Servidores y Usuarios

Practica 1 de Inteligencia Artificial
11 Abril 2016

Felix Axel Gimeno
Josep Llistosella
Ali Muhammad
Guillermo Serrahima

Índice

Introducción

Identificación del problema

Se nos presenta un sistema habitual de acceso a grandes sistemas de ficheros a través de servidores. En esta situación, tenemos una serie de servidores con un número de archivos repetido en un subconjunto de ellos, y una serie de peticiones por parte de clientes para acceder a ficheros. El problema consiste en determinar qué servidor va a resolver cada una de las peticiones.

Los servidores están distribuidos geográficamente, por lo cual, los tiempos de transmisión de un fichero entre un servidor y un cliente concretos varían considerablemente.

Para que ningún cliente sea asignado una conexión lenta, se deben asignar peticiones a servidores de forma que el tiempo de transmisión sea mínimo; pero no es suficiente con eso, o podría provocar saturaciones en aquellos servidores al alcance de la mayoría y con copias de los ficheros más pedidos: hace falta también equilibrar las transmisiones entre los servidores. Dicho de otra manera, hay que asignar peticiones a servidores de forma que el tiempo total de un servidor de transmisión esté cerca de la media de tiempos de transmisión de los servidores para las peticiones asignadas.

Descripción del estado

Un solución consiste en la asignación de servidores a peticiones, tal que cada petición tenga al menos y como mucho un servidor que resuelve esa petición. Así que podríamos guardar para cada petición, por cual servidor es resuelta, o a la inversa, para cada servidor guardar las peticiones que resuelve. Con cualquiera de estas representaciones recubriríamos el espacio de soluciones, cuyo tamaño es número de replicaciones elevado a número de peticiones (cada petición puede ir a número de replicaciones servidores distintos). Nosotros nos hemos decantado por, usando memoria dinámica, para cada servidor guardar sus peticiones, implementado en java como ArrayList<Arraylist<int>>, cuyo coste en memoria sería O(Número de peticiones + Número de Servidores). Además de guardar memorizamos el cómputo del tiempo de transmisión de cada servidor, resultando en O(Número de Servidores) extra memoria.

Descripción del conjunto de operadores

Notación: |S| representa el número de servidores, |R| representa el número de peticiones, |Rep| el número de replicaciones

Nuestro estado tiene dos operadores:

- Operador: Mover (Servidor 1, Servidor 2, Petición)

  Dada una petición del servidor 1, se mueve esa petición al servidor 2 (deja de estar en el servidor 1 para estar en el servidor 2)

  Condición de aplicabilidad: se mira que el servidor 1 sea diferente del servidor 2, que el servidor 1 contenga la petición 1 y que el servidor 2 pueda atender la petición (es decir, que el servidor contenga el archivo de la petición)

  Factor de ramificación: |R| * (|Rep| - 1), ya que se puede mover cualquier petición de cualquier servidor a cualquier otro servidor que no sea el mismo que el primero

- Operador: Intercambiar (Servidor 1, Petición 1, Servidor 2, Petición 2)

  Dada una petición del servidor 1 y una petición del servidor 2 intercambia estas peticiones (la petición 1 del servidor 1 ahora está en el servidor 2 y la petición 2 del servidor 2 ahora está en el servidor 1)

  Condición de aplicabilidad: se mira que el servidor 1 sea diferente del servidor 2, que el servidor 1 contenga la petición 1, que el servidor 2 contenga la petición 2, que el servidor 2 pueda atender la petición 1 y que el servidor 1 pueda atender la petición 2.

  Factor de ramificación: |R|*|R| * (|Rep| - 1), ya que se puede intercambiar cualquier par de peticiones de cualquier servidor a cualquier otro servidor que no sea el mismo que el primero y no hay dos peticiones iguales.

Hemos decidido usar dos conjuntos de operadores para ver cuales de ellos irán mejor para los experimentos. Los conjuntos son los siguientes:

- Conjunto 1 (Moves): el único operador que se usa es el de mover peticiones. Es un conjunto correcto ya que el generador de la solución inicial pone todas las peticiones en los servidores, y con solo moverlas entre servidores se llegan a otras soluciones, por lo tanto recorre todo el espacio de soluciones.
- Conjunto 2 (Moves y Swaps): consiste de los dos operadores, mover e intercambiar peticiones. Como intercambiar equivale a hacer dos

movimientos de peticiones, por las mismas razones que antes el conjunto de operadores es correcto porque recorre todo el espacio de soluciones.

Descripción del generador inicial

Disponemos de dos generadores de estados iniciales:

- Generador aleatorio (random): La idea de este generador es que usando la aleatoriedad, coloque todas las peticiones en los servidores. A partir del número de servidores, el número de usuarios, el número máximo de peticiones por usuario, el número de repeticiones de los archivos (cuántas veces están en los servidores) y una semilla, genera un estado en el que las peticiones se han asignado a uno de los servidores que puede atenderla de forma aleatoria. Su coste es de O(|R| * |S|) aunque su coste esperado es de O(|R|)
- Generador iterativo equilibrado (iterativo): Este generador es similar al aleatorio pero intenta equilibrar las peticiones entre todos los servidores. A partir del número de servidores, el número de usuarios, el número máximo de peticiones por usuario, el número de repeticiones de los archivos (cuántas veces están en los servidores) y una semilla, genera un estado en el que las peticiones se asignan a un servidor que pueda atenderlas señalado por un iterador, que aumentará su posición si el servidor señalado no puede atender esa petición. De esta forma se equilibra mucho más las peticiones entre todos los servidores. Su coste es de O(|R| * |S|).

Descripción de las funciones heurísticas

Para el problema hemos usados dos heurísticas (ambas para el algoritmo Hill Climbing). La primera heurística optimiza solamente el primer criterio del enunciado. Y la segunda heurística optimiza a la vez los dos criterios que nos pide inicialmente el enunciado, es decir, reducir el tiempo de transmisión del servidor que tarda más tiempo en transmitir la información, y optimizar el tiempo total de transmisión, pero con la restricción de que el tiempo total de transmisión de cada servidor ha de ser lo más similar posible.

Ambas heurísticas buscan los estados con menor valor heurístico como solución o camino a solución.

A nivel técnico, lo que hace la ***heurística 1*** es: coger una tira de valores (los tiempos de cada uno de los servidores), para los cuales coge 2 consecutivos y los compara, y se queda con el más grande; y lo compara con el siguiente, y se queda con el más grande; y así hasta llegar al final del los valores de los tiempos. De esta

forma se consigue optimizar el criterio 1 del enunciado (reducir el tiempo del servidor que tarda más en transmitir). La optimización del criterio 1 consiste en una minimización del tiempo del servidor que tarda más en transmitir, aplicando la función descrita.

Lo que hace la ***heurística 2*** es: optimizar los dos criterios que nos pide el enunciado a la vez. Para conseguir esto, parte de la idea de optimizar el criterio 1. Concretamente asignamos un valor inicial igual al tiempo del servidor con tiempo máximo (criterio 1). A este valor base se le suma a la "distancia" que tiene cada uno de los servidores de un estado respecto de la media de tiempos de ese estado. Dicha distancia se considera positiva tanto hacia valores superiores a la media como a valores inferiores. De esta forma, el heurístico busca estados que aproximan sus servidores a la media, mientras minimiza el tiempo máximo.

Finalmente, implementando funciones/operaciones matemáticas que realizan los cálculos descritos, se consigue hacer las dos heurísticas para hacer los experimentos que nos pide el problema.

Experimentos del 1 al 8

Experimento 1:

Determinar qué conjunto de operadores da mejores resultados para una función heurística que optimice el primer criterio (3.3) con un escenario en el que el número de usuarios que piden ficheros es 200, el número máximo de peticiones por usuario es 5, el número de servidores es 50 y el número mínimo de replicaciones es 5. 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.

Conjunto de operadores 1 (a partir de ahora “moves”):

-Mover petición de un servidor a otro.

Conjunto de operadores 2 (a partir de ahora “moves_swaps”):

-Mover petición de un servidor a otro.

-Intercambiar dos peticiones entre dos servidores.

Observación	Dependiendo del heurístico puede haber un conjunto de operadores que den una mejor solución, en este caso el heurístico está fijado.
Planteamiento	Escogeremos diferentes conjuntos de operadores y observaremos sus soluciones.
Hipótesis	Los conjuntos de operadores dan los mismos resultados (H0) o, hay conjuntos de operadores que dan mejores resultados que otros.
Método	● Elegiremos 10 semillas aleatorias, una para cada réplica. ● Ejecutaremos 2 réplicas por semilla, una para cada conjunto de operadores. En total, haremos 20 ejecuciones. ● El número de usuarios es 200, el número máximo de peticiones por usuario es de 5, el número de servidores es 50 y el número mínimo de replicaciones es de 5. ● Usaremos el algoritmo Hill Climbing. ● Usaremos la estrategia de inicialización aleatoria. ● El cálculo del heurístico viene determinado por el enunciado.

Partimos de la hipótesis de que los conjuntos de operadores no afectan al resultado del algoritmo. Para refutar esta hipótesis, ejecutaremos el algoritmo de Hill Climbing varias veces con los dos conjuntos de operadores bajo las mismas condiciones en cada réplica. El resultado del experimento se recoge en la siguiente tabla, en la que se destacan el número de pasos hasta llegar a la solución, el valor de la solución (en este caso dado por el heurístico de minimizar el tiempo del servidor con más tiempo) y el tiempo que ha tardado el algoritmo en llegar a la solución:

EXP 1	**Pasos**		**Solución (ms)**		**Tiempo (ms)**	
Réplica	Conjunto 1	Conjunto 2	Conjunto 1	Conjunto 2	Conjunto 1	Conjunto 2
0	185	206	39826	44269	100955	150412
1	179	245	41455	51920	105499	168728
2	272	87	38593	43157	166984	66555
3	182	326	42386	36543	133242	240571
4	153	364	59351	28487	104001	285877
5	12	167	63279	36119	10849	157579
6	204	203	37833	34239	156362	176791
7	176	70	35797	42927	134506	71195
8	128	133	39820	37203	98263	128682
9	52	182	49236	37935	38475	171910

Tabla 1.1: Datos del experimento 1.

A continuación los gráficos que muestran las comparaciones entre los dos conjuntos de operadores. A partir de estas comparaciones se determinará qué conjunto de operadores es el más adecuado para el resto de experimentos:

Figura 1.1: Número de pasos para los dos conjuntos de operaciones.

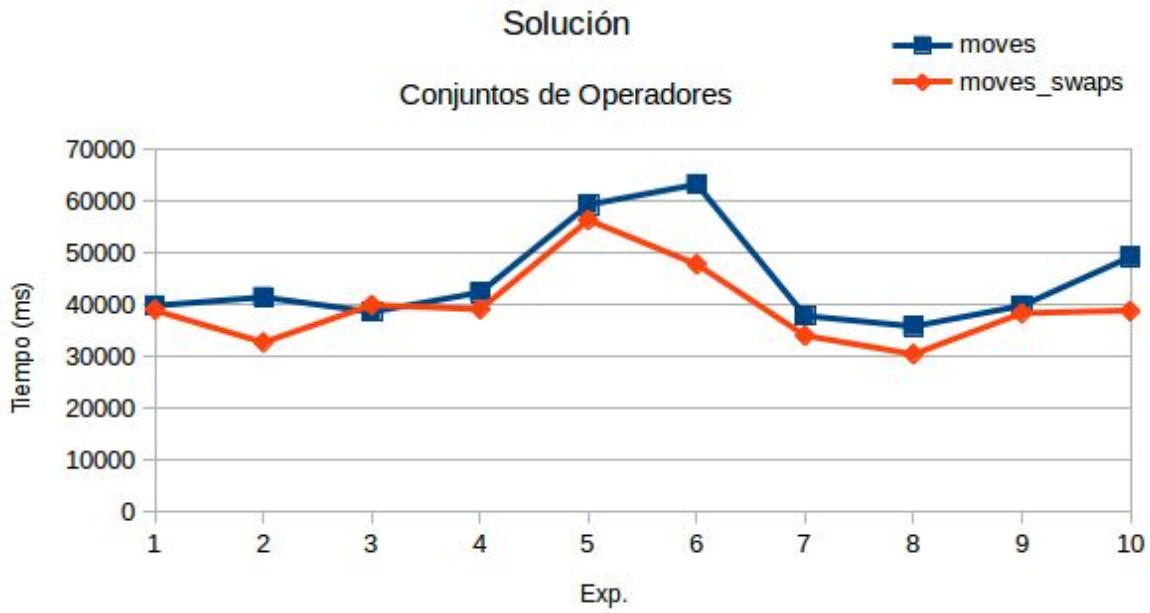


Figura 1.2: Valor de la solución (tiempo máximo de los servidores) de los dos conjuntos de operaciones.

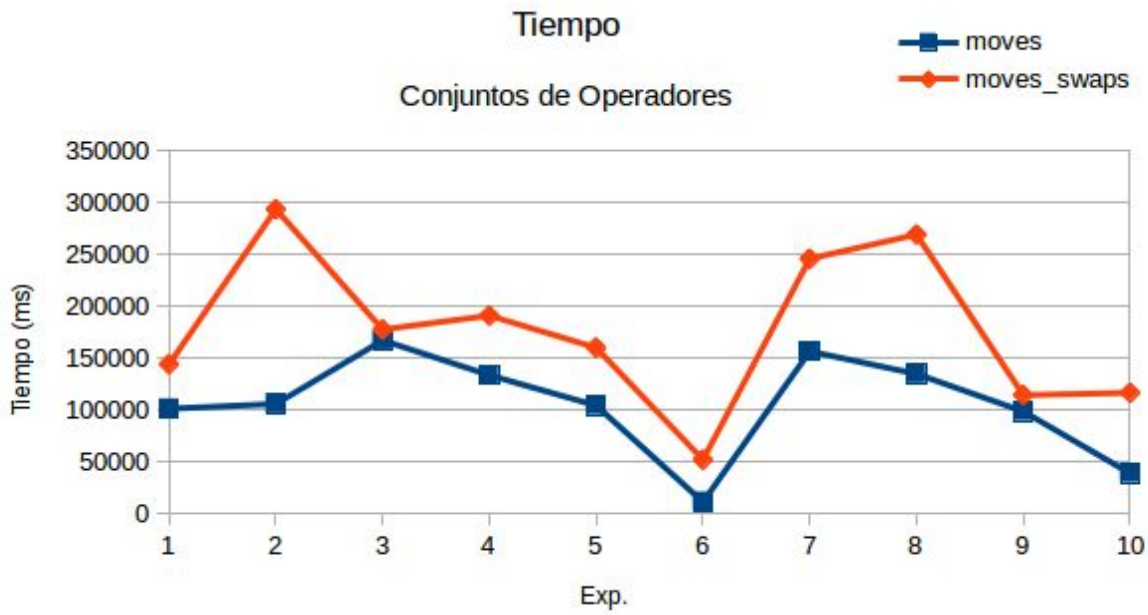
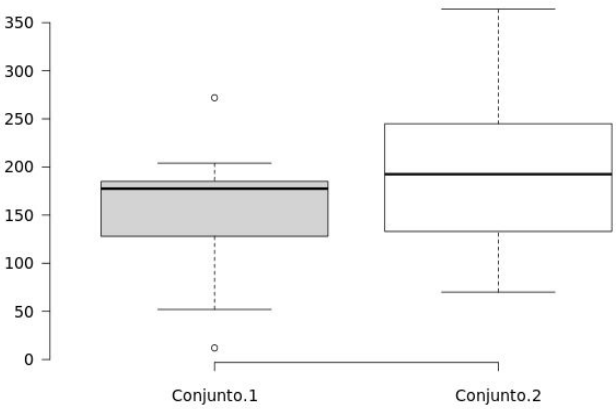


Figura 1.3: Tiempo para encontrar la solución de los dos conjuntos de operaciones.



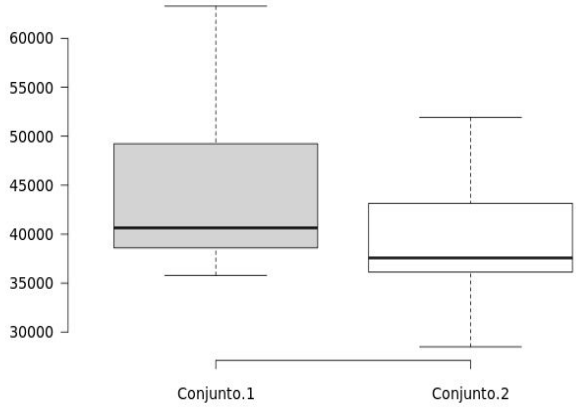
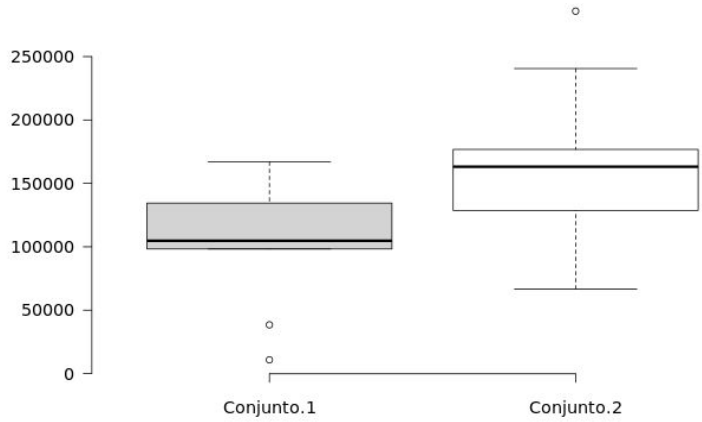


Figura 1.4: Pasos hasta encontrar la solución

Figura 1.5: Valor de la solución encontrada

Figura 1.6: Tiempo para encontrar la solución



Conclusiones del experimento 1:

Con estos resultados, se determina que el segundo conjunto de operadores (moves y swaps) es el más acertado para el resto de experimentos ya que, a pesar de tardar más en encontrar la solución, las soluciones que se encuentran con ese conjunto son en general mejores (el tiempo máximo de los servidores es más pequeño). Esto se debe a que al añadir el operador de intercambiar el factor de ramificación aumenta considerablemente, cosa que provoca que haya más pasos y por consecuencia más tiempo para encontrar la solución, pero también más posibilidades de encontrar un sucessor con mejores características, aparte de que intercambiar dos peticiones es equivalente a hacer dos moves. Por estas razones, se puede descartar la hipótesis inicial y aceptar que el conjunto de operadores move y swap es mejor y será usado para el resto de experimentos.

Experimento 2:

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. Aseguraos de que el número de iteraciones que uséis sea suficiente para que la búsqueda llegue a converger.

Generador 1: Asignación de peticiones a servidores aleatoria (a partir de ahora “random”)

Generador 2: Asignación de peticiones iterativo equilibrado (a partir de ahora “iterativo”)

Observación	El resultado del algoritmo puede ser mejor o peor dependiendo del generador inicial
Planteamiento	Escogeremos diferentes métodos de inicialización y observaremos sus soluciones.
Hipótesis	Todos los métodos de inicialización son iguales (H0) o hay métodos mejores que otros (H1).
Método	● Elegiremos 10 semillas aleatorias, una para cada réplica. ● Ejecutaremos 2 réplicas por semilla, una para cada conjunto de operadores. En total, haremos 20 ejecuciones. ● El número de usuarios es 200, el número máximo de peticiones por usuario es de 5, el número de servidores es 50 y el número mínimo de replicaciones es de 5. ● Usaremos el algoritmo Hill Climbing. ● Usaremos el conjunto de operadores establecidos en el experimento 1. ● El cálculo del heurístico viene determinado por el enunciado.

Ahora que ya se ha determinado que el segundo conjunto de operaciones es mejor, determinaremos qué generador de soluciones iniciales es el más adecuado para el resto de experimentos de una forma similar a lo que se ha hecho en el experimento 1. Hemos supuesto que el generador inicial no influye en el resultado del algoritmo y el experimento servirá para refutar esta hipótesis. A continuación se muestra la tabla con el resultado del experimento usando el algoritmo de Hill Climbing. Se destaca el número de pasos hasta encontrar la solución, el valor de la solución determinado por el heurístico y el tiempo que ha tardado el algoritmo en encontrar la solución.

EXP 1	**Pasos**		**Solución (ms)**		**Tiempo (ms)**	
Réplica	Random	Iterativo	Random	Iterativo	Random	Iterativo
0	153	153	39019	29304	143892	106804
1	306	115	32704	34233	293685	89879
2	187	88	39944	33299	177450	61400
3	199	266	39126	28169	190850	189447
4	173	134	56387	30663	159795	107230
5	50	84	47804	36970	51637	67909
6	256	66	34003	33221	245837	46667
7	274	110	30413	34196	269152	86723
8	116	91	38345	33020	114053	69252
9	120	91	38753	37827	116303	73514

Tabla 2.1: Datos del experimento 2.

Igual que antes, a continuación se muestran una serie de gráficos con el objetivo de comparar los dos generadores de soluciones iniciales:

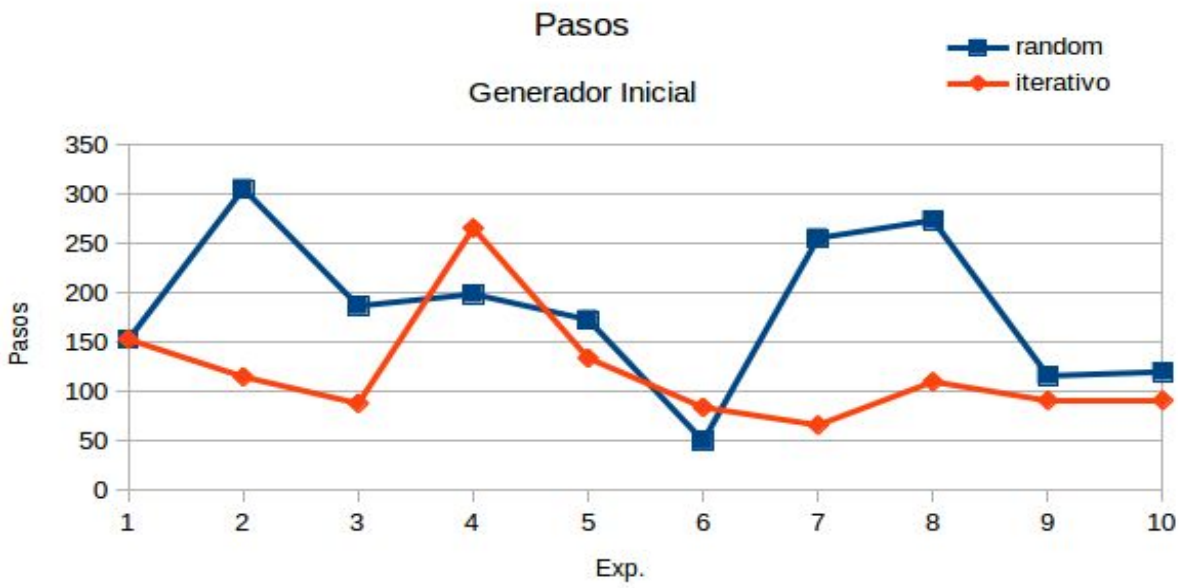


Figura 2.1: Pasos para encontrar la solución de los dos generadores iniciales.

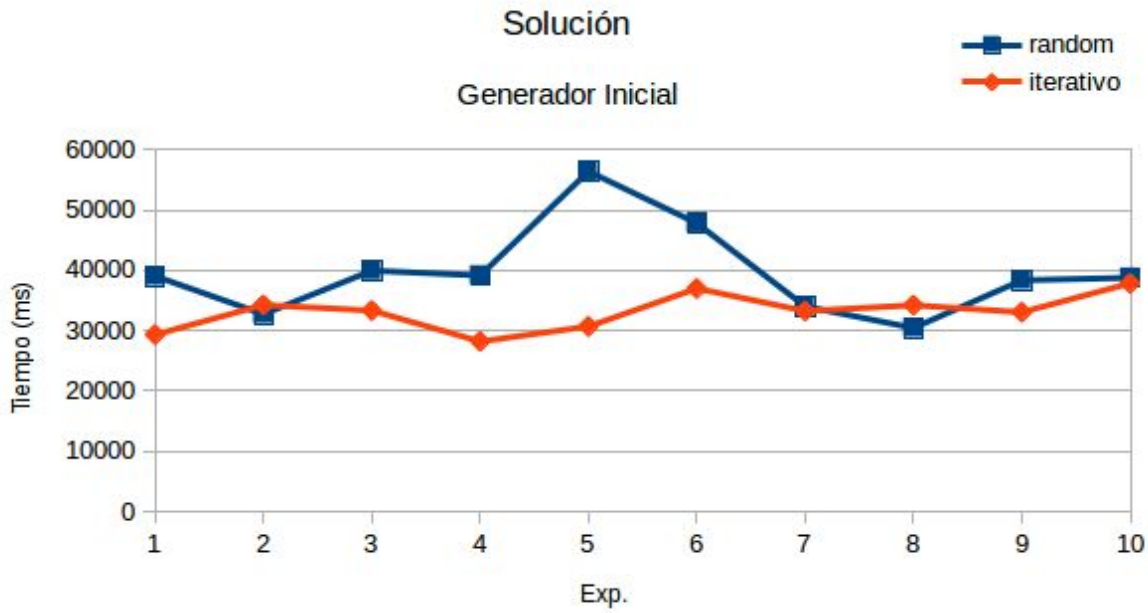


Figura 2.2: Valor de la solución (tiempo máximo de los servidores) de los dos generadores iniciales.

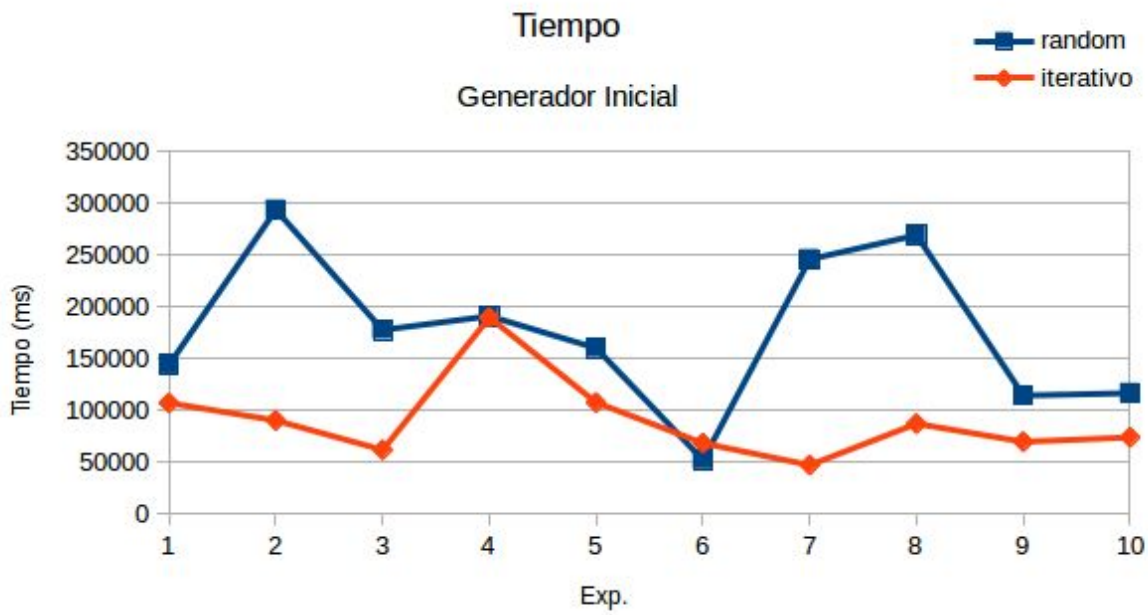


Figura 2.3: Tiempo para encontrar la solución de los generadores iniciales.

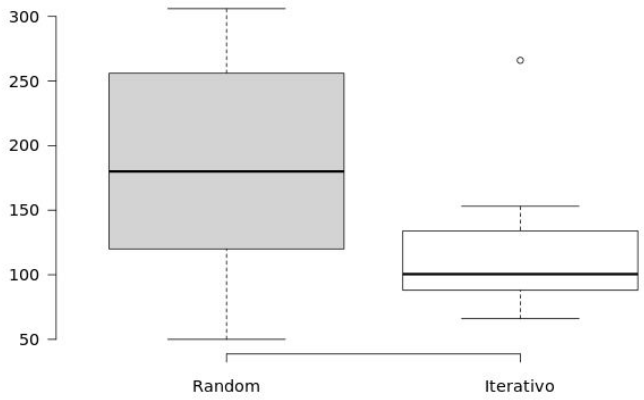


Figura 2.4: Pasos para encontrar la solución encontrada

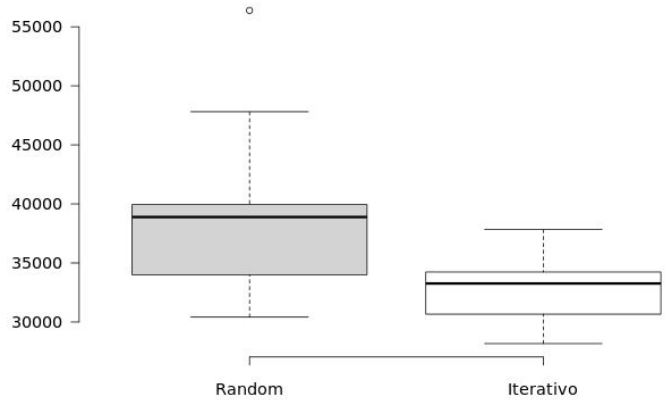


Figura 2.5: Valor de la solución

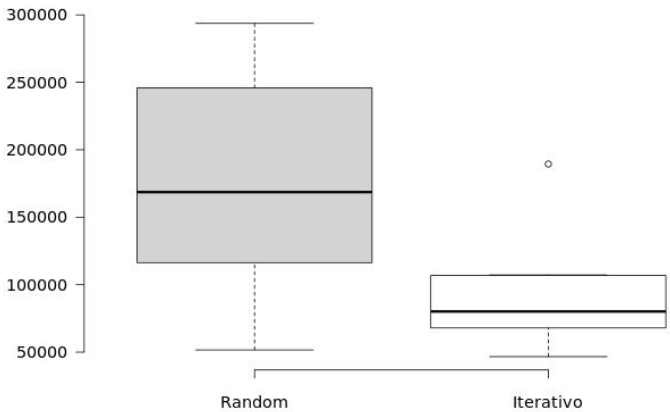


Figura 2.6: Tiempo para encontrar la solución

Conclusiones del experimento 2:

Se puede ver claramente que el generador inicial iterativo es mejor en todos los aspectos al generador inicial aleatorio. El hecho de que genere una solución inicial que equilibra las peticiones entre los servidores sin ser muy costosa hace que mejore los pasos para encontrar la solución y disminuye el tiempo de ejecución. Además encuentra un valor mejor de la solución debido a que la carga entre los servidores está más equilibrada y por lo tanto el tiempo máximo de los servidores acaba siendo más bajo. De esta forma descartamos la hipótesis inicial y aceptamos que la solución inicial iterativa es mejor y la usaremos para el resto de los experimentos

Experimento 3:

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 resultado del algoritmo puede verse afectado por los parámetros del Simulated Annealing
Planteamiento	Escogeremos diferentes valores de los parámetros y compararemos
Hipótesis	H0: Los parámetros no afectan a la calidad de la solución H1: Hay unos parámetros mejores que otros
Método	● Nos limitaremos a un problema específico, el usado en experimentos anteriores ● Usaremos el conjunto de operadores, el generador inicial y el heurístico establecidos antes. ● Usaremos el algoritmo Simulated Annealing ● Ejecutaremos 3 veces cada tupla de parámetros ● El número de pasos es 300000 ● Los pasos por iteración son 100 ● El tiempo para hallar la solución mediante SA debe ser aproximadamente menor o igual que el de mediante HC

Compararemos los resultados entre sí y con el resultado del Hill Climbing, que es 36436 en aproximadamente 140000 ms. Haremos un muestreo de algunos valores de k y lambda para acotar la búsqueda, estos serán k en {1, 5, 25, 125} y lambda en {1, 0.01, 0.0001, 0.000001,0.00000001}, los pasos por iteración serán 100.

Figura 3.1: Gráfica del tiempo y el heurístico en función del número de fila

K	Lambda	Tiempo (ms)	Heuristico
1	1	609	78524
5	1	386	79952
25	1	376	69438
125	1	392	71448
1	0.01	37393	37899
5	0.01	37473	37583
25	0.01	37515	38317
125	0.01	37180	37987
1	0.0001	153005	35823
5	0.0001	151675	35378
25	0.0001	149996	35917
125	0.0001	152891	36760
1	0.000001	151608	35879
1	0.00000001	148707	36365

Tabla 3.1: Datos del experimento 3 parte 1

Conclusiones del experimento 3:

La gráfica son las dos últimas columnas de la tabla anterior, ordenadas según número de fila creciente. Se observan dos tendencias según lambda disminuye: el tiempo para finalizar aumenta, y a cambio se obtiene una solución de mayor calidad.

Específicamente, para lambda igual a 1, es demasiado malo, para lambda igual a 0.01 es peor que 0.0001 pero es bastante más rápido, y para lambda igual a 0.0001 ta se obtendrían resultados muy similares a los obtenidos mediante Hill Climbing, usando lambdas más pequeños aún, no observamos diferencias con lambda igual a 0.0001.

Hemos obtenido una intervalo para las lambdas, 0.0001 o más pequeñas, ahora obtendremos datos para decidir que k escoger, probaremos con lambda igual a 0.0001 y k con valor en {5, 10, 15, 20}

0.0001	5	143545	35757
0.0001	10	151455	36120
0.0001	15	141177	34840
0.0001	20	151643	35748

Tabla 3.2: Datos del experimento 3 parte 2

Por lo que decidimos que k valdrá 15, y ahora, a partir de este valor de k, obtendremos la mejor lambda

0.01	15	34975	37185
0.0001	15	142579	36552
0.000001	15	143374	35585

Tabla 3.3: Datos del experimento 3 parte 3

Con lo que nuestros parámetros serán número de pasos 300000, pasos por iteración 100, k igual a 15, lambda igual a 0.000001, a continuación una gráfica del comportamiento del SA con estos parámetros.

Figura 3.2: gráfica calidad del SA en función del tiempo

Antes de realizar el experimento, pensábamos que las lambda y k serían claras, pero la aleatoriedad inherente al Simulated Annealing causa mucho ruido para efectuar un ajuste más fino que el obtenido.

*Si permitimos que tarde más tiempo que HC, podemos obtener soluciones de mejor calidad, por ejemplo, con 500000 pasos, 1 paso por iteración, lambda igual a 0.0001 y k

igual a 500 se obtiene un tiempo de 246501 ms y un heurístico de 34504, mejor calidad pero tarda más, obteniéndose la siguiente gráfica:

Figura 3.2: gráfica calidad del SA en función del tiempo (más tiempo que HC)

Experimento 4:

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 usuarios que piden ficheros (manteniendo 5 peticiones por usuario)*
- *Número de servidores (manteniendo el número de replicaciones)*

Para 50 servidores, comenzad con 100 usuarios e incrementad el número de 100 en 100 hasta que se vea la tendencia. Para 200 usuarios, comenzad con 50 servidores e incrementad el número de 50 en 50 hasta que se vea la tendencia. Usad el algoritmo de Hill Climbing y la misma función heurística que antes.

Observación	El resultado del algoritmo se verá muy afectado por el número de servidores y usuarios
Planteamiento	Experimentaremos con el número de usuarios y servidores para ver cómo afectan estos parámetros al tiempo para encontrar la solución
Hipótesis	El número de servidores no afecta a la solución. El número de usuarios no afecta a la solución (H0). El número de servidores afecta a la solución. El número de usuarios afecta a la solución (H1).
Método	● Elegiremos 5 semillas aleatorias, una para cada réplica ● Haremos un total de 5 réplicas para cada valor tanto para los servidores como para los usuarios. ● Para los usuarios, iremos incrementando de 100 en 100 hasta que se vea la tendencia (unas 4 veces en total) manteniendo el resto de parámetros fijos. ● Para los servidores, iremos incrementando de 50 en 50 hasta que se vea la tendencia (unas 4 veces en total) manteniendo el resto de parametros fijos. ● Usaremos el algoritmo Hill Climbing ● Usaremos el conjunto de operadores y el generador inicial establecidos antes. ● El cálculo del heurístico viene determinado por el enunciado.

Con este experimento se podrá ver cómo influye el número de servidores y usuarios en el problema para encontrar la solución mediante Hill Climbing. Hemos supuesto que en principio no afectan al tiempo de solución aunque a priori lo más normal es pensar que si afecta al resultado. A continuación se muestran dos tablas con los datos que forman el resultado del experimento. Únicamente se destaca el tiempo para encontrar la solución y los valores de servidores y usuarios usados para cada réplica.

	Réplica					Media
Usuarios	0	1	2	3	4	
100	16868	4884	17603	10135	23140	14526
200	42102	31219	94393	180517	9930	71632.2
300	114123	447971	458293	151473	271703	288712.6
400	1080129	894521	898782	875458	1009967	951771.4

Tabla 4.1: Datos del tiempo en función del número de usuarios.

	Réplica					Media
Servidores	0	1	2	3	4	
50	40769	24419	77621	141868	8571	58649.6
100	56091	162400	57586	7825	71260	71032.4
150	82967	93603	57353	92960	135230	92422.6
200	64128	30569	127709	279378	40548	108466.4

Tabla 4.2: Datos del tiempo en función del número de servidores.

A continuación las gráficas que recogen estos datos (se han hecho a partir de las medias):

Figura 4.1: Tiempo para encontrar la solución en función del número de usuarios.

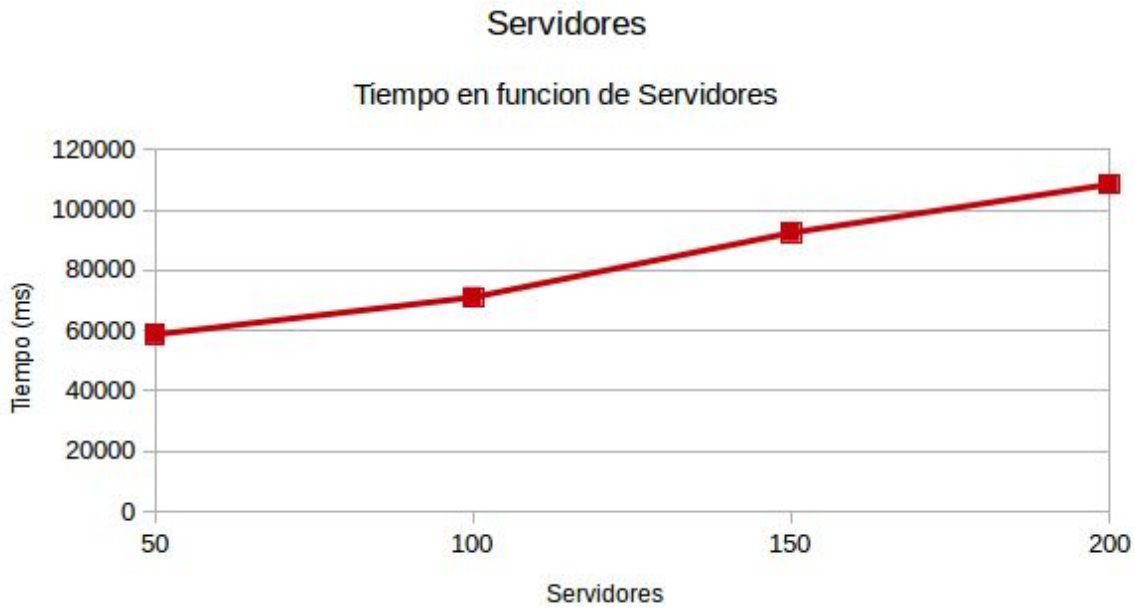


Figura 4.2: Tiempo para encontrar la solución en función del número de servidores.

Conclusiones del experimento 4:

La conclusión de estos datos es que al aumentar el número de usuarios del problema, el tiempo para encontrar la solución crece exponencialmente y al aumentar el número de servidores del problema el tiempo crece de manera linear. Esto se debe a que si se aumenta el número de usuarios, el número de peticiones totales aumenta también y como el algoritmo trabaja sobre las peticiones a servidores el tiempo para llegar a una solución lo suficientemente buena aumenta considerablemente debido a los factores de ramificación de los operadores y a que |S| << |R| (el número de servidores es muy pequeño comparado con las peticiones normalmente). Por esta misma razón también aumenta el tiempo cuando añadimos más servidores, pero de forma lineal, ya que al haber siempre el mismo número de peticiones solo se tarda un poco más en crear sucesores para cada estado debido a los factores de ramificación de los operadores y a que |S| << |R|. Por todo esto se descarta la hipótesis inicial.

Experimento 5:

Dado el escenario del primer apartado, estimad la diferencia entre el tiempo total de transmisión y el tiempo para hallar la solución, usando las dos heurísticas, implementando los criterios definidos, el que habéis usado en los experimentos previos y el otro que optimiza el segundo criterio de calidad (3.3). Usad el algoritmo de Hill climbing. En este caso deberéis experimentar cómo debéis incorporar la penalización del segundo criterio.

Observación	La diferencia entre el tiempo total de transmisión y el tiempo para hallar la solución, usando la heurística que optimiza los dos criterios del enunciado y, usando el mismo algoritmo de búsqueda local que el experimento 1 (Hill-Climbing), es menor con la siguiente penalización para el criterio 2 del enunciado: n+0.5*sum , donde "n" es el criterio 1 a optimizar , dónde 0.5 es el factor de penalización del criterio 2, y donde sum es el criterio 2 (criterio a penalizar).
Planteamiento	Escogemos distintas ponderaciones (penalización) para el criterio 2 y observamos los efectos que producen en el tiempo total de transmisión y el tiempo que tarda en hallar la solución.
Hipótesis	La diferencia entre el tiempo total de transmisión y el tiempo para hallar la solución usando las dos heurísticas y el mismo algoritmo de búsqueda local no es el mismo, teniendo en cuenta que usamos distintas ponderaciones/penalizaciones para el segundo criterio.
Método	● Ponderamos/penalizamos criterio c2 de la siguiente forma: c1+x*c2, donde c1 es el criterio 1, es decir, minimizar el tiempo del servidor que tarda más; y donde c2 es el criterio 2, es decir, minimizar el tiempo total de las transmisiones; y, finalmente, donde x es la ponderación/penalización que se le da al segundo criterio de optimización del enunciado. Experimentamos con 3 penalizaciones diferentes. ● Usaremos el algoritmo Hill Climbing. ● El número de usuarios es 200, el número máximo de peticiones por usuario es de 5, el número de servidores es 50 y el número mínimo de replicaciones de ficheros es de 5. ● El criterio de generador de estados iniciales es iterativo. ● Usamos una semilla para cada réplica. En total, hacemos 10 réplicas, es decir, 10 semillas distintas, como mínimo. ● Se usa el heurístico que optimiza a la vez los dos criterios que nos pide el enunciado; en concreto, se penaliza el c2.

EXP 5	**Penalizacion 1**		**Penalización 2**		**Penalización 3**	
Réplica	**Total(ms) Transmission**	**Total(ms) Solución**	**Total (ms) Transmissio n**	**Total(ms) Solución**	**Total(ms) Transmission**	**Total(ms) Solución**
1	1520599	93066	1531324	118962	1531471	80075
2	1461722	109339	1471132	148545	1472675	97911
3	1564734	130194	1565932	111583	1553117	115233
4	1546989	119338	1546989	112226	1541957	111518
5	1501408	137671	1494076	130527	1501436	124423
6	1360886	90551	1352807	81988	1369173	79243
7	1444435	133691	1436513	102879	1441545	116338
8	1448824	95442	1468757	79836	1456630	95568
9	1526875	135416	1529596	121397	1526246	115550
10	1525962	115518	1529264	109904	1539797	107699

Tabla 5.1: Tiempo total de transmisión y tiempo total en hallar las solución de las 3 penalizaciones

Penalización 1: n + 0.5*sum

Penalización 2: 0.6*n + 0.4*sum

Penalización 3: n + 1.50*sum

Hemos escogido estas tres penalizaciones. Donde “n” es el valor del primer criterio a optimizar que nos pide el enunciado y, donde “sum” es el valor del segundo criterio a optimizar y lo hemos ido ponderando.

A continuación se muestran dos gráficos. El primer gráfico nos ilustra cómo evoluciona el tiempo total de transmisión en las 3 penalizaciones. El segundo gráfico nos ilustra cómo evoluciona el tiempo total para hallar la solución con las tres penalizaciones.

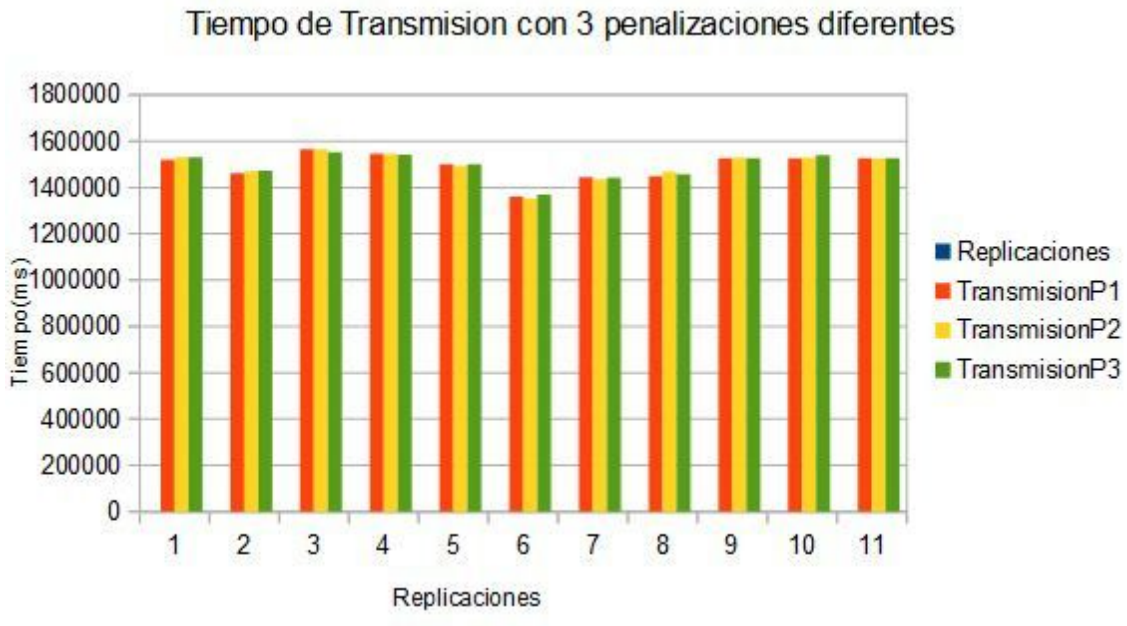


Figura 5.1 :Tiempo total de transmisión con 3 penalizaciones diferentes.

Medias: P1: 1525962 ms; P2: 1525963 ms; P3: 1525964 ms

Nota : La "replicación" 11 NO es una replicación. Es la media del tiempo total de transmisión de las penalizaciones.

Podemos ver que la media está muy ajustada; pero la mejor penalización es la penalización 1, ya que tarda menos (Penalización 1: n + 0.5*sum)

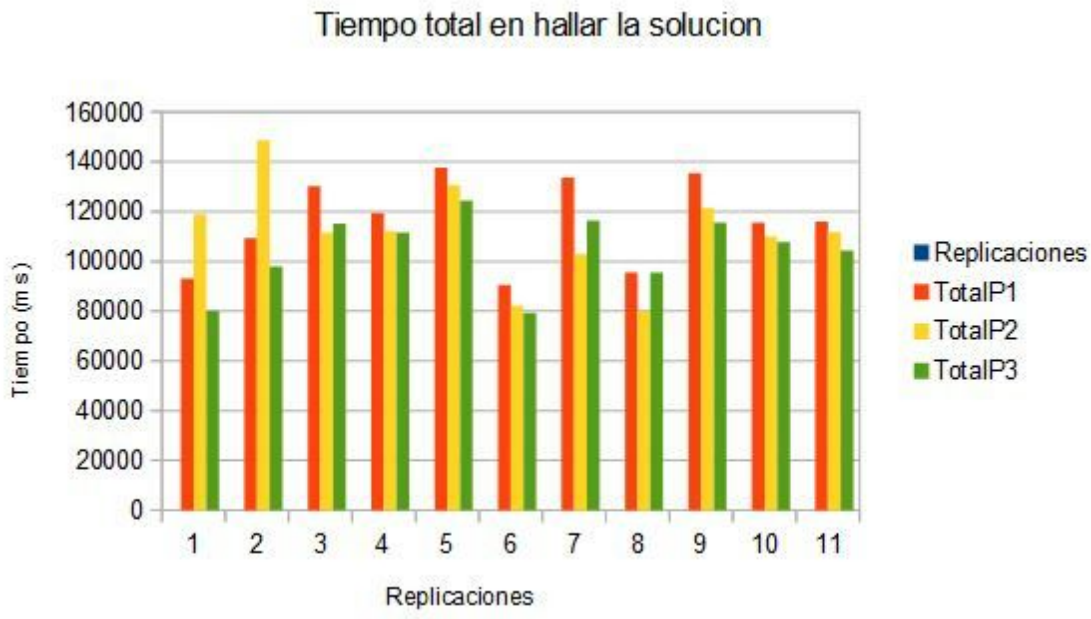


Figura 5.2 : Tiempo en hallar la solución con 3 penalizaciones diferentes

Nota : La “replicación” 11 del gráfico 2 NO es una replicación. Es la media del tiempo total en hallar la solución de las diferentes penalizaciones.

En este caso la penalización que es mejor es la penalización 3, ya que tarda menos (Penalización 3 : n + 1.50*sum)

Medias: P1: 116022.6ms; P2: 111784.7ms; P3: 104355.8ms

En cuanto al tiempo de transmisión podemos observar que la penalización 1 es la mejor, pues tarda menos. En cuanto al tiempo total en hallar la solución, podemos ver que la penalización 3 es la mejor, pues tarda menos.

Una vez hallada cual de estas tres penalizaciones da mejor resultado (en este caso la 1 porque reduce el tiempo total de transmisión del servidor que tarda más, algo que como usuario nos interesa), usaremos dicha penalización para realizar el EXP7: ir aumentando el número de replicaciones de los ficheros, y observar sus efectos en el tiempo total de transmisión y el tiempo total en hallar la solución.

Una vez experimentando con la penalización, nos quedamos con dicha penalización 1. A continuación, se muestra la tabla con la diferencia entre el tiempo total de transmisión y el tiempo total en hallar la solución, usando también las 3 penalizaciones, es decir, tendremos una tabla con la diferencia entre el tiempo total de transmisión y el tiempo en hallar la solución para cada una de las 3 penalizaciones.

Replicaciones	**Diferencia P1 (ms)**	**Diferencia P2 (ms)**	**Diferencia P3 (ms)**
1	1427533	1412362	1451396
2	1352383	1322587	1374764
3	1434540	1454349	1437884
4	1427651	1434763	1430439
5	1363737	1363549	1377013
6	1270335	1270819	1289930
7	1310744	1333634	1325207
8	1353382	1388921	1361062
9	1391459	1408199	1410696
10	1410444	1419360	1432098
Media	1374220.8	1380854.3	1389048.9

Tabla 5.2: Tiempo total de transmisión menos tiempo total en hallar las solución de las 3 penalizaciones

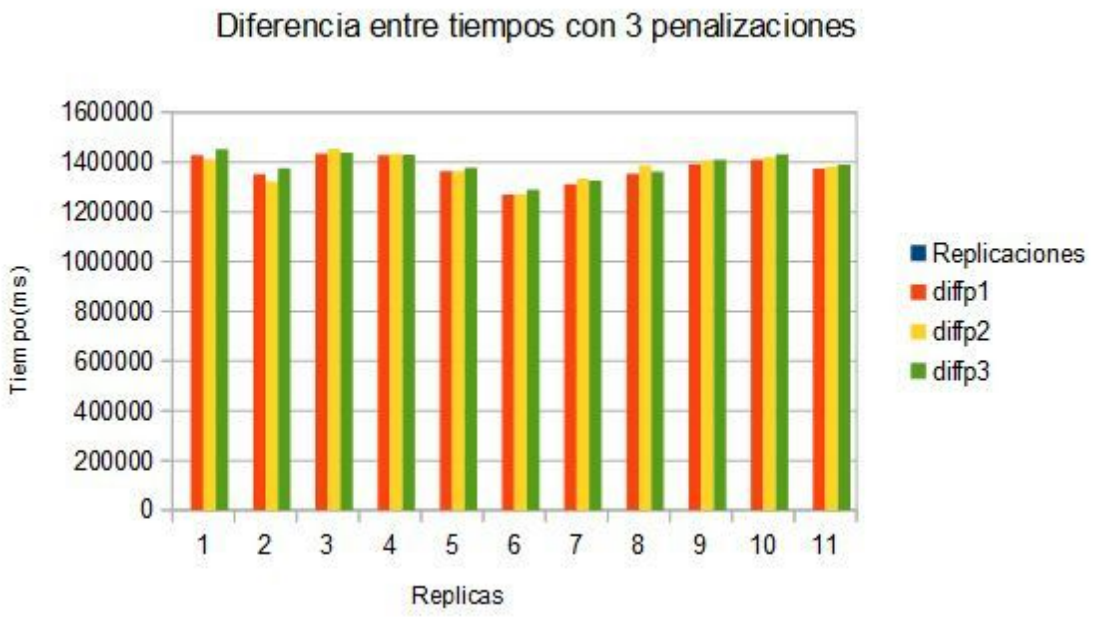


Figura 5.3 : Tiempo total de transmisión menos tiempo total en hallar las solución de las 3 penalizaciones

Nota : La "replicación" 11 del gráfico 2 NO es una replicación. Es la media del Tiempo total de transmisión menos tiempo total en hallar la solución de las 3 penalizaciones

En este caso la penalización que es mejor es la penalización 1, ya que tarda menos (Penalización 1: n + 0.50*sum)

Medias: P1: 1374220.8ms; P2: 1380854.3ms; P3: 1389048.9ms

Conclusiones del experimento 5:

Hemos experimentando con tres penalizaciones distintas, y la que nos da mejor resultado es la penalización 1, es decir, n + 0 .5*sum, penalizando el segundo criterio del enunciado a la mitad. Esto tiene su lógica en que como usuario individual que será cada uno en el problema, nos interesará más que el tiempo de transmisión del servidor que tarda más sea lo menor posible y podemos "despreciar" si la carga del tiempo está perfectamente equilibrada con los servidores a los que *no* estamos conectados, al menos, a nivel de usuario individual.

Experimento 6:

Dado el mismo escenario que en el apartado anterior, estimad la diferencia entre el tiempo total de transmisión y el tiempo de ejecución para hallar la solución con Hill Climbing y Simulated Annealing para las dos heurísticas usadas en el apartado anterior.

Observación	La diferencia entre el tiempo total de transmisión, entre usar el mismo algoritmo de búsqueda local que el experimento 1 (Hill-Climbing) y usar el algoritmo de búsqueda local de Simulated Annealing, en las condiciones del experimento 1 es más o menos despreciable; el tiempo para hallar la solución, en cambio, es bastante más rápido para Hill Climbing.
Planteamiento	Escogemos varias seeds de generación de estado inicial arbitrarias, y probamos 1 vez cada uno de los dos algoritmos para el problema generado con los mismos parámetros y criterio heurístico (el 2).
Hipótesis	El tiempo (tanto el total de transmisión como el de ejecución para hallar una solución) de ambos algoritmos es distinto, ya que recorren el espacio de soluciones de maneras distintas.
Método	● Elegiremos 10 semillas aleatorias, una para cada réplica. ● Ejecutaremos 2 réplicas por semilla, una para cada algoritmo heurístico. En total, haremos 20 ejecuciones. ● El número de usuarios es 200, el número máximo de peticiones por usuario es de 5, el número de servidores es 50 y el número mínimo de replicaciones es de 5. ● Usaremos los algoritmos Hill Climbing y Simulated Annealing. ● Usaremos la misma estrategia de inicialización para ambos algoritmos. ● El cálculo del heurístico es el criterio 2 del experimento anterior, con los coeficientes de penalización 1 (es decir, n + 0.5*sum).

EX 6	Simulated Annealing		Hill Climbing	
Réplica	**Tiempo de trans.**	**Tiempo de ejec.**	**T. de trans.**	**T. de ejec.**
1234	1567721	358495	1556724	77671
538	1488323	320776	1450878	71119
1173	1667856	361609	1669473	88819
2	1504345	350561	1494586	77165
20	1493697	358128	1470789	91739
39	1444288	337348	1519521	80574
4276	1653291	349382	1659399	89003
1163	1564961	323537	1549826	82316
687	1565402	333484	1615635	96908
11	1514853	348085	1582024	75125

Tabla 6.1: Simulated Annealing vs Hill Climbing

A continuación se muestran dos gráficos; el primer gráfico muestra los valores del tiempo total de transmisión en Hill Climbing y en Simulated Annealing. El segundo gráfico compara los tiempos para encontrar una solución para cada uno de los algoritmos.

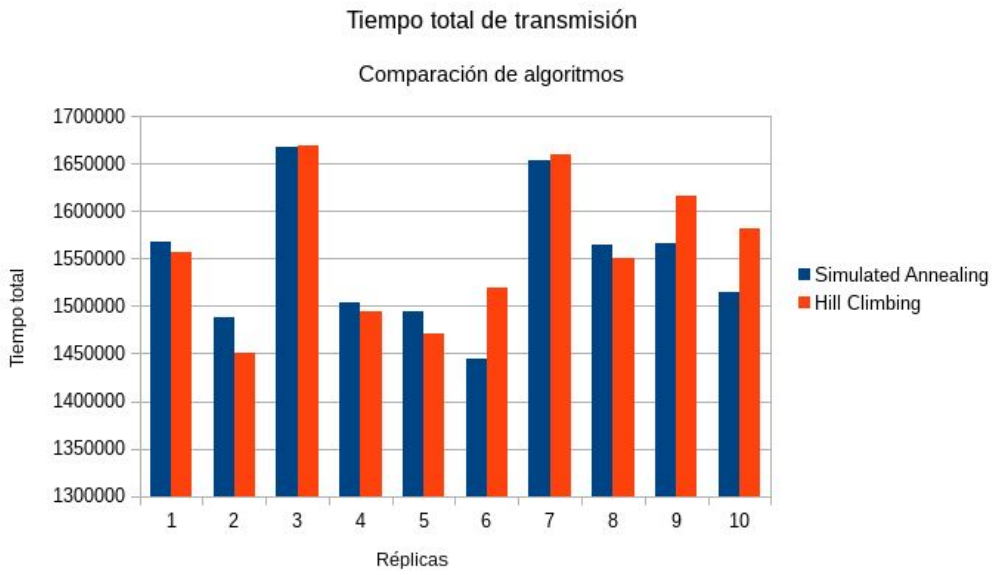


Figura 6.1 :Tiempo total de transmisión con para ambos algoritmos

Podemos ver que dan más o menos valores parecidos, aunque en algunos casos Simulated Annealing tiene una ventaja a considerar.

Medias:

Simulated Annealing: 1546473.7 ms; Hill Climbing: 1556885.5 ms

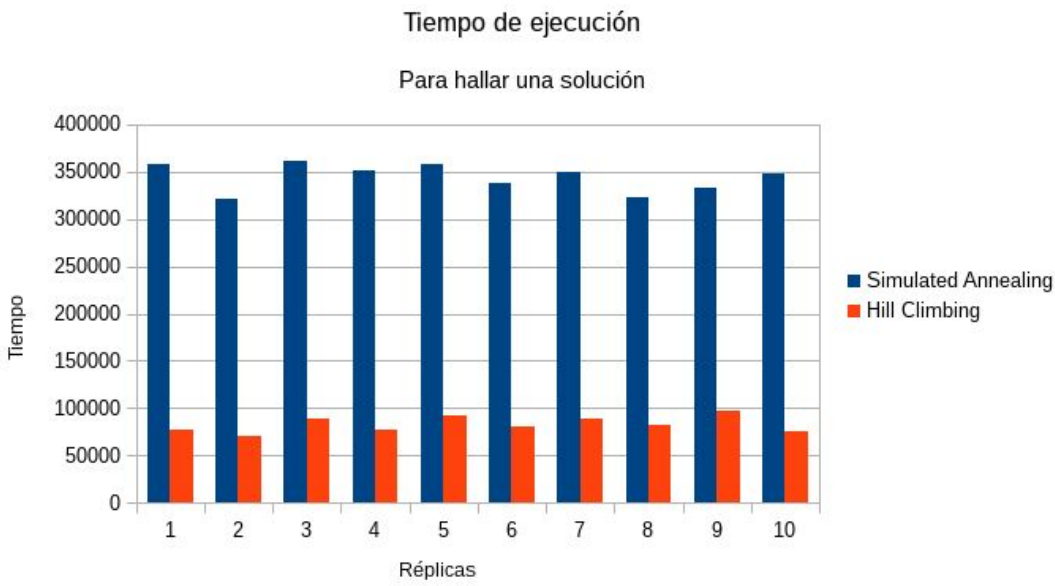


Figura 6.2 : Tiempo en hallar la solución con 3 penalizaciones diferentes

En este caso es claramente observable que el tiempo de cálculo de Hill Climbing es varias veces más rápido que el de Simulated Annealing.

Medias:

Simulated Annealing:344140.5 ms; Hill Climbing: 83043.9 ms

Conclusiones del experimento 6:

En cuanto al tiempo de transmisión, aunque se puede observar una mejora ligera de la solución propuesta por Simulated Annealing sobre la propuesta por Hill Climbing, resulta despreciable comparada con la mejora en tiempo de ejecución de Hill Climbing.

No es difícil, por tanto, determinar Hill Climbing como el algoritmo superior para el problema estudiado en este documento.

Experimento 7:

Un elemento clave en el problema es el número de replicaciones de un fichero. La intuición dice que cuantas más replicaciones, menor es el tiempo de transmisión que se puede conseguir. Usando el escenario inicial, el algoritmo de Hill Climbing y ambas heurísticas, experimentad con el número de replicaciones desde 5 hasta 25 en pasos de 5. Medid el tiempo total de transmisión y el tiempo necesario para hallar la solución.

Observación	Se observa que el tiempo de transmisión no se ve afectado, o apenas se ve afectado . En cuanto al tiempo total para hallar la solución, sí se ve que a medida que se sube el número de replicaciones, éste aumenta considerablemente.
Planteamiento	Aumentamos el número de replicaciones de 5 en 5, empezando desde 5 y acabado en 25, para ver cómo afecta el aumento del número de replicaciones en el tiempo total de transmisión y en el tiempo necesario para hallar la solución.
Hipótesis	Hipótesis: A medida que va aumentando el número de replicaciones de los ficheros, el tiempo en hallar la solución y el tiempo total de transmisión disminuye.
Método	● Usaremos el algoritmo Hill Climbing. ● El número de usuarios es 200, el número máximo de peticiones por usuario es de 5, el número de servidores es 50. ● El número de replicaciones de los ficheros va subiendo de 5 en 5 (Parte variable del problema). ● El generador de estados iniciales es iterativo ● Se usa el heurístico que optimiza los dos criterios que nos pide el enunciado.Incluyendo la penalización del segundo criterio que se ha experimentado en el EXP5. ● Usamos una semilla para cada réplica; en total, hacemos 10 réplicas, es decir, 10 semillas distintas, como mínimo, cada vez que aumentamos en 5 el número de replicaciones. Terminaremos con un total de 50 ejecuciones, con lo que analizaremos las tablas y los gráficos para analizar los resultados.

Una vez hecho el experimento número 5, hemos visto que p1 (penalización 1) del criterio 2 del enunciado va mejor de cara a los 2 tiempos. Con lo cual, reducimos a la mitad la influencia del criterio dos del enunciado en el heurístico que optimiza los dos criterios a la vez.

El desarrollo del experimento será de la siguiente forma: lo dividiremos en dos partes; en la primera parte, observamos cómo afecta el aumento del número de replicaciones de los ficheros en el tiempo de transmisión. La segunda parte, será esencialmente lo mismo, pero

ahora observaremos cómo afecta el aumento del número de replicaciones de los ficheros al tiempo total en hallar la solución.

A continuación, se muestran las tablas y los gráficos a medida que vamos aumentando el número de replicaciones de los fichero en el servidor.

Primera parte del experimento: Afectaciones al tiempo total de transmisión. Se muestra una tabla con los resultados y su correspondiente gráfica.

EXP7	**Réplicas de ficheros**				
Replicas	5	10	15	20	25
1	1520599	1531874	1519875	1439612	1402491
2	1461722	1480096	1469068	1466278	1435143
3	1564734	1531258	1459922	1470048	1444585
4	1546989	1489455	1557115	1440002	1547359
5	1501408	1554479	1504895	1532650	1580284
6	1360886	1407223	1420728	1423942	1364601
7	1444435	1446241	1459033	1484325	1461955
8	1448824	1530868	1567722	1486743	1438075
9	1526875	1637457	1544019	1516431	1542141
10	1525962	1487768	1477291	1468503	1561931
Media	1337647,2	1360895,1	1350237,7	1326003,1	1321663,4

Tabla 7.1: Tiempo total de transmisión. Aumento del número de replicaciones de los ficheros

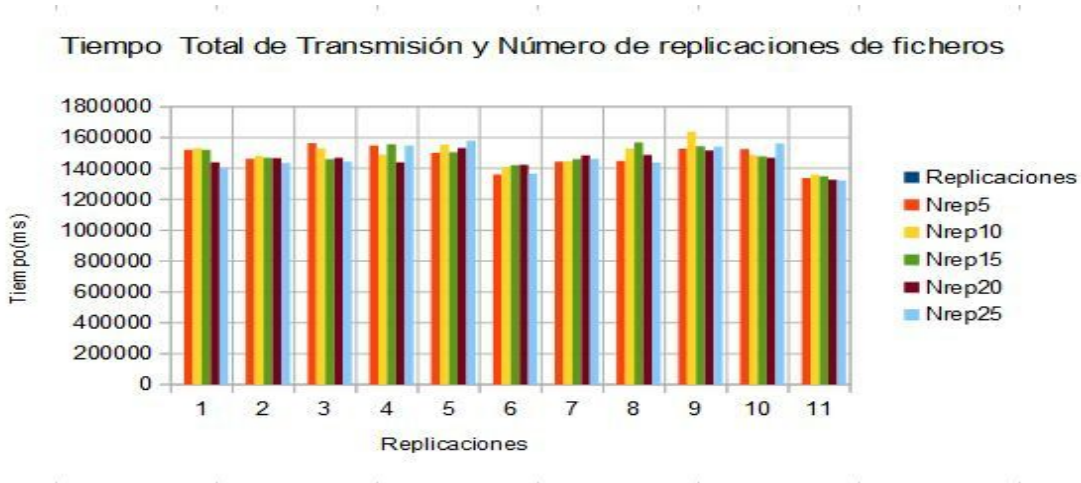


Figura 7.2: Tiempo total de transmisión. Aumento del número de replicaciones de los ficheros

Segunda parte del experimento: Afectaciones al tiempo total en hallar la solución. Se muestra una tabla con los resultados y su correspondiente gráfica.

EXP7	**Réplicas de ficheros**				
Replicas	5	10	15	20	25
1	18885	71338	154955	334222	420040
2	20303	71160	140802	292591	426288
3	20703	75122	149563	298157	473745
4	19870	66540	162243	271203	430495
5	22891	79817	202064	335923	404595
6	17042	64864	152721	235469	401542
7	20530	86562	158515	291133	460680
8	18325	68313	177877	293217	452378
9	22208	78560	217197	297431	507291
10	21530	67242	202631	345931	573620
Media	20228,7	72951,8	171856,8	299527,7	455067,4

Tabla 7.2 : Tiempo total en hallar la solución. Aumento del número de replicaciones de los ficheros

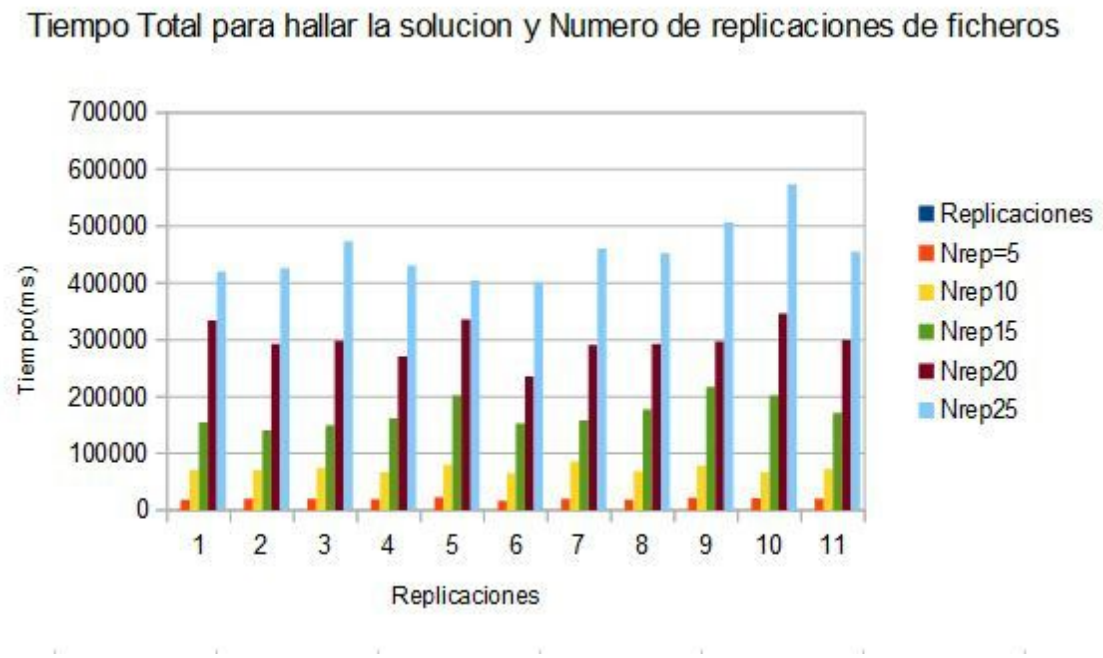


Figura 7.3: Tiempo total de transmisión. Aumento del número de replicaciones de los ficheros

Los tiempos son diferentes porque la forma de generar todos los sucesores en memoria, cuesta mucha memoria, y ralentizan por el recogedor de basuras de java, como no cabía en memoria, se decidió adaptar la función generadora de sucesores del HC de este experimento de forma que solo guarde en memoria estados sucesores que pudiesen ser escogidos, es decir, no guardamos los estados que empeoran el heurístico.

Conclusiones del experimento 7:

Conclusiones de la primera parte: Afectaciones al tiempo total de transmisión

En este caso, se concluye que aunque aumenta el número de replicaciones de los ficheros, esto no afecta a mucho al tiempo total de transmisión.

Conclusiones de la segunda parte: Afectaciones al tiempo total en hallar la solución

En este caso, se concluye que si aumenta el número de replicaciones de los ficheros, afecta considerable al tiempo total en hallar la solución.

Experimento 8:

8. Dados los resultados obtenidos con todos estos experimentos, comparad las dos aproximaciones y comentad las ventajas y desventajas de cada una. ¿Cual de las dos consideráis que es una mejor solución?

Hemos observado que, para este problema en concreto, el algoritmo de aproximación de Hill Climbing es objetivamente superior al de Simulated Annealing.

El problema del algoritmo de Simulated Annealing es que en la consideración de vecinos no se consideran todos, si no sólamente *algunos*, y luego decide probabilísticamente si efectuar el cambio de estado o no. Este algoritmo es apto en aquellos casos en los cuales, en un espacio de soluciones, un mismo estado dispone de muchos vecinos para los que no es práctico analizarlos todos, o no puede hacerse un recorrido unidireccional por el espacio, como hace (y a lo que está limitado) Hill Climbing.

En un problema finito como el nuestro, es perfectamente posible definir una heurística de valores que realizan el espacio de soluciones con un único máximo, de forma que se examinan todos los vecinos y se desplaza a aquél con mayor altura, y así poder recorrer el espacio siempre hacia arriba. Y resulta factible analizar todos los vecinos, porque nuestro número limitado de operadores asegura un número práctico de vecinos para analizar.

Por lo tanto, Simulated Annealing será en este caso **siempre peor** en todos los casos en los que, al analizar *algunos* vecinos, en algún momento del recorrido no se analize el mejor vecino, y sólo por azar y con poca probabilidad el algoritmo de Simulated Annealing analizará consecutivamente el mejor vecino de cada estado. O, dicho de otra forma, **Hill Climbing es el mejor algoritmo de los dos** para la resolución de nuestro problema.