Universitat Politècnica de Catalunya

Inteligencia artificial

Bachelor Degree in Computer Science

Centrales

Autores:
Carles Balsells Rodas
Miguel Cidrás
Cesc Folch Aldehuelo
Dean Zhu

Q1 Curs 2018-2019

Contents

1 Introducción

Esta práctica tiene como objetivo resolver un problema surgido del mercado de la energía: qué centrales tener en funcionamento y qué centrales tener paradas dependiendo de la demanda a de parte de los consumidores. Para ello, se implementará una inteligencia artificial que resolverá mediante algoritmos de búsqueda local.

En primer lugar, haremos una descripción del problema y definiremos los estados del problema, su representación y el espacio de búsqueda, los operadores para desplazarnos por el espacio de búsqueda, la función heurítica y la generación del estado inicial.

A continuación, pasaremos a la parte experimental, en la que de forma empírica, observaremos las influencias de la solución inicial, los operadores y la función heurística, además de comparar el algoritmo de *Hill Climbing* contra el algoritmo de *Simulated Annealing* para observar qué algoritmo tiene mejores resultados, y razonar sobre las posibles decisiones que se han de tomar respecto a los algoritmos.

2 Descripción del problema

2.1 Identificación del problema

2.1.1 Características del problema

El problema a tratar tiene como elementos los siguientes:

- El problema considera un área de 100x100 Km^2
- Existen M **centrales**, con sus respectivas localizaciones y que tienen cierta producción máxima. Según esta producción, se definen 3 tipos de centrales.
- Existen N **clientes**, que pueden tener un servicio garantizado o no, y que tienen cierta demanda energética. Que un cliente tenga un servicio garantizado significa que su demanda ha de ser satisfecha, mientras que si es no garantizado, que si la demanda no es satisfecha, el cliente recibirá una indemnización. Paralelamente a las centrales, según la demanda, se definen 3 tipos de clientes.

Además, existen una serie de restricciones que se han de satisfacer:

- Cada cliente está conectado a una sola central y esta lo ha de servir completamente.
- Una central no puede satisfacer más demanda energética que su producción máxima.
- Existe una pérdida de energía en función de la distancia entre la central y el cliente. De este modo, la central ha de suministrar más energía a un cliente lejano que a uno cercano para satisfacer la misma demanda.

Teniendo en cuenta el coste de las centrales, la pérdida de energía y las distintas tarifas e indemnizaciones de los clientes, podemos definir cual es el beneficio que se obtiene de cierta configuración. Nuestro problema trata de encontrar la configuración que aporte más beneficio.

2.1.2 Justificación de búsqueda local

En este problema tenemos que decidir para cada cliente si se le asigna o no una central y, en el caso de que se le asigne una, cuál. De este modo, observamos que el tamaño del espacio de soluciones es muy grande como veremos en la siguiente sección, por lo que obtener el óptimo se convierte muy difícil y una exploración sistemática en el espacio de soluciones es prácticamente imposible.

De este modo, se debe de utilizar otro acercamiento, que use cierta función heurística para podar soluciones en el espacio de búsqueda y no acceder a soluciones que no merece la pena explorar.

Asimismo, tampoco necesitamos saber la historia del camino recorrido.

Por todo ello, consideramos que este es un problema de búsqueda local y, por lo tanto, debería de poder ser resuelto con los algoritmos de *Hill Climbing* o de *Simulated Annealing*.

2.2 Estados del problema y su representación

Definiremos un **estado** como una asignación a cada cliente de servicio garantizado de una central y a cada cliente de servicio garantizado de una central o de ninguna. De este modo, nos aseguramos que solo una central suministre energia a cierto cliente y que todos los clientes de servicio garantizado sean satisfechos. Notemos que aún es necesario comprobar que la producción de una central no supere su máximo, teniendo en cuenta de satisfacer completamente la demanda de sus clientes y la pérdida de la energía debido a la distancia cliente-central.

2.2.1 Espacio de soluciones

Analizaremos cuál es el espacio de soluciones.

Consideremos el problema donde hay M centrales y N clientes, donde N_1 clientes tienen demanda garantizada y $N_2 = N - N_1$ no.

Nos fijamos que en la configuración final podemos asociar cada cliente garantizado una central, y a cada cliente no garantizado una central o que no se le suministre la energía requerida. Por tanto en el peor de los casos tendremos $N_1^M + N_2^{M+1} \approx N^M$. Además sabemos que este problema es NP, y por tanto cualquier solución debera navegar por todo o gran parte del espacio de soluciones, y la tamaño de ella crece exponencialmente respecto el número de centrales y fuertemente polinómica respecto el número de clientes.

2.2.2 Implementación del estado

Teniendo en cuenta que el estado final lo hemos representado como el conjunto de asignaciones cliente-central, la implementación de los estados más natural es ésta misma, que es representarlo como un **vector de N elementos** que representan las asignaciones. Este vector satisface que el elemento i-ésimo representa la central que suministra energía al cliente i-ésimo; teniendo en cuenta de que si el elemento es -1, significa que al cliente i-ésimo no le aportan energía. El coste espacial esperado es, por tanto, $O(N)$.

Además, para mejorar el tiempo de ejecución, decidimos implementar el cálculo de las funciones heurísticas en el estado. Para ello, le hemos añadido al estado, dos variables y dos estructuras auxiliares, que son:

- Una variable que guarda el valor de la ganancia de cierta configuración

- Una variable que tiene como valor de la heurística.
- Un vector de M elementos que sirve para mantener actualizado cuál es la demanda que ha de satisfacer cada central.
- Un vector de M elementos que es usada para saber cuántos clientes tienen asignados cierta central en particular.

De este modo, el coste espacial esperado es $O(N + M)$. Es importante observar que N es significativamente mayor a M, por lo que esta implementación del estado solo supone un pequeño aumento en el coste espacial.

2.3 Elección y generación del estado inicial

Para ejecutar un algoritmo de búsqueda local, primero precisamos una solución inicial. En un inicio, implementamos siete inicializaciones para ver como se comportan:

- *ONE*: el estado inicial es asignar una central escogida de forma aleatoria a todos los clientes.
  El coste temporal es $O(N)$.
- *RANDOM*: El estado inicial es asignar a cada cliente, una central de forma aleatoria y equiprobable.
  El coste temporal es $O(N)$.
- *RANDOM2*: el estado inicial es similar a la inicialización *RANDOM*, pero asignando una central de forma aleatoria y equiprobable solo a los clientes que tenga servicio garantizado; si tiene servicio no garantizado, entonces no se le asigna ninguna central.
  El coste temporal es $O(N)$.
- *FUZZY*: el estado inicial es asignar una central a cada cliente, pero asignando distintos pesos a cada central. Los pesos se asignan de forma linealmente proporcional a la producción de la central y cuadráticamente proporcional al porcentaje que no se perdería desde la central hasta el cliente.
  El coste temporal de nuestra implementación es $O(NM)$.
- *FUZZY2*: inicializa de la misma forma que *FUZZY*, pero solo a los clientes con servicio garantizado. Al resto, no les asigna ninguna central.
  El coste temporal de nuestra implementación es $O(NM)$.

