Inteligencia Artificial

Práctica de búsqueda local

Marzo 2020

Aleix Boné Ribó
Alex Herrero Pons
Moisés Balcells

Índice

1. Descripción del problema **1**
1.1. Elementos del problema . . . 1
1.2. Notas importantes . . . 1
1.3. Objetivo . . . 1
1.4. Variables . . . 2

2. Implementación del estado **2**
2.1. Primera versión . . . 2
2.2. Segunda versión . . . 3
2.3. Versión definitiva . . . 3

3. Operadores que hemos elegido **3**

4. Estrategias para hallar la solución inicial **3**

5. Funciones heurísticas **4**
5.1. Primera heurística . . . 4
5.2. Segunda heurística . . . 4

6. Experimentos **5**
6.1. Influencia de los operadores . . . 5
6.2. Influencia de la solución inicial . . . 6
6.3. Parámetros para el simulated annealing . . . 7
6.4. Evolución del tiempo de ejecución para valores crecientes de los parámetros 11
6.4.1. Número de usuarios que piden ficheros . . . 11
6.4.2. Número de servidores (Manteniendo el número de replicaciones) . . 14
6.5. Comparación de las dos heurísticas al calcular el Tiempo total de transmisión y tiempo para hallar la solución utilizando Hill Climbing . . . 16
6.6. Comparación de las dos heurísticas al calcular el Tiempo total de transmisión y tiempo para hallar la solución utilizando Simulated Annealing . . . 18
6.6.1. Comparación con Hill Climbing . . . 19
6.7. Influencia del número de replicaciones de los ficheros . . . 20
6.7.1. Primera heurística . . . 20
6.7.2. Segunda heurística . . . 23
6.8. Comparación de las heurísticas . . . 25

7. Conclusión **25**

1. Descripción del problema

1.1. Elementos del problema

Los elementos del problema consisten en un conjunto de servidores y usuarios distribuidos de manera geográfica. Cada uno de estos servidores tiene a su vez un conjunto de ficheros. Cada fichero esta identificado por un identificador único y varios servidores pueden tener copias de un mismo fichero (replicaciones) pero no necesariamente todos los servidores tienen todos los ficheros. Cuando un usuario hace una petición (de un fichero) un servidor central que se encarga de distribuir las peticiones realizadas por los usuarios. Este servidor tiene información sobre el contenido de cada uno de los servidores e indica al usuario a que servidor acceder para obtener el fichero que ha pedido. El servidor que gestiona las peticiones sabe también el tiempo de transmisión de cada uno de los servidores a cada usuario. Al igual que los ficheros, los servidores, los usuarios y las peticiones que estos realizan se identifican mediante IDs.

1.2. Notas importantes

Los ficheros son servidos de manera secuencial y los tiempos que se tarda en servir una petición no pueden superar a los 5000ms ni bajar de los 100ms. Un usuario puede hacer una o múltiples peticiones de distintos ficheros y un fichero puede ser pedido muchas veces por distintos usuarios.

1.3. Objetivo

El objetivo del problema es asignar para cada petición realizada el servidor que servirá el archivo al usuario. Debemos tener en cuenta que se considera una mejor solución aquella que tiene un menor tiempo de transmisión total y aquella que tiene las cargas de los servidores distribuidas de manera equilibrada. Ya que si únicamente tuvieras en cuenta que fuera el menor tiempo de transmisión total posible las cargas de los servidores no serian equilibradas cosa que no nos interesa.

Así pues, una solución del problema es una asignación para cada petición a un servidor. Debido a que queremos encontrar una solución que minimiza el tiempo de carga de los servidores y equilibre el trabajo de estos se trata de un problema de optimización.

1.4. Variables

Las variables que afectan al problema y a las que nos referiremos en el resto de la práctica son las siguientes:

1. `USERS`: número de usuarios.
2. `NSERV`: número de servidores.
3. `REQUESTS`: máximo numero de peticiones por usuario.
4. `NREP`: número de replicaciones mínimo de cada fichero.

Tal como hemos dicho en la sección previa, una solución es una asignación de cada petición a un servidor. Considerando esto y con las variables del problema que hemos definido tenemos un espacio de soluciones de:

$$\texttt{REQUESTS*USERS*REPLICACIONES} \tag{1}$$

$$\texttt{NREP} \leq \texttt{REPLICACIONES} \leq \texttt{NSERV} \tag{2}$$

El espacio de soluciones es por lo tanto muy grande para valores relativamente pequeños de usuarios, peticiones y servidores, por lo que una búsqueda exhaustiva no seria eficiente. Dada una solución válida, podemos explorar soluciones vecinas cambiando uno de los servidores de una de las peticiones.

2. Implementación del estado

Para lograr el estado inicial con el cuál hemos ejecutado los experimentos hicimos distintas versiones del estado inicial hasta que nos quedamos con la que creíamos que era mejor.

2.1. Primera versión

La primera versión de la implementación planteada era muy simple y consistía guardar un array con los identificadores de los servidores que enviarían el fichero de la petición correspondiente al índice del array, es decir para cada petición en el array guardábamos el servidor que iba a responder esa petición.

Esta versión era eficiente en memoria debido a los pocos datos que se almacenan. Pero al empezar a implementar las heurísticas nos percatamos que se tenían que realizar muchos cálculos para obtener los valores de las heurísticas. Esto hacia que la ejecución fuera muy lenta y se calculaban muchos datos de forma repetida.

2.2. Segunda versión

Para evitar la repetición de cálculos al computar las heurísticas de la primera versión decidimos añadir al estado dos variables adicionales: una que guardaba el tiempo total de todas las transmisiones asignadas y un array con el tiempo de transmisión de cada petición para el servidor asignado en el estado.

Con esta implementación, el cálculo de las funciones heurísticas tardaba menos que en la primera versión, debido a que hora guardamos variables que antes no guardábamos las cuales nos facilitan el cálculo de las funciones heurísticas. Sin embargo la mejora no era muy sustancial ya que seguíamos requiriendo bastantes cálculos que se podían evitar. En cuanto a la eficiencia en la memoria esta sigue siendo eficiente pero menos que antes.

2.3. Versión definitiva

Finalmente decidimos guardar también el tiempo de transmisión de los ficheros de cada servidor, ya que para la primera heurística necesitábamos el tiempo de los servidores, no de las peticiones individualmente. Puesto que los IDs de los servidores no estaban delimitados, tuvimos que usar un Map de entero entero donde guardamos para cada ID de servidor el tiempo total de transmisión de las peticiones que se le asignaban. De este modo el cálculo de las heurísticas era mucho más rápido y el tiempo de ejecución se reducía considerablemente.

Esta versión final, nos proporciona el menor tiempo de ejecución de todas las versiones anteriores, y teniendo en cuenta que mantiene una eficiencia en memoria aceptable, para el tipo de problemas al que nos enfrentamos creemos que es la mejor opción.

3. Operadores que hemos elegido

Definimos dos operadores, uno que cambiaba la asignación de uno de los servidores que daba una petición por otro distinto. La otra operadora era muy similar, hacia lo mismo pero con la restricción de que solo se podía hacer cambios a servidores que no se hubieran probado antes para ese archivo.

La segunda operadora reduce mucho la ramificación respecto a la primera y a medida que se avanza en la búsqueda se va reduciendo aún más. Por consiguiente debería ser más rápida que la primera operación, pero es muy restrictiva y seguramente no permita llegar a estados de solución tan buenos como los alcanzables por la primera operación.

4. Estrategias para hallar la solución inicial

Implementamos dos estrategias para hallar la solución inicial, una muy naïve que cogía el primer servidor de la lista de servidores que tenia el fichero para cada petición y otra greedy que cogía para cada petición el servidor que tenía menor tiempo de transmisión al usuario. La naïve tenía muchos problemas, el principal que al seleccionar siempre el primer servidor de la lista de servidores que tenían el fichero, la carga de trabajo se distribuía siempre en los mismos servidores para cada fichero.

5. Funciones heurísticas

Implementamos dos funciones heurísticas tal y como pedía el enunciado. La primera minimiza el tiempo de transmisión de los ficheros para el servidor que necesita mas tiempo para transmitir sus peticiones. La segunda minimiza el tiempo total de transmisión de los ficheros pero con la restricción de que los tiempos de transmisión de los servidores han de ser lo más similares posibles entre ellos

5.1. Primera heurística

Para la primera heurística simplemente debemos computar el tiempo de transmisión de los ficheros de cada servidor y hallar el máximo. En nuestra primera versión del estado esto suponía recorrer todas las peticiones y calcular para cada una de ellas el tiempo de transmisión del servidor al usuario y sumar-lo al tiempo de cada servidor. Después recorríamos los tiempos de los servidores y encontrábamos el máximo. Este método era lineal en el número de peticiones y de servidores y se tenia que computar para cada rama de cada estado por el que pasábamos, era por lo tanto muy poco eficiente.

Con la versión definitiva del estado al tener un Map con el tiempo de cada servidor nos ahorramos el paso de recorrer y computar el tiempo de todas las peticiones y solo tenemos que encontrar el máximo entre los servidores, esta operación es lineal en el número de servidores y aunque no lo parezca es mucho más eficiente que la primera implementación ya que el número de servidores es mucho menor al número de peticiones y no tenemos que hacer llamadas a las funciones de `DistFS`.

5.2. Segunda heurística

La segunda heurística es más compleja ya que se tienen que considerar dos factores: la minimización del tiempo total de transmisión y el equilibrio de cargas entre servidores.

Nos decidimos por calcular el tiempo total de transmisión y multiplicar-lo por la desviación estándar de los tiempos de transmisión de cada servidor.

6. Experimentos

6.1. Influencia de los operadores

Tal como se explica en la sección 3, implementamos dos versiones de operadores, una que permitía acceder a todo el espacio de búsqueda y otra más restrictiva que evitaba repetir combinaciones de fichero servidor.

Hipótesis. *El operador más restrictivo dará resultados algo peores debido que limita el espacio de búsqueda, pero por este mismo motivo se reducirá en gran cantidad el tiempo de ejecución.*

La tabla 1 muestra un resumen del experimento, con los valores que usamos y los resultados obtenidos. Decidimos incluir también el experimento 2 en ya que como se puede observar en la tabla la solución inicial influya también sobre la eficacia del operador. Se realizaron 100 repeticiones para cada caso.

Tabla 1: Parámetros y resultados del experimento 1

Parámetro	valor
USERS	200
NSERV	50
NREP	5
REQUESTS	5

Operador	Generador	Heurística
1	naïve	73574
	greedy	**12274**
2	naïve	65460
	greedy	12383

El mejor resultado lo obtuvimos con el primer operador y la heurística greedy. Con el segundo operador obtuvimos mejores resultados que con el primero si se usaba la heurística naïve como se puede ver en la tabla.

En las tablas 2 y 3 se puede ver en detalle los resultados de los experimentos[1]. Al contrario de lo que creíamos inicialmente, el tiempo de ejecución entre los dos operadores es muy similar. El resto de datos también son muy cercanos entre sí, por lo que consideramos que no era necesario el segundo operador ya que no daba mejores resultados ni en tiempo de ejecución ni en calidad de la solución, y además limitaba el espacio de búsqueda.

Tabla 2: Resultados del experimento 1 con el primer operador

Generador	T_{ej}		TTT		Heuristica	
	media	std	media	std	media	std
0	3.44893	1.709478	1480128.90	72604.13	73573.59	27387.23
1	0.95197	0.249743	539064.86	29720.42	12273.96	978.54

Tabla 3: Resultados del experimento 1 con el segundo operador

Generador	T_{ej}		TTT		Heuristica	
	media	std	media	std	media	std
0	3.24974	1.462422	1485563.24	73873.02	65460.31	26056.55
1	0.87680	0.257158	544767.65	39168.81	12382.63	994.10

[1]La tabla 2 se corresponde a la tabla 4 del segundo experimento

6.2. Influencia de la solución inicial

Tal como se ha comentado en la sección 4, implementamos 2 versiones distintas para generar la solución inicial. Una que cogía el primer servidor de la lista de servidores que daban ficheros para cada petición y otra que buscaba entre los servidores que tenían el fichero, el que tenía mínimo tiempo de transmisión para el usuario de la petición.

La primera versión tenía un coste computacional mínimo ya que solo se accedía al primer elemento del conjunto retornado por `DistFS.Server.fileLocations(fileID)`. La otra versión tenia que recorrer todo el conjunto para encontrar el servidor con tiempo de transmisión mínimo, lo que equivale a un coste lineal en $NREP$ (Número de replicaciones mínima de un fichero).

Hipótesis. *A pesar de que la segunda versión del generador de estado inicial es mucho más costoso, esta limitado por $NREP$ que es una constante y no muy grande ($^{NREP}/_{NSERV} < 0{,}5$), por lo que el coste es despreciable y el tener un mejor estado inicial nos permite llegar a un máximo local más óptimo.*

Tal como se especifica en el enunciado de la práctica usamos los mismos parámetros que en el experimento 1 y repetimos el experimento 100 veces en el mismo ordenador para minimizar los efectos externos que pueden afectar al tiempo de ejecución.

Tabla 4: T_{ej}, TTT y heurística obtenidos con los generadores de estado inicial

Generador	T_{ej}		TTT		Heuristica	
	media	std	media	std	media	std
0	3.44893	1.709478	1480128.90	72604.13	73573.59	27387.23
1	0.95197	0.249743	539064.86	29720.42	12273.96	978.54

Los resultados obtenidos se muestran en la tabla 4, en ella podemos apreciar claramente como el generador greedy no solo es significativamente más rápido que el de elegir el primer servidor que se encuentra sino que también llega a un máximo local con una heurística menor y un tiempo total de transmisión (TTT) también menor (incluso la desviación estándar es menor en un orden de magnitud). Se cumple nuestra hipótesis inicial y dados los resultados de este experimento decidimos usar el segundo generador para el resto ya que es mejor en todos los ámbitos analizados.

6.3. Parámetros para el simulated annealing

El objetivo de este experimento es ajustar los parámetros del Simulated Annealing para conseguir mejores resultados que con Hill Climbing.

Hicimos el estudio para el mismo escenario que en los experimentos anteriores, utilizando la función heurística, los operadores y la estrategia de generación de la solución inicial escogidos. Con el fin de facilitar la comparación de resultados también establecimos la semilla del generador de números aleatorios 1234.

Para asegurarnos que la búsqueda llega a converger hemos establecido el número de iteraciones a 10^6.

Hipótesis. *Con Simulated Annealing dará mejores resultados con el parámetro* k *lo más grande posible y* λ *lo más cercano a 0. Por otra parte* stiter[2] *dará mejores resultados cuanto más grande sea pero no habrá mucha diferencia entre los distintos valores.*

La mayor dificultad para ajustar los parámetros es que la única indicación que se tiene son los resultados obtenidos experimentalmente, eso obliga hacer pruebas exhaustivas para ver como se comporta el problema.

En nuestro caso probamos los siguientes valores:

$$\begin{aligned} stiter &= \{100,\ 1000,\ 10000\}, \\ k &= \{10,\ 100,\ 1000,\ 10000\}, \\ \lambda &= \{0{,}01;\ 0{,}001;\ 0{,}0001\}. \end{aligned}$$

Siguiendo la hipótesis inicial los parámetros esperados serían $stiter = 10000$, $k = 10000$ y $\lambda = 0{,}0001$.

Para cada combinación de parámetros hemos realizado 10 ejecuciones del programa y hemos extraído la media del tiempo total de transmisión. Estos resultados se pueden comparar con claridad en la figura 1.

[2]Iteraciones por cada cambio de temperatura

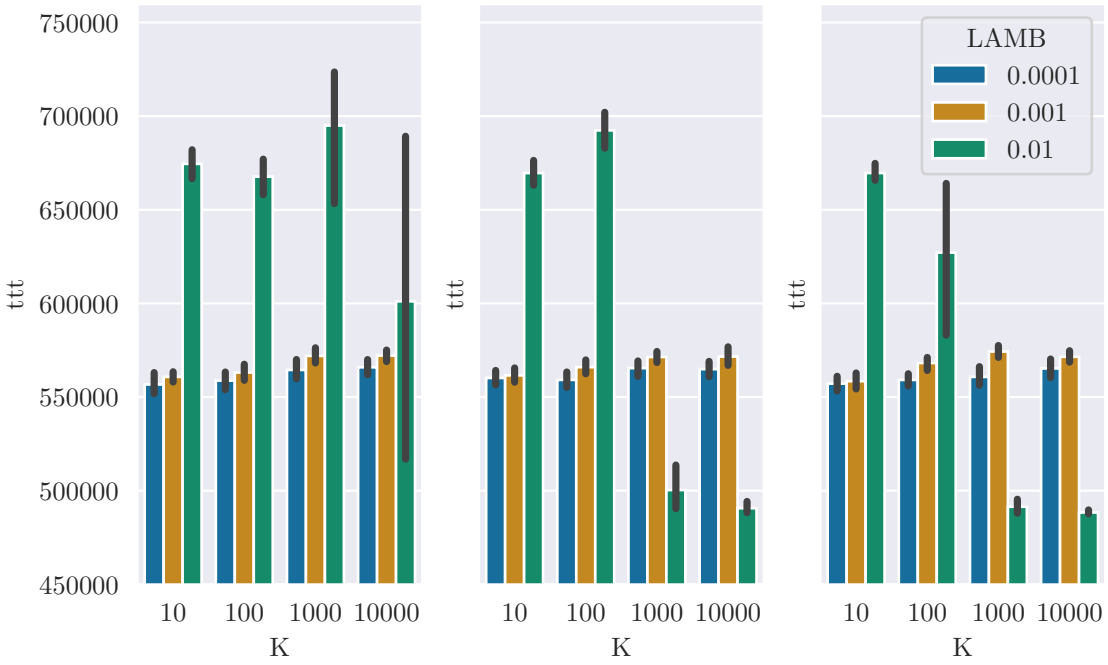

Figura 1: Tiempo de transmisión total para los distintos parámetros.
($stiter$ = 100 1000 10000)

Como se puede observar en la gráfica los mejores resultados los hemos obtenido con $stiter = 10000$, $k = 10000$ y $\lambda = 0{,}01$ con un tiempo total de transmisión medio de 488365.8 ms notablemente menor que el de Hill Climbing (559269ms).

Este resultado como se puede comprobar concuerda con nuestra hipótesis inicial en los valores de $stiter$ y k, pero en el caso de λ nos dio un resultado bastante inesperado, del cual extrajimos que valores demasiado pequeños podían dar peores resultados.

También cabe destacar que por lo general no hay mucha diferencias entre los distintos valores de $stiter$, pero si nos fijamos de nuevo en la figura 1 se ve como en el caso de $k = 10000$ y $\lambda = 0{,}01$ se observa una gran diferencia entre $stiter = 100$ y los otros valores.

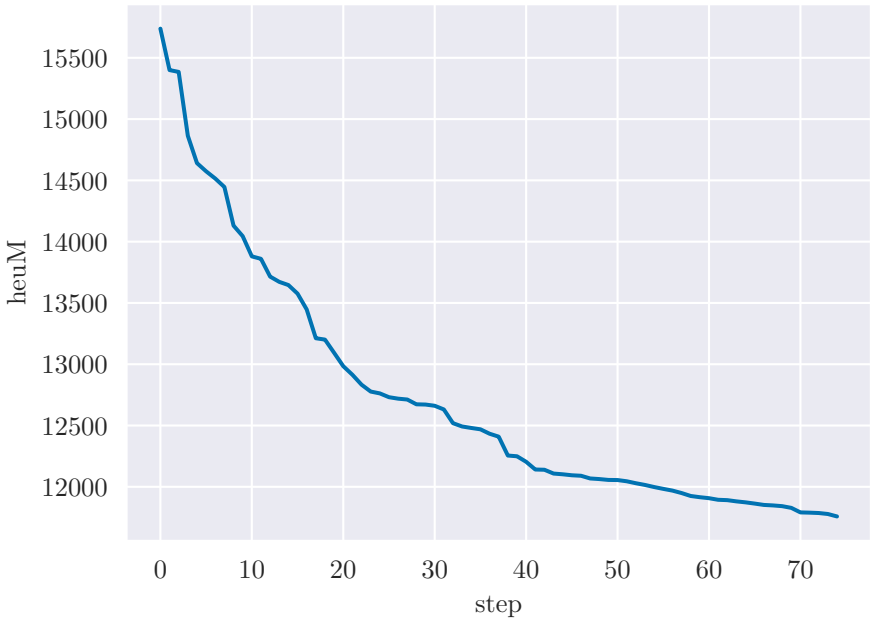

Figura 2: Evolución de la heurística (HC)

