes con suministro garantizado usando la solución inicial y garantizando que se mantiene la restricción en la función generadora de sucesores. Otra manera de hacerlo es usar la función heurística, añadiendo una penalización a las soluciones donde esta restricción no se cumpla. En este caso se podrían considerar todas las soluciones válidas, por lo que la solución inicial podría ser vacía. Experimentad con diferentes valores para la penalización a no servir a estos clientes y, partiendo siempre de una solución vacía, usad los algoritmos del Hill Climbing y el Simulated Annealing para hallar una solución al problema. Pensad cómo afecta esto también a los operadores de búsqueda. Determinad el rango de valores a partir de los cuales la penalización permite obtener siempre una solución válida. Es decir, que se sirva a todos los clientes con suministro garantizado.

Observación: Puede haber un rango limitado de valores para la penalización con los que se obtienen soluciones válidas. Además, debemos adaptar los operadores para permitir quitar y añadir a clientes garantizados.

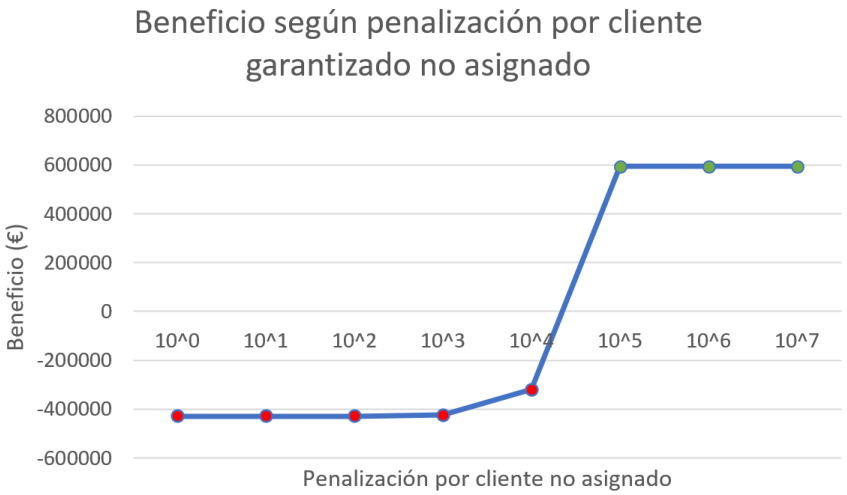
Planteamiento: Empezando desde una solución inicial, probamos varios valores de penalización por cliente garantizado y sacamos conclusiones.

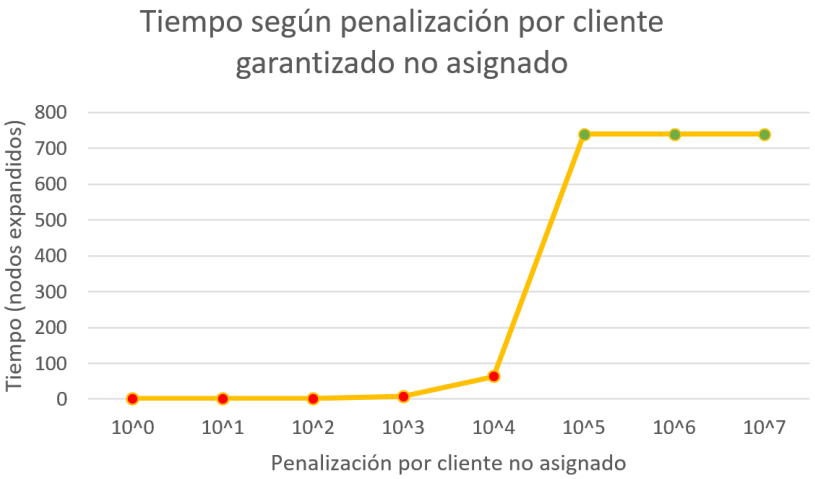
Hipótesis: Ejecutar el algoritmo de esta forma requerirá más tiempo, pero llegará a mejores soluciones (H_o).

Método:

- Se crea una solución inicial vacía.
- Añadimos a la heurística una penalización de 10^x por cada cliente garantizado no asignado.
- Hacemos experimentos variando la x de 0 a 9 con *Hill Climbing* y *Simulated Annealing.*
- Anotamos los resultados de:
    - Beneficio.
    - Tiempo (basado en cantidad de nodos expandidos).
    - Clientes asignados.
    - Centrales usadas.

Concluciones con el Hill Climbing:





Cabe destacar que todos los clientes asignados en nuestros resultados han resultado ser clientes garantizados. Esto se debe a que en un caso en el que se quiera asignar un cliente, con la heurística de este experimento siempre se escogerá antes a un cliente garantizado ya que dejará de sumarse una penalización enorme.

Como se puede apreciar fácilmente en las gráficas, sólo se generan soluciones buenas cuando la penalización supera los 10^5 por cliente garantizado no asignado. En el resto de casos, el beneficio es negativo: prácticamente no hay clientes asignados y el algoritmo apenas expande nodos.

En los rangos bajos de penalización, nos encontramos un número muy bajo de clientes asignados. Esto se debe a que, para añadir un cliente a una central (partiendo de una solución vacía) hay que activar dicha central; y la cantidad de megavatios perdidos sobrepasa la penalización que aplicamos por no tener asignado ese cliente. Esta relación entre “megavatios perdidos” y “penalización por cliente no asignado” se estabiliza cuando nos aproximamos a 10^4, y la penalización supera al otro valor a partir de 10^5.

Simulated Annealing

Al ser un algoritmo basado en cierto modo en la aleatoriedad, hemos decidido calcular cada valor como la media de 15 ejecuciones con distintas semillas.

Al ejecutar el algoritmo *Simulated Annealing* nos encontramos con un problema: nunca nos da soluciones válidas. Aunque cabe destacar que, al aumentar la penalización, se reducen los clientes garantizados no asignados.

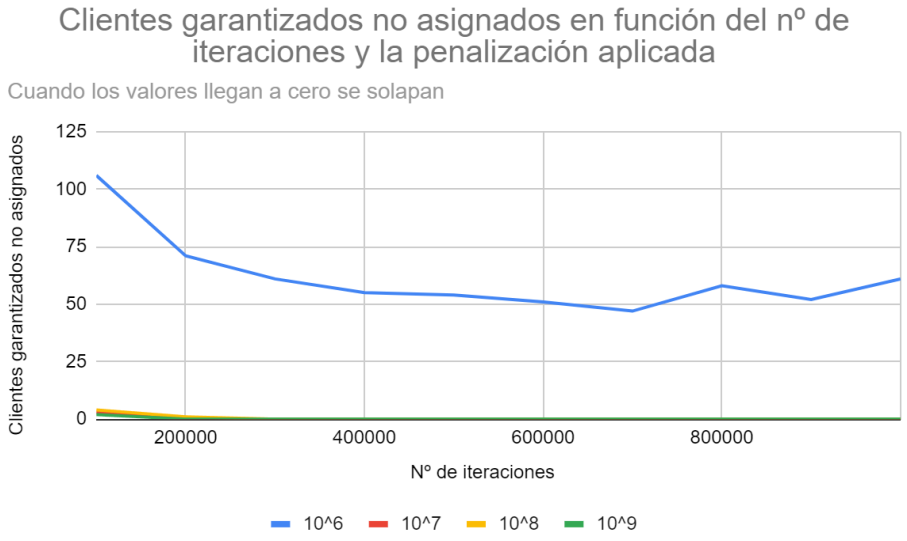
Se nos ocurrió que el problema podría deberse a dos factores:

- Al empezar sin tener clientes asignados, el algoritmo tenía que hacer más pasos para llegar a una solución, por lo que podría ser que no le diese tiempo a llegar a una solución válida.
- Al ser un algoritmo que se mueve por el espacio al principio de forma aleatoria y, al final escogiendo aleatoriamente los nodos a los que intenta ir (A diferencia del Hill Climbing, que siempre busca el mejor nodo), podría necesitar más intentos para progresar.

Ambos se pueden resolver aumentando el número de iteraciones del algoritmo. Debido a esto, decidimos hacer un experimento adicional para ver si, al aumentarlo, llegábamos a una solución válida.

Parámetros del experimento:

- Algoritmo a usar: *Simulated Annealing*
- *stiter = 100*, *k = 125*, *lambda = 0.0001* (Valores sacados del experimento 3)
- Iteraciones => Valor variable en el rango [100.000, 1.000.000]
- Penalización por cliente garantizado no asignado => Valor variable en el rango [1000000, 1000000000]
- Objetivo: ver la variación de clientes garantizados no asignados en función de las iteraciones del algoritmo y la penalización aplicada por cada cliente garantizado que no esté asignado.



En la gráfica anterior, se puede ver que, con una penalización menor de 10^8, el algoritmo siempre nos da soluciones no válidas. En el siguiente gráfico se muestran los mismos datos sin los resultados para la penalización de 10^6, donde se pueden observar mejor los datos de las otras penalizaciones:

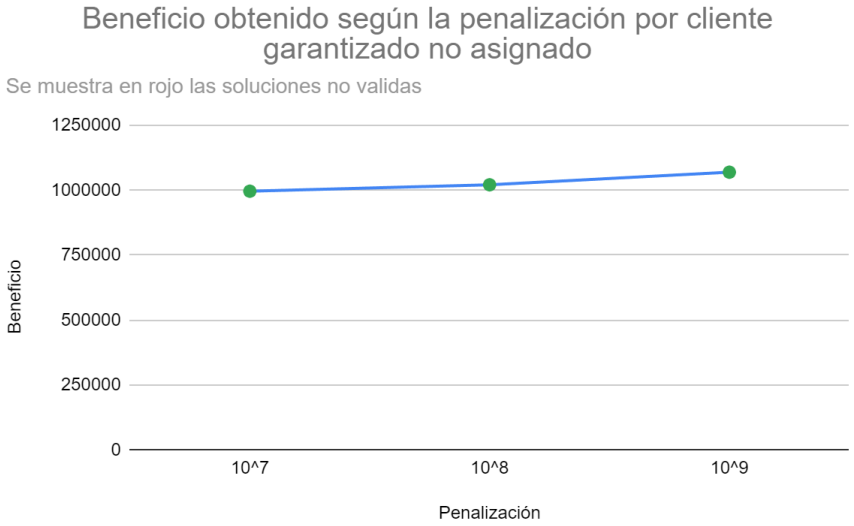
Cabe esperar que, si aumentamos más el número de iteraciones, las soluciones generadas con una penalización mayor o igual a 10^5, nos darán soluciones válidas, vista la tendencia vista con el *Hill Climbing*. Sin embargo, hay que tener en cuenta el coste temporal del algoritmo. Dado que aumentar la penalización no afecta en ninguna medida al tiempo de ejecución, pero aumentar las iteraciones afecta directamente, optamos por repetir la parte del *Simulated Annealing* de este experimento con las penalizaciones 10^7, 10^8 y 10^9.

De este modo, vemos que podemos usar el algoritmo *Simulated Annealing* para resolver el problema empezando con una solución vacía con ciertas limitaciones:

- Debemos usar una cantidad de iteraciones mayor (al menos 300.000, comparado con las 22.000 iteraciones que nos salieron en el experimento 3).
- En caso de 300.000 iteraciones, debemos tener una penalización mayor o igual a 10^7.

Teniendo lo dicho anteriormente en cuenta, repetimos el experimento 5:

Planteamiento: Visto que al llegar a 300.000 iteraciones el algoritmo es capaz de generar soluciones válidas, decidimos ejecutarlo con 400.000, para darle espacio al *Simulated Annealing* a mejorar la solución una vez llegada a una válida.



En este caso, vemos que todas las soluciones son válidas. Además, el beneficio logrado por todos las penalizaciones es aproximadamente el mismo. Esto se debe a que la penalización es una forma de decirle al algoritmo la importancia que tienen los clientes garantizados, pero no afecta directamente al beneficio aportado por las soluciones. Una vez todas las soluciones contienen todos los clientes garantizados, el hecho de que la penalización sea mayor no interfiere en la ejecución.

Los resultados no son los esperados, ya que el beneficio logrado por ambos algoritmos es menor que el que hemos obtenido en los experimentos anteriores.

Experimento 6

Descripción: Un elemento clave en el problema es la proporción de los tipos de centrales eléctricas. Tener más o menos centrales más pequeñas distribuidas geográficamente puede reducir las pérdidas en el transporte y por lo tanto suministrar a más gente de manera más eficiente. Partiendo del escenario inicial, experimenta si duplicando y triplicando las centrales de tipo C hace que las centrales de tipo A y B se usen menos. Usad los algoritmos de Hill Climbing y el Simulated Annealing y la función heurística original. Observad también cual es el aumento de coste en tiempo para hallar la solución.

Observación: Aunque el coste de producción por MW de las centrales de tipo C es mayor que el de los otros dos tipos de centrales, el coste de parada y el coste base de producción es menor.

Tipo	Producción	Coste marcha	Coste parada
A	250 a 750 Mw	Prod*50+20000	15000
B	100 a 250 Mw	Prod*80+10000	5000
C	10 a 100 Mw	Prod*150+5000	1500

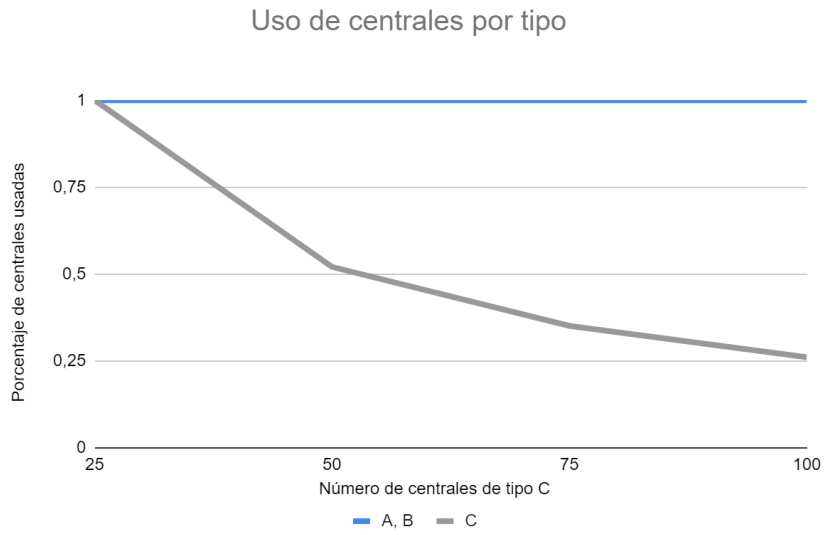
Planteamiento: Ejecutamos los dos algoritmos variando la cantidad de centrales de tipo C.

Hipótesis: Aumentar las centrales de tipo C reducirá el uso de los otros dos tipos (H_o).

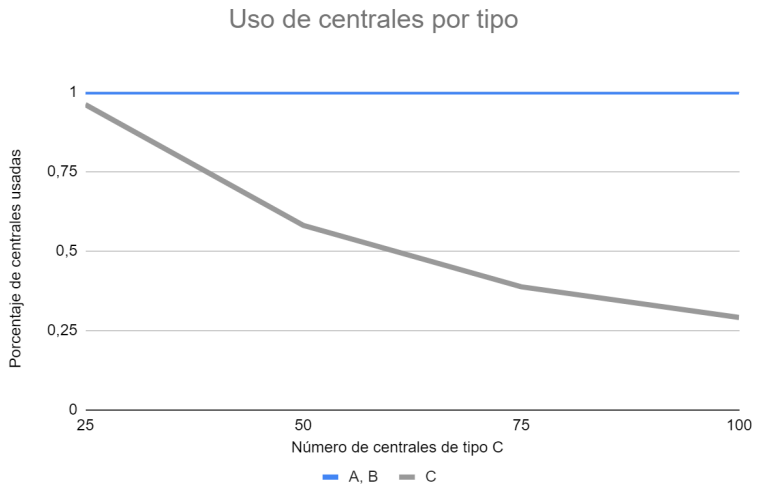
Método:

- Se asigna:
  - 5 centrales de tipo A
  - 10 centrales de tipo B
  - x centrales de tipo C (donde x varía entre 25, 50, 75, 100)
- Se crea una solución inicial válida
- Se ejecuta el algoritmo con los operadores *swap* y *move* y la función heurística *3*

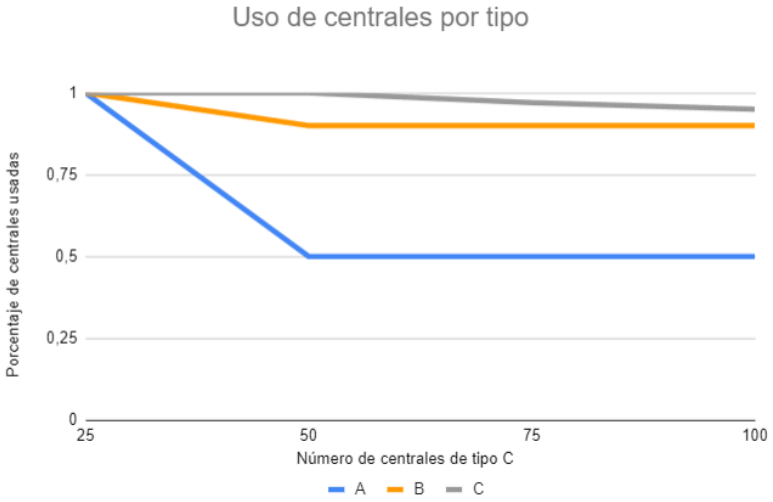
Hill Climbing



Simulated Annealing



En ambos casos se puede ver que hay una progresión prácticamente idéntica en el porcentaje de centrales usadas de cada tipo, aunque estos resultados no són los que creemos que són óptimos teniendo en cuenta las características de cada tipo de central. Los resultados no encajan con lo esperado. Por esta razón hemos creado el siguiente gráfico, que muestra el mismo test hecho con Hill Climbing, pero usando otra solución inicial (*getSolIniOrdenadaTodos*).



El resultado es completamente distinto, así que llegamos a las siguientes conclusiones:

- Con la primera solución inicial (*getSolIniAsignadosTodos*) asignamos todos los clientes a centrales en el mismo orden en el que están en el vector creado por las clases AIMA. Este vector contiene primero las centrales de tipo A, luego las de tipo B y finalmente las de tipo C. Nuestra heurística no contiene ningún factor que favorece vaciar centrales, es por esta razón por la que prácticamente no apaga ninguna central y genera resultados en los que simplemente no asigna las centrales de tipo C que són las que se posicionan al final del vector. Por esta misma razón el gráfico de

la primera solución inicial muestra cada vez menos porcentaje de centrales de tipo C usadas, al encontrarse al final del vector la solución inicial las dejará apagadas.

- Con la segunda solución inicial (*getSolIniOrdenadaTodos*) asignamos los clientes intentando minimizar la distancia entre estos y las centrales, por tanto, se distribuyen mucho más los clientes entre las centrales de todos los diferentes tipos. Gracias a esta gráfica podemos observar como el Hill Climbing intenta minimizar la cantidad de clientes asignados a centrales de tipo A, ya que son las centrales que más cuesta mantener encendidas cuando no están proporcionando toda su energía generada. En cambio, las centrales de tipo C son las que presentan menos pérdidas cuando no están en su máxima capacidad.

Comparación de los dos algoritmos

Después de haber realizado los experimentos anteriores podemos proceder a comparar los algoritmos de Hill Climbing y Simulated Annealing, para concluir cuál es mejor para el problema de este trabajo.

Para compararlos, nos vamos a centrar en los dos factores más importantes: el tiempo y la bondad de la solución, (es decir, el beneficio final).

Como hemos observado en el experimento 3, al experimentar con diferentes valores para los parámetros del Simulated Annealing, hemos concluido que los valores para k y λ deben ser 10^{-8} y 5 respectivamente, para maximizar la eficacia del algoritmo

Coste temporal de la búsqueda

Durante los experimentos, hemos podido comprobar que el algoritmo de Hill Climbing parece tardar más en la ejecución de la búsqueda. Hemos decidido hacer un nuevo experimento para poder comparar explícitamente los algoritmos, con 10 semillas y las mismas condiciones para todos los parámetros

Tiempo en nodos expandidos		
Seed	**Hill Climbing**	**Simulated Annealing**
1234	16339	30
1	634	651
2	739	198
3	70	11
4	722	12
5	106	11
6	91	10
7	111	9
8	120	12
9	705	9
Media	1963,7	102,5555556
Desv Est.	5059,855818	214,8075366

Como se puede apreciar, el Simulated Annealing tarda muchísimo menos, prácticamente un 5% de lo que tarda el Hill Climbing

Bondad de las soluciones

Para el experimento anterior también hemos ido apuntando el beneficio final.

Tiempo en nodos expandidos		
Seed	**Hill Climbing**	**Simulated Annealing**
1234	1441767	1452774
1	1473729	1473729
2	1405671	1396216
3	1427194	1426096
4	1439182	1439261
5	1370578	1356423
6	1352897	1352717
7	1384570	1386111
8	1344642	1381993
9	1396602	1381848
Media	1403683,2	1399377,111
Desv Est.	41833,23675	39765,73275

Como se puede apreciar en la tabla anterior, el Simulated Annealing obtiene soluciones prácticamente idénticas que el Hilll Climbing.

Pese a obtener unas soluciones ligeramente peores, el algoritmo de Simulated Annealing es mucho más apropiado para este problema al ser muchísimo más rápido.

Apéndices

Proyecto de Innovación

Descripción del tema:

DLSS es una tecnología desarrollada por Nvidia, que permite, mediante la inteligencia artificial, incrementar la resolución de imágenes intentando mantener su calidad. De esta forma, permite que las tarjetas gráficas generen imágenes en resoluciones más bajas, lo que aumenta la eficiencia, sin afectar a la calidad de imagen.

Referencias:

- Wikipedia: Deep learning super sampling:

  https://en.wikipedia.org/wiki/Deep_learning_super_sampling

- TensorFlow: autoencoder

  https://www.tensorflow.org/tutorials/generative/autoencoder

- DataScientest: Convolutional neural network

  https://datascientest.com/es/convolutional-neural-network-es

- Nvidia: DLSS

  https://www.nvidia.com/en-gb/geforce/technologies/dlss/

- Nvidia: AI-Powered Neural Graphics

  https://www.nvidia.com/en-us/geforce/news/dlss3-ai-powered-neural-graphics-innovations/

- OpenVC: Optical Flow

  https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html

- Youtube Nvidia: ¿Que es DLSS?

  https://www.youtube.com/watch?v=rjWh3ZtyXGg