Dado que el algoritmo de Hill Climbing simplemente encuentra máximos/mínimos locales, la única forma de inicializar los estados centrales que permite ejecutar varias veces el algoritmo de busqueda local y que converjan en distintos óptimos locales. Además dado a resultados que obtendremos de forma empírica obserevaremos que la inicialización que más nos conviene es la *FUZZY*.

2.4 Representación y análisis de los operadores

Para poder movernos sobre el espacio de soluciones debemos definir unos operadores, notemos que la elección del operador tendrá una consecuencia directa sobre el factor de ramificación. Los operadores que consideraremos son los siguientes:

- *setCentral(id, jd)*: asigna el cliente *id* a la central *jd*. Tiene una ramificación de $O(NM)$.

- *swapCentral(id1, id2)*: intercambia las asignaciones de los clientes *id1* y *id2*. Tiene un factor de ramificación de $O(NN)$.
- *substituteClient(id1, id2)*: Asigna al cliente *id1* a la central que ocupaba el cliente *id2* y desasigna al cliente *id2*. Tiene un factor de ramificación de $O(NN)$.

Notemos que cualquier operador que actúe sobre k clientes tendrá un factor de $\binom{n}{k}$, el cual es excesivo en la gran mayoría de casos. Además es díficil crear operadores sobre centrales, pues por ejemplo, vaciar una central y reasignar las centrales asignadas también supone un problema NP.

El primer operador es también el más natural, simplemente asignamos un cliente a un sitio mejor. Pero cabe mencionar que usando solamente este operador vemos que hay muchos estados que serán óptimos locales, de hecho, cualquier estado en que las centrales estén saturadas la serán. Esto hace que el Hill Climbing se pare con mucha facilidad ya que los estados vecinos a un estado con centrales saturadas forman un espacio con forma de cresta.

Por tanto parece lógico necesitar operadores que nos permitan mejorar la solución una vez llegado a este caso. El swap central nos permite redistribuir los clientes si hace que haya menos pérdidas en el transporte, y por tanto en las centrales haya menos malgasto de electricidad lo cual significa una menor saturación.

Finalmente el operador de substituteClient también permite movernos de forma lateral en el espacio de soluciones, es decir, nos permite mejorar la solución sin tener que aumentar el número de asignaciones y por consiguiente saturar las centrales.

2.5 Análisis de la función heurística

El problema ya define una función numérica que se ha de maximizar, que es la *ganancia* en euros que produce una configuración de clientes-centrales. De esta forma, hemos de minimizar la *-ganancia*. Para poder navegar sobre los estados consideraremos una función h que penalizará sobre el valor de la ganancia.

A partir de ello hemos definido tres funciones heurísticas:

- La suma de la *-ganancia* y el cuadrado de la sobreproducción de cada central, multiplicado por 50000 (que es mayor que la ganancia máxima de una central) y el número de centrales. Esto nos garantiza que el estado final sea válida pues ninguna central podrá tener una sobreproducción. Además usaremos el cuadrado de la sobre producción con el fin de que se penalice más una central que sobreproduzca mucho, y no varias que sobreproduzcan un poco. Pues tras arreglar centrales con mucha sobreproducción no sería de extrañar que creemos otras centrales que sobre produzcan.
- La suma de la *-ganancia* y el cuadrado del número de clientes que están asignados a una central con sobreproducción, multiplicado por 50000 (que es mayor que la ganancia máxima de una central). Esta función llega a un estado final válido porque penaliza a las centrales que sobreproducen y tenderá reducir el número de clientes en centrales que sobreproducen, ya que la penalización aumenta mucho que una central que sobreproduce a que no (ya que es cuadrática respecto al número de clientes en centrales que sobreproducen).
- Es la *-ganancia* cuando no hay ninguna central con sobreproducción y si hay alguna central con sobreproducción, entonces es el cuadrado de la sobreproducción de cada central, multiplicado por 50000 (que es mayor que la ganancia máxima de una central) y el número de centrales. Esta heurística es una versión de la primera; en esta versión tiene como objetivo que la penalización

por sobreproducción sea mayor que en la heurística original, para lo cual se elimina el factor de la ganancia de la heurística.

Es muy importante comentar que el cálculo de la función heurística tiene que ser lo más eficiente posible; ya que la generación de un gran número de estados sucesores cuya función heurística sea poco eficiente haría que el tiempo de ejecución del problema sea muy grande. Por lo que respecta a nuestra implementación del cálculo función heurística, se ha definido su resultado como un parámetro más del propio estado, que se actualiza en tiempo constante al ejecutar un operador de cambio de estado.

3 Parte experimental

3.1 Sobre los experimentos en general

En general, al menos que se indique lo contrario, realizaremos nuestros experimentos en el mismo 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.

3.2 Experimento 1: Influencia de los operadores

Para realizar una buena búsqueda local es fundamental tener definida una buena función generadora de estados sucesores. Esta, está compuesta por un conjunto de operadores que permiten generar nuevos estados. En este experimento vamos a partir de las definiciones de los operadores que hemos visto anteriormente para un estado, y se van a proponer funciones generadoras de estados sucesores escogiendo diferentes conjuntos de operadores y aplicándolos con distintas estrategias.

En general, el objetivo de este experimento es el de encontrar cúal es el conjunto de operadores que proporciona mejores resultados en la ejecución del algoritmo de Hill Climbing (HC). A su vez, vamos a aprovechar este punto para comparar las tres funciones heurísticas descritas anteriormente y ver cuál es mejor de cara a la relación de la ganancia y el tiempo de ejecución. Los experimentos posteriores van a realizarse escogiendo la heurística y la función de generación de estados sucesores que tengan una mejor ejecución en este apartado.

Concretamente, para este experimento se han propuesto tres funciones generadoras de estados sucesores, que se definen de la siguiente forma:

1. Aplicación del operador *setCentral(id, jd)*, para cada cliente *id-ésimo* y cada central *jd-ésima*. Es un operador de implementación simple, por lo tanto es interesante estudiar su comportamiento. El coste de esta función de estados sucesores es *O(NM)*.
2. Para mejorar el comportamiento de solo usar el operador *setCentral(id, jd)* descrito en el apartado de operadores, creamos una segunda función sucesora que hará lo mismo que la anterior pero además usará los otros dos operadores. Considerará el intercambio de dos centrales y considerá la sustitución de los clientes en una central. Deberíamos ver si añadiendo estos dos cambios conseguimos navegar a mejores extremos locales. Su factor de ramificación es de $O(N*M+N*N)$, pero dado a que el número de clientes es mayor al de centrales, para mejorar el rendimiento consideraremos como mucho 10000 intercambios, y como mucho 100 sustituciones por cliente, además las sustituciones solo tendrán sentido si asignaremos un cliente en una central saturada a una no saturada. Por tanto el factor de ramificación final será de $O(N*M+100*N+10000)$

3. Aplicación del operador *setCentral(id, jd)*, para cada cliente *id-ésimo* y cada central *jd-ésima* que no tenga sobreproducción. Luego, para cada cliente, se ejecuta el operador *swapCentral(id1,id2* para hacer un intercambio de centrales entre la central del cliente *id1-ésimo* y la central del cliente *id2-ésimo*. Se realiza esta operación para cada cliente *id1-ésimo* y central, por lo tanto el coste de esta función es *O(NM)*.

En referencia a las funciones heurísticas que se usarán para este experimento, éstas han sido analizadas anteriormente en la sección *2.5*. La solución inicial utilizada en este experimento será la *FUZZY*, definida en la sección *2.3*. Es importante mencionar que muchas veces el algoritmo de Hill Climbing utilizado en este experimento no va a encontrar un mínimo local que corresponda a una solución correcta dependiendo del par función heurística y conjunto de operadores seleccionado. Recordemos que una solución es correcta si parte de la solución inicial y además no tiene sobreproducción en las centrales.

Una vez tenemos definidas las funciones generadoras de estados sucesores y las funciones heurísticas, vamos a plantear la ejecución del experimento.

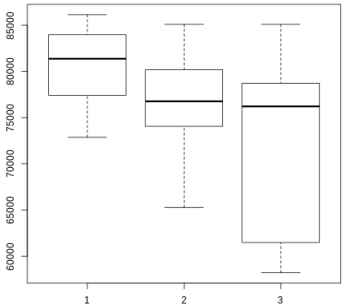
Observación	Existen conjuntos de operadores que funcionan mejor que otros.
Planteamiento	Queremos ver cúal es el mejor operador para la mayoría de heurísticas planteadas.
Hipótesis	• Los tres conjutos de operadores son iguales en ganancia (H0) o uno de ellos es mejor que el resto • Todos los conjuntos conducen a un estado final correcto (H0) o alguno de ellos no lo hace • El tiempo de ejecución es el mismo (H0), o algún conjuto de operadores encuentra un mínimo local con más rapidez
Método	• Escogemos 10 semillas aleatorias. • Experimentaremos con problemas con el escenario planteado en la sección *3.1*. • Ejecutaremos un experimento con el algoritmo de HC para cada semilla, conjunto de operadores y función heurística; partiendo de la solución inicial *FUZZY* • Para cada ejecución obtendremos el valor de la ganancia, el tiempo de ejecución y informaremos sobre si la solución es correcta, es decir, si no tiene sobreproducción en alguna de sus centrales.

Después de realizar la ejecución descrita anteriormente, se han obtenido los siguientes resultados:

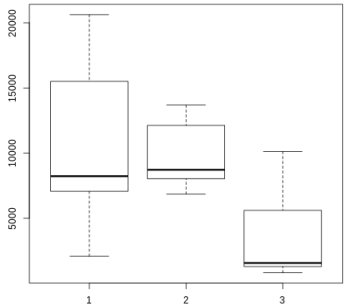
3.2.1 Influencia con la primera función heurística

	Conjunto 1			Conjunto 2			Conjunto 3		
seed	ganancia	tiempo	correcto	ganancia	tiempo	correcto	ganancia	tiempo	correcto
1183	83267.82	8250	no	80182.37	12127	sí	76311.97	1561	no
1556	80547.45	6819	no	77478.60	7811	sí	76867.10	5607	sí
533	82201.82	20624	no	76051.37	12767	sí	61489.82	1119	no
182	85094.36	2085	sí	85094.36	8041	sí	85094.36	2312	sí
1131	77680.39	13282	no	75779.99	13690	sí	68258.24	1436	no
554	77422.08	20406	no	66887.28	6851	sí	58216.88	818	no
880	83982.11	7439	no	78962.11	8592	sí	78727.41	1294	no
1241	86140.82	7072	no	82865.77	8816	sí	82659.52	6068	sí
490	77394.83	8212	no	74063.38	8703	sí	76120.58	10119	sí
448	72870.74	15507	no	65282.94	8739	sí	61450.74	1576	no

En la tabla superior podemos observar que el primer y tercer conjunto de operadores no encuentran una solución correcta en la mayoría de sus ejecuciones para el caso de la primera función heurística. A continuación, hemos representado los valores obtenidos en un *boxplot* para una mejor interpretación.



(a) Gananancia de los operadores.



(b) Tiempo de ejecución, en ms.

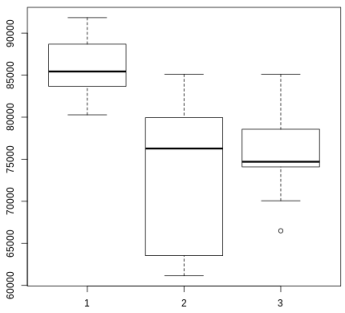
Figure 1: Resultados para la primera función heurística

A partir de los resultados obtenidos, podemos ver que el primer y tercer conjunto de operadores nos proporcionan un menor tiempo de ejecución. Además, el primer conjunto de operadores consigue obtener una mejor ganancia en promedio. En resumen, no podemos considerar válidos los dos conjuntos anteriormente mencionados ya que, al contrario que con el segundo conjunto, no siempre dan una solución correcta. Por lo tanto para esta función heurística nos quedaríamos con el segundo conjunto de operadores.

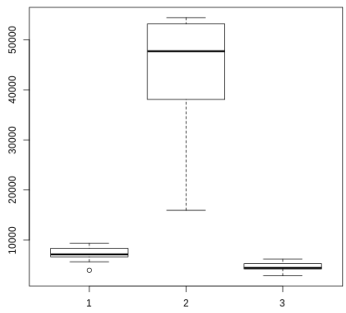
3.2.2 Influencia con la segunda función heurística

	Conjunto 1			Conjunto 2			Conjunto 3		
seed	ganancia	tiempo	correcto	ganancia	tiempo	correcto	ganancia	tiempo	correcto
1183	87846.57	7020	no	79823.42	50877	sí	78570.17	5270	no
1556	85100.10	6623	no	78148.20	38096	sí	74502.00	4594	no
533	91834.52	8325	no	63543.97	54438	sí	74523.02	3964	no
182	85094.36	3948	no	85094.36	15929	sí	85094.36	2841	no
1131	74886.00	9339	no	72355.39	53487	no	68258.24	6163	no
554	70064.53	9275	no	61144.38	49251	sí	58216.88	4410	no
880	77189.31	7061	no	79945.81	46214	sí	78727.41	5624	no
1241	83366.82	5639	no	84841.12	37943	sí	82659.52	4421	no
490	74090.93	7353	no	74405.93	45108	sí	76120.58	4204	no
448	66492.39	7171	no	62663.19	53191	sí	61450.74	4420	no

A diferencia del caso anterior, podemos ver que ahora el primer y tercer conjunto de operadores no alcanzan una solución correcta en ninguna de las ejecuciones. Además, podemos encontrar un caso en el cual para el segundo conjunto se encuentra un mínimo local que corresponde a una solución incorrecta. Por lo tanto, a priori podemos pensar que la segunda función heurística no penaliza lo suficiente le hecho que una central tenga superproducción.



(a) Gananancia de los operadores.



(b) Tiempo de ejecución, en ms.

Figure 2: Resultados para la segunda función heurística

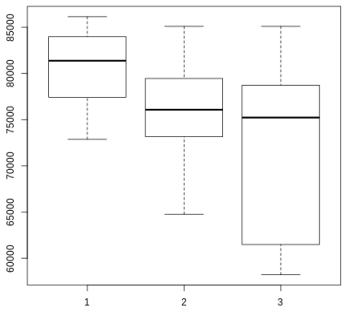
En la figura **??**, podemos ver la representación de los valores mostrados anteriormente. Para el caso de la ganancia, se sigue una tendencia parecida al caso de la primera función heurística (fig. **??**). Además, respecto al tiempo de ejecución, se puede observar que para esta función heurística, el tiempo de ejecución para el segundo conjunto de operadores es de casi un minuto en promedio. Por lo tanto, la segunda función heurística no aporta ninguna mejora si lo comparamos con el caso anterior.

Respecto al tiempo de ejecución del resto de conjunto de operadores, no podemos comentar nada a su favor (aunque sea menor), ya que los mínimos locales encontrados para estos conjuntos son estados que describen soluciones incorrectas.

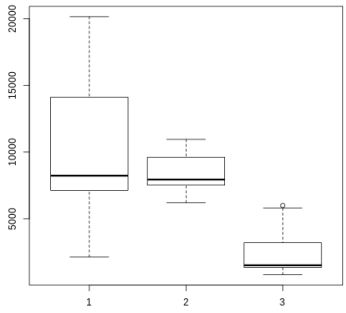
3.2.3 Influencia con la tercera función heurística

	Conjunto 1			Conjunto 2			Conjunto 3		
seed	ganancia	tiempo	correcto	ganancia	tiempo	correcto	ganancia	tiempo	correcto
1183	83267.82	8179	no	79455.37	10953	sí	76311.97	1561	no
1556	80547.45	7135	no	77478.60	7407	sí	76867.10	5607	sí
533	82201.82	19514	no	74670.22	9608	sí	61489.82	1119	no
182	85094.36	2138	sí	85094.36	7753	sí	85094.36	2312	sí
1131	77680.39	12718	no	73764.64	9741	sí	68258.24	1436	no
554	77422.08	20150	no	66887.28	6207	sí	58216.88	818	no
880	83982.11	7299	no	78962.11	7960	sí	78727.41	1294	no
1241	86140.82	7034	no	82210.72	8218	sí	82659.52	6068	sí
490	77394.83	8285	no	73164.43	7923	sí	76120.58	10119	sí
448	72870.74	14118	no	64764.54	7525	sí	61450.74	1576	no

Excepto por diferencias muy puntuales, los resultados que hemos obtenido para la tercera función heurística son casi idénticos en comparación a los que se han obtenido en el primer caso.



(a) Gananancia de los operadores.



(b) Tiempo de ejecución, en ms.

Figure 3: Resultados para la tercera función heurística

Como podemos ver, los resultados son los mismos que se han obtenido para el caso de la primera función heurística. De hecho, este resultado no debería sorprendernos; ya que como hemos visto en secciones anteriores, la primera y la tercera función heurística tienen un comportamiento muy similar.

3.2.4 Selección final

Después de observar el comportamiento que cada conjunto de operadores tiene dependiendo de la función heurística que se seleccione, es hora de escoger cual es la pareja de función generadora de estados sucesores y función heurística que mejores resultados nos da (tanto en ganancia como en tiempo de ejecución).

En referencia a las funciones generadoras de estados sucesores, nos vemos obligados a descartar la primera y la tercera de las que hemos definido, ya que en algunas de sus ejecuciones se llegaba a una solución incorrecta y esto no se puede permitir en ningún caso.

Por otra parte, una vez hemos fijado el segundo conjunto de operadores, vamos a ver cual es la función heurística que más nos conviene. Puesto que la primera y la tercera daban casi los mismos resultados, podemos escoger cualquiera de las dos. En comparación a la segunda función heurística, a parte de generar menos estados que describían soluciones correctas, para los casos con soluciones incorrectas su

tiempo de ejecución era demasiado grande. Como las ganancias obtenidas eran similares a los otros casos, podemos concluir que este conjunto es el peor de los tres.

Por lo tanto, a partir de ahora vamos a escoger el segundo conjunto de operadores como función generadora de estados sucesores, y la primera de las funciones heurísticas.

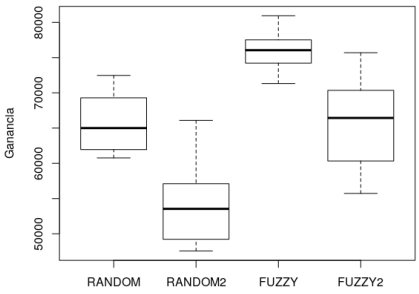
3.3 Experimento 2: Influencia de la solución inicial

Habiendo obtenido en la sección anterior una pareja de función heurística y función de creadora de sucesores, ahora escogeremos cuál es la mejor manera de iniciar la búsqueda. Los métodos que compararemos entre sí son los nombrados anteriormente en la sección **Elección y generación del estado inicial**.
En primer lugar, con las ejecuciones iniciales, observamos que la inicialización *ONE* tiene una ejecución de alrededor de 4 minutos (223 segundos) y, como veremos a continuación con el experimento, el resto de ejecuciones son sustancialmente mayor que cualquier otra opción. Por ello, la descartaremos antes de realizar los experimentos.
Debido a las grandes diferencias entre la mejor inicialización y el resto, no vimos necesario experimentar más allá del escenario inicial.

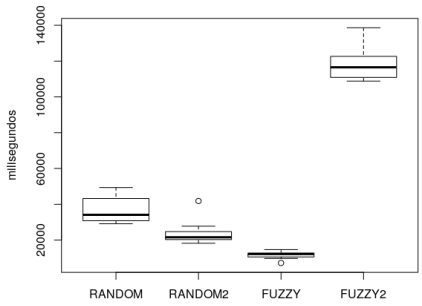
Observación	Existen distintas soluciones iniciales y algunas tienen mejor comportamiento que otras
Planteamiento	Queremos ver cómo cambian el tiempo de ejecución y la ganancia dependiendo de la solución de central.
Hipótesis	• Todos los estados iniciales son iguales (H0) o hay unos con más ganancia que otros. • En caso de empate entre los que llevan a soluciones con más ganancia, también son iguales en tiempo de ejecución (H0) o uno es más rápido que el otro.
Método	• Escogemos 10 semillas aleatorias. • Para cada semilla ejecutamos HC con el escenario inicial. • Después de la ejecución, obtenemos la ganancia y el tiempo de ejecución y hacemos la media para cada grupo de experimentos. • Observamos si qué solución es la más rápida • Hacemos uso del unpaired test t-student con una confianza del 95% para distinguir qué la solución provoca más ganancia

Los boxplots de las medias de ganancia y de tiempo de ejecución son los siguientes:

Observamos que claramente la inicialización *FUZZY* es la más rápida, por lo que, si fuese la que maximizase la ganancia, sería la mejor solución. Para ello, se realizarán los test para saber si hay diferencias significativas de ganancias entre *FUZZY* y el resto de inicializaciones. Trabajaremos con los datos obtenidos después de realizar la ejecución descrita anteriormente, que son los siguientes:



(a) Ganancia



(b) Tiempo de ejecución, en ms.

RANDOM	*RANDOM2*	*FUZZY*	*FUZZY2*
69760.67	66094.97	74224.57	72711.57
60893.22	57102.22	73606.82	69817.57
64313.53	47562.04	76948.14	60331.84
60766.74	52868.345	71332.95	68029.94
69292.7	54089.03	77320.08	64848.73
72477.04	63539	80934.55	75692.46
65828.1	52993.25	77521.8	60901.25
65684.08	54209.13	75155.53	57405.78
61952.42	49224.82	74675.62	55724.82
63407.98	48606.223	78327.68	70354.03

Realizando los tests, obtenemos lo siguiente

	t	df	p-value	diferencia de las medias
RANDOM y *FUZZY*	10.9974	18	<0.0001	-10567.1260
RANDOM2 y *FUZZY*	10.3018	18	<0.0001	-21375.87120
FUZZY2 y *FUZZY*	4.8581	18	0.009	-10423.31120

Ya que los p-values son tan pequeños, podemos decir con una confianza mayor a 99% que la inicialización de *FUZZY* es diferente al resto y es la mejor solución.
De este modo concluimos que esta es nuestra mejor inicialización, lo cual también es lógico pues una inicialización *FUZZY*, asigna con gran probabilidad a un cliente con una central con pocas pérdidas, lo cual estará más cerca de un óptimo local que una asignación aleatoria.

3.4 Experimento 3: Parámetros del Simulated Annealing

Una tarea a realizar es encontrar parámetro adecuados para ejecutar el Simulated Annealing, que son:

- El número total de iteraciones (**steps**).
- Iteraciones por cada cambio de temperatura (**stiter**), que ha de ser un divisor del número de iteraciones.
- El parámetro **k** de la función de aceptación de estados.

- El parámetro **lambda** de la función de aceptación de estados.

Tenemos como objetivo optimizar el SA, partiendo del escenario tipo y con los parámetros de los experimentos anteriores, esto es, la inicialización *FUZZY*, la función de sucesores 2 y la función heurística 1. Observamos que puede haber parámetro que consigan un coste menor, ya que k y lambda influyen en la función de energía y la probabilidad de aceptar un estado peor, que seguramente dependen de la heurística, steps y stiter (que determinan las iteraciones y su distribución). Se intuye que a mayor número de steps, habrá una mejor tendencia. Nuestra hipótesis es si los parámetros son indiferentes (H0) o influyen decisivamente. Dado la complejidad resultante de tener cuatro parámetro libres, procederemos del siguiente modo:

- Fijaremos unos valores para *steps* y *siter*, teniendo en cuenta que el primero ha de ser suficientemente grande para que haya convergencia.
  Para ello, fijaremos $siter = 100$, y pondremos unos valores por defecto: $k = 20$, $lambda = 0.005$, para comprobar si se llega a una solución válida.
- Probaremos una serie de valores distintos, que sean extremos, para k y $lambda$, calcularemos la media y para tener una idea dónde minimiza la función heurísitica, es decir, que maximiza la ganancia en una solución válida.
- Si es necesario, volveremos a repetir acotando valores.
- Observaremos, fijando el resto de parámetros, si hay alguna diferencia variando el número de *siter*.

A continucación, solo se incluirán las tablas resultantes del experimento. Los datos mostrados son la media de tres ejecuciones con semillas aleatorias.
Paso a paso, los resultados fueron los siguientes.
Se ejecutó SA con distintos rangos de steps y observamos que se necesitaban al menos 40000 para garantizar algo de convergencia. Decidimos escoger 70000, ya que para estar seguros de la convergencia, y no probamos más allá porque el tiempo de ejecución sería demasiado elevado.

steps	gananciaSA	gananciaHC
5000	no válida	78201.82
10000	67869.5	78201.82
20000	61189.72	78201.82
40000	71991.2	78201.82
70000	72050.3	78201.82

Después, procedemos a ejecutar la versión con valores extremos de $k = 1, 5, 25, 125$ y $lambda = 1, 0.01, 0.0001$.

lambda	k	ganancia
1	1	no válida
1	5	no válida
1	25	no válida
1	125	no válida
0.01	1	71041.64
0.01	5	71358.2
0.01	25	70647.4
0.01	125	71565.1
0.0001	1	70873.6
0.0001	5	69866.2
0.0001	25	69143.3
0.0001	125	68827.6

Observamos que la desviación típica de los valores con $lambda = 0.01$ es 390.4255. Por ello, concluimos que k no tiene una importancia significativa en el SA. Por ello, escogeremos $k = 25$.

A continuación, experimentando con $siter = 50, 100, 200$ con $lambda = 0.01$ y $k = 0.05$.

siter	ganancia
50	72122
100	71786.55
200	70671.32

Comparando con $siter = 50$ y $siter = 100$, y usando un test paired t de student, concluimos que hay una diferencia significativa, por lo que fijamos $siter = 50$.

t	df	p-value	diferencia de las medias
5.4578	10	0.0055	-732.67

Incrementando más allá steps, solo consiguiríamos aumentar demasiado el tiempo de ejecución. Así que, finalmente, fijamos los valores $steps = 70000, siter = 50, k = 25, lamba = 0.01$
Refutamos H0, aunque los más determinantes son k y steps y no hemos conseguido mejorar SA de tal manera que mejore los resultados de HC.

3.5 Experimento 4: HC para valores crecientes

En este experimento vamos a observar cómo se ve afectado el comportamiento del algoritmo de Hill Climbing cuando se aumenta la complejidad del problema. En particular, vamos partir del escenario planteado en la sección *3.1.* En todo el experimento, el algoritmo de búsqueda local va a utilizar la función de generación de sucesores y estrategia de inicialización que mejores resultados dieron en los experimentos anteriores.

Podemos dividir este experimento en dos partes. En primer lugar, manteniendo un número fijo de 1000 clientes, vamos a realizar ejecuciones del algoritmo de búsqueda local con valores crecientes de las centrales manteniendo las proporciones; empezando por 40 y aumentando su valor en intervalos de 40 también. En segundo lugar, manteniendo un valor de 40 centrales, haremos exactamente lo mismo para valores crecientes de clientes; empezando por 500 y aumentando su valor en intervalos de 500 manteniendo las proporciones. El experimento terminará cuando se aprecie una tendencia a medida que el valor aumenta.

3.5.1 Valores crecientes de las centrales

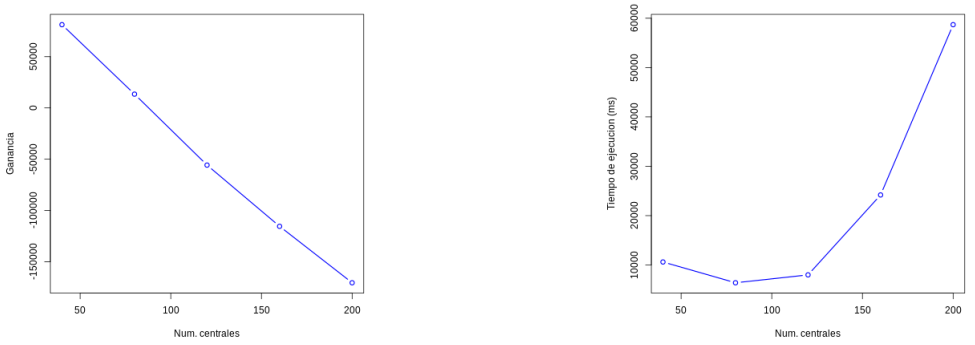
Para valores crecientes en las centrales, podemos plantear el experimento de la siguiente forma:

Observación	Si se aumenta el número de centrales para un número de clientes fijo, HC generará muchos más sucesores y por lo tanto el tiempo de ejecución se verá afectado. La ganancia también puede resultar afectada ya que tendremos más coste.
Planteamiento	Queremos ver cómo cambian el valor de la ganancia y el tiempo de ejecución a medida que aumentamos el número de centrales para un valor fijo de clientes.
Hipótesis	• El tiempo de ejecución aumenta con el número de clientes (H0) o existen comportamientos distintos. • La ganancia se ve afectada negativamente debido al coste de las nuevas centrales (H0) o su comportamiento es distinto.
Método	• Escogemos 10 semillas aleatorias. • Para cada semilla ejecutamos HC empezando por 40 centrales, y aumentando en intervalos de 40 hasta que se vea una tendencia. • Después de la ejecución, obtenemos la ganancia y el tiempo de ejecución y hacemos la media para cada grupo de experimentos.

Después de realizar el procedimiento descrito anteriormente, los resultados que se obtienen son los siguientes:

num. centrales	ganancia	tiempo
40	81179.193	10604.33
80	13425.61	6391.33
120	-55787.82	7990.0
160	-115508.84	24187.33
200	-170561.97	58692.33

En la figura 5, están representados los valores que aparecen en la tabla para una mejor interpretación de los resultados del experimento.



(a) Ganancia

(b) Tiempo de ejecución, en ms.

Figure 5: Ganancia y tiempo de ejecución en función del número de centrales.

Por lo que respecta a la ganancia, se confirma la hipótesis que se ha introducido. Como podemos observar la ganancia se ve afectada negativamente a medida que aumentamos el número de centrales de forma aproximadamente lineal. Este comportamiento es debido a que cada vez que se introduce una nueva central en el problema, estamos añadiendo coste y por lo tanto tendremos menos ganancia

en el resultado final. Como el valor de clientes es constante, a medida que introducimos más centrales tendremos una ganancia menor.

Por otra parte, también se confirma la hipótesis sobre el aumento del tiempo de ejecución con el número de centrales. El resultado obtenido es razonable ya que cuantas más centrales se introduzcan en el problema, mayor será el espacio a tener en cuenta. Podemos ver que si seguimos aumentando el total de centrales, la tendencia que sigue el tiempo de ejecución es exponencial creciente.

3.5.2 Valores crecientes de los clientes

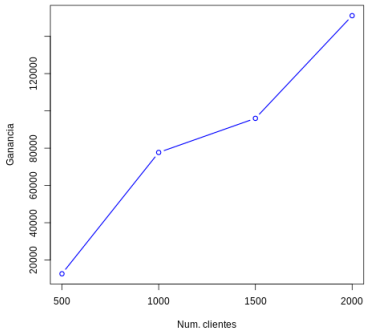
Para valores crecientes de clientes, podemos plantear el experimento de la siguiente forma:

Observación	Si se aumenta el número de clientes, el algoritmo HC generará más sucesores y por lo tanto tendremos un mayor tiempo de ejecución en promedio. El valor final de la ganancia aumentará ya que tendremos unos costes similares, pero encontrar una solución correcta (sin sobreproducción en la central) va a ser más complicado.
Planteamiento	Queremos ver cómo cambian el valor de la ganancia y el tiempo de ejecución a medida que aumentamos el número de clientes para un valor fijo de centrales.
Hipótesis	• El tiempo de ejecución aumenta con el número de centrales (H0) o existen comportamientos distintos. • La ganancia se ve afectada positivamente debido a la introducción de los nuevos clientes (H0) o su comportamiento es distinto. • Para valores grandes de clientes, tendremos sobreproducción en las centrales (H0) o las soluciones seguirán siendo correctas.
Método	• Escogemos 10 semillas aleatorias. • Para cada semilla ejecutamos HC empezando por 500 clientes aumentando en intervalos de 500 hasta que se vea una tendencia. • Después de la ejecución, obtenemos la ganancia y el tiempo de ejecución y hacemos la media para cada grupo de experimentos. También miramos si la solución que obtenemos es correcta.

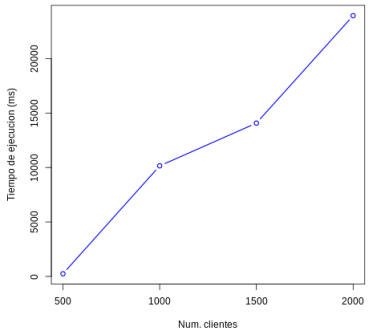
Una vez se ha realitzado el experimento descrito, los valores que se obtienen son los siguientes:

num. clientes	ganancia	tiempo	correcto.
500	12617.097	244.3	sí
1000	77702.772	10152.8	sí
1500	96005.64	14077.2	no
2000	151080.39	23940.67	no

Podemos observar que la hipótesis de tener sobreproducción en las centrales para valores mayores de clientes se cumple. Puesto que el algoritmo encuentra mínimos locales en estados que no describen una solución correcta. Es importante comentar que cuando el problema encuentra un mínimo local en un estado que no describe una solución correcta, el promedio del número de nodos explorados en el problema es menor que en el caso de encontrar una solución correcta. Por ejemplo, en una de las ejecuciones para 1500 clientes, el problema encontró una solución correcta en 80000 milisegundos; un valor muy superior al que aparece en la tabla debido a que la exploración es mucho mayor.



(a) Ganancia



(b) Tiempo de ejecución, en ms.

Figure 6: Ganancia y tiempo de ejecución en función del número de clientes.

En la figura 6, podemos ver la representación de los valores que aparecen en la tabla. Es importante comentar que las hipotesis del aumento en el tiempo de ejecución y el aumento en la ganancia son correctas.

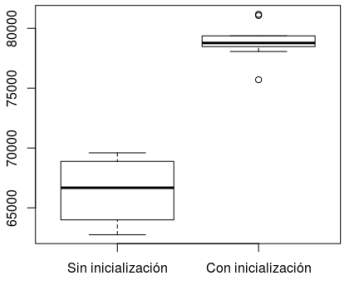
Para el primer caso tiene sentido esperar este resultado ya que cuando se introducen más clientes en el problema el espacio que el algoritmo de busqueda tiene que recorrer es mucho mayor. Para valores en los cuales la solución es incorrecta, hemos comentado que el recorrido de la búsqueda local es mas pequeño, pero aún así podemos observar que el tiempo sigue siendo mayor. En el caso de la ganancia, a medida que tengamos más clientes, podremos generar más ingresos y por lo tanto la producción sera mayor. Sin embargo, si aumentamos el número de clientes llegará un punto en el cual será más difícil encontrar un mínimo local que represente un estado sin sobreproducción en las centrales.

3.6 Experimento 5: Influencia del estado inicial

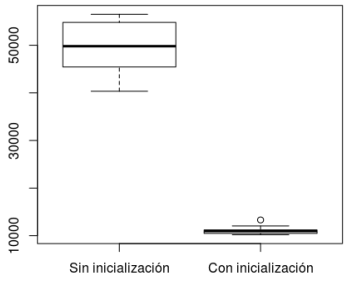
Tenemos ahora una función heurística, una función creador de estados sucesores y un estado inicial que nos dan el máximo rendimiento. Vamos a comprobar ahora como se comportan la función heurística y la función de sucesores si empezamos con un estado inicial no válido. En este experimento empezaremos con un estado inicial donde ningún cliente tiene centra asignada.
Primero tenemos que cambiar la función heurística para que penalice el hecho de que un cliente garantizado no tenga una central asignada. Para esto, hemos añadido un factor de penalización en la heurística que crece cuadráticamente en función del número de clientes garantizados no servidos. Además cualquier solución no válida tiene un valor más alto que una solución válida, ya que el factor de penalización es más alto que el valor óptimo al que se puede llegar en valor absoluto. Esto fuerza el algoritmo a reducir al mínimo el número de clientes garantizados no asignados, esto se consigue asignando una central a cada cliente y, por lo tanto, llegando a un estado válido. Una vez conseguido un estado válido ya no se puede llegar a un estado no válido ya que todos tienen coste mayor.
Sin cambiar la función de estados sucesores obtenemos resultados con un tiempo de ejecución muy alto, esto nos obliga a adaptar la función de sucesores a esta nueva estrategia. Lo que se ha añadido es un nuevo operador que asigna a una central un número muy alto de clientes de forma aleatoria. Este numero, después de probar varios candidatos, es el número de clientes entre el número de centrales. Ya que es el que corresponde a cada central en media.
Una vez hecho esto el rendimiento ha mejorado mucho, aun así se observa que el rendimiento con un estado inicial válido es mucho mejor. Esto se debe a que al empezar con un estado no válido el primer estado válido al que se llega es muy importante, ya que nunca se vuelve a un estado no válido después,

y no es tan bueno como los estados iniciales que se han probado.

	Sin estado inicial		Con estado inicial	
caso	ganancia	tiempo	ganancia	tiempo
1	68879.50	40341	79221.26	10993
2	69593.00	51018	75706.21	13317
3	65298.00	54888	78470.16	10491
4	63038.55	48872	81070.36	11004
5	62749.00	40364	81175.96	10543
6	69009.75	50424	78816.46	11174
7	63995.71	49192	78064.61	10207
8	66008.70	45439	78568.41	12057
9	67347.05	56505	78738.36	11202
10	67937.85	54800	79373.06	10269



(a) Gananancia.

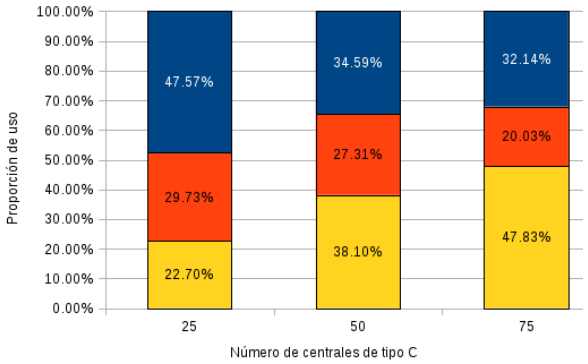


(b) Tiempo de ejecución, en ms.

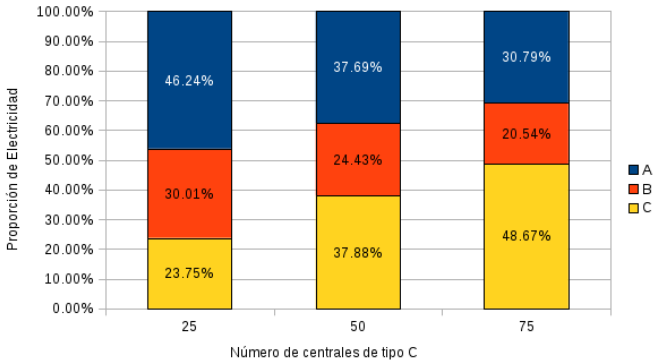
3.7 Experimento 6: Aumento de centrales de tipo C

Consideremos el caso donde aumenta el número de centrales de tipo C, es decir en nuestra cuadrícula tendremos más centrales pequeñas para satisfacer a los clientes con menos pérdidas. Lo que nos dice la intuición es que a medida que aumentamos el número de este tipo, se vuelve más probable que para cada cliente exista una central de tipo C cercana, que pueda suministrar electricidad sin muchas pérdidas. Y por tanto aumente el uso de centrales de este tipo y disminuya la de los otros.

Nos fijamos que ambos gráficos parecen estar correlaciondos, y que el uso es proporcional a la producción de las centrales, para verificar este hecho.



(a) Porcentaje de uso de las centrales



(b) Porcentaje de electricidad producida

Observación	El uso de un tipo de central es proporcional a la producción de electricidad de dicho tipo de central
Planteamiento	Queremos saber si provienen de la misma distribución el porcentaje de uso de tipos de central y el portentaje de electricidad producida.
Hipótesis	• Las proporciones del uso de centrales y de la electricidad producida son iguales
Método	• Escogemos 10 semillas aleatorias. • Para cada semilla ejecutamos HC con 5 centrales de tipo A, 10 de tipo B y 25, 50, 75 centrales de tipo C • Después de cada ejecución guardamos la electricidad producida por tipo y por central encendida. • Después de cada ejecución guardamos el número de clientes asignados a cada tipo de central. • Realizamos un paired t - test con una confianza del 95% para verificar si hay diferencias significativas para cada cantidad de centrales de tipo C

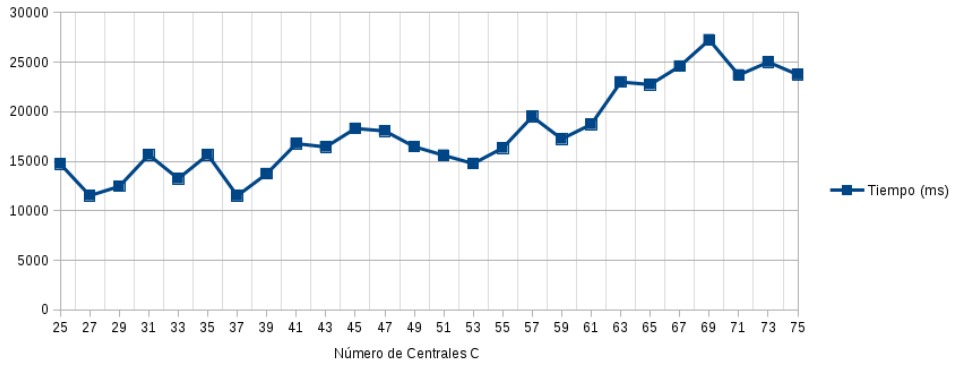
Siguiendo el procedimiento descrito anteriormente obtenemos los siguientes datos

	t	df	p-value	diferencia de las medias
25	1.0824	10	0.3045	0.01105
50	0.02066	10	0.8405	0.00158
75	1.3480	10	0.2074	0.00893

En los tres casos, dado que el p-value es mayor a 0.05 podemos concluir que no hay diferencias significativas. Y por lo tanto podemos suponer que son la misma distribución.

En cambio si nos fijamos en el beneficio obtenido, veremos que el valor decrece rápidamente a medida que añadimos centrales, lo cuál nos lleva a pensar que nuestros operadores trabajan mal en el caso de que exista más oferta que demanda. En concreto, no aprovecha poder dejar centrales vacías para no pagar el coste de generar toda la electricidad de esa central. Dado los operadores que hemos definido, a cada paso el número de clientes asignados solo puede aumentar o disminuir lo mismo, o mantenerse igual. Esto querrá decir que si una central tiene a dos clientes asignados la única operación que podremos hacer sobre esta central es asignarle otro cliente. Además dado que una central con dos clientes no puede sobreproducir, las heurísticas que hemos desarrollado no la penalizarán y por lo tanto nunca resultará en una mejora quitar a una persona de esta central. Por tanto es lógico que los resultados obtenidos a medida que aumentamos el número de centrales disminuiyan.

Dado que nuestro programa no trata este caso de ninguna forma especial, se espera que al aumentar el número de centrales tipo C solo aumente el coste temporal debido al aumento del factor de ramificación.



(a) Media del coste temporal en función de el número de centrales de tipo C

Lo cual parece concordar con nuestra conclusión.

3.8 Comparación de algoritmos

Después de la práctica realizado, nos hayamos en posición de realizar ciertas comparaciones entre nuestros dos algoritmos principales.
En primer lugar, queda bastante claro su utilidad, ya que ambos son capaces de encontrar óptimos locales en un enorme espacio de búsqueda, con mayor o menor acierto en la sucesión de nodos para alcanzarlos desde una solución inicial.
En segundo lugar, la implementación de ambos difiere bastante: mientras que el Hill Climbing solo necesita una función generadora de sucesores y una función heurística, el Simulated Annealing necesita además cuatro parámetros que pueden depender entre sí y de las dos funciones anteriormente dichas. De este modo, buscar estos parámetros se convierte en una tarea relativamente costosa, ya que el espacio de búsqueda es enorme y, de hecho, podríamos definirlo como un problema de búsqueda local.
En tercer lugar, compararemos su ejecución. En nuestra implementación, por posiblemente diversas razones, el Hill Climbing ha resultado ganador respecto a la comparativa de ejecuciones, ya que precisa mucho menos tiempo y da soluciones con ganancias mayores. Esto ha sido un resultado sorpredente, ya que esperábamos que el Simulated Annealing, aún teniendo un tiempo de ejecución mayor en los casos iniciales que el Hill Climbing, sería un algoritmo que escalaría mucho mejor el problema y saldría ganador para entradas mayores que el escenario inicial; sin embargo, nos hemos topado con que en los casos inciales comparábamos ejecuciones de en torno a 10 segundos con ejecuciones de casi 1 hora. Además, se esperaba que el Simulated Annealing tuviese más variaciones al cambiar sus parámetros.

4 Conclusiones

Tras la ejecución de la parte de implementación y la de experimentación, hemos obtenido una visión más general del problema de la busqueda local y como interviene sus elementos en ella. En particular hemos podido observar como debido a la localidad del problema, es esencial tener una inicialización cercana a una solución para poder llegar a la solución en un tiempo razonable. Además el camino a esta solución depende mucho de los operadores y hemos podido comprovar empíricamente algunas hipótesis sobre el comportamiento del problema. En particular hemos podido ver los problemas que

causaba tener operadores tan simples que no aprovechasen el exceso de oferta, o la rigidez de una función sucesora que representaba el espacio de estados como una función cresta. Antes de todo, esta práctica nos ha mostrado la importancia del tiempo. Esto ha sido un limitante, tanto en la primera implementación del estado como en la experimentación de nuestro Simulated Annealing (que ha tenido un tiempo de ejecución muy elevado).
Esto nos ha permitido apreciar, por un lado, la importancia de la inteligencia artificial como herramienta de resolución de problemas cotidianos de logística y, por otro, como aumenta la dificultad de resolver problemas que son NP, incluso con entradas relativamente pequeñas.
Cabe decir que esperabamos otra ejecución del Simulated Annealing, que necesitaba mucho más tiempo para hallar una solución peor que el Hill Climbing. Esto posible será debido a una mala elección de los operadores, que como hemos mencionado en el trabajo, eran demasiado rigidos y permitían cambios demasiados pequeños, lo cual crea muchos máximos locales y dificulta la busqueda de mejores extremos locales.
Por otra parte, aunque esta es una práctica relativamente pequeña, nos ha permitido apreciar la necesidad de una buena organización y comunicación grupal; tanto por la necesidad de realizar los experimentos para asegurarnos que se podía pasar al siguiente paso, como que los cambios en, por ejemplo, la función heurística que tiene efectos en toda la práctica, hace falta tener claro qué hace qué y cuándo.

5 Trabajo de innovación

Este apartado recoge toda la información referente a la competencia transversal del trabajo de innovación. A continuación vamos a hablar sobre el tema del trabajo, y a establecer una primera planificación de su realización.

5.1 Descripción del tema escogido

El tema que hemos escogido para nuestro trabajo de innovación es el análisis del servicio de detección de *spam* que Google utiliza para la detección de correo no deseado en las cuentas de los usuarios de gmail.

5.2 Reparto del trabajo

En esta lista vamos a describir "groso modo" el desarrollo de nuestro trabajo.

- Presentación y motivación del servicio de detección de correo no deseado.
- Descripción detallada del algoritmo de detección de correo no deseado. Si no se encuentra una fuente oficial, describiremos uno parecido.
- Búsqueda y prueba de ejecución de un algoritmo de detección similar. Este último apartado aún se tiene que discutir.
- Efectos, ventajas e inconvenientes que este servicio ha tenido a lo largo de los años.

En cuanto al reparto de tareas, cada miembro del grupo ha participado activamente en la organización y desarrollo del trabajo. Por lo tanto, no vamos a citar exactamente quién se ha encargado de cada aspecto ya que no lo podemos saber con certeza.

5.3 Lista de referencias

Este es un resumen de las fuentes que se han visitado, hemos explorado bastante más pero aquí solamente citaremos las más importantes.

- Inspiración: uno de nosotros es un frecuente usuario de Quora, la idea surgió de aquí `https://www.quora.com/How-does-Gmail-spam-detection-work-1`, el *4-10-2018*.
- Recientes actuaciones de Google respecto al spam `https://www.blog.google/products/search/search-spam-report-2017/`
- Presentación y descripción general del algoritmo de detección `https://www.techlila.com/gmail-spam-filter/`, el *4-10-2018*.
- Descripción detallada del algoritmo de spam `http://robotics.stanford.edu/users/sahami/papers-dir/spam.pdf`, el *12-10-2018*.

5.4 Dificultades en la búsqueda

Por el momento no hemos sido capaces de encontrar una declaración oficial del funcionamiento exacto del algoritmo de detección de correo no deseado que google utiliza. En caso de no hacerlo, buscaremos un artículo científico que describa un algoritmo de detección de spam similar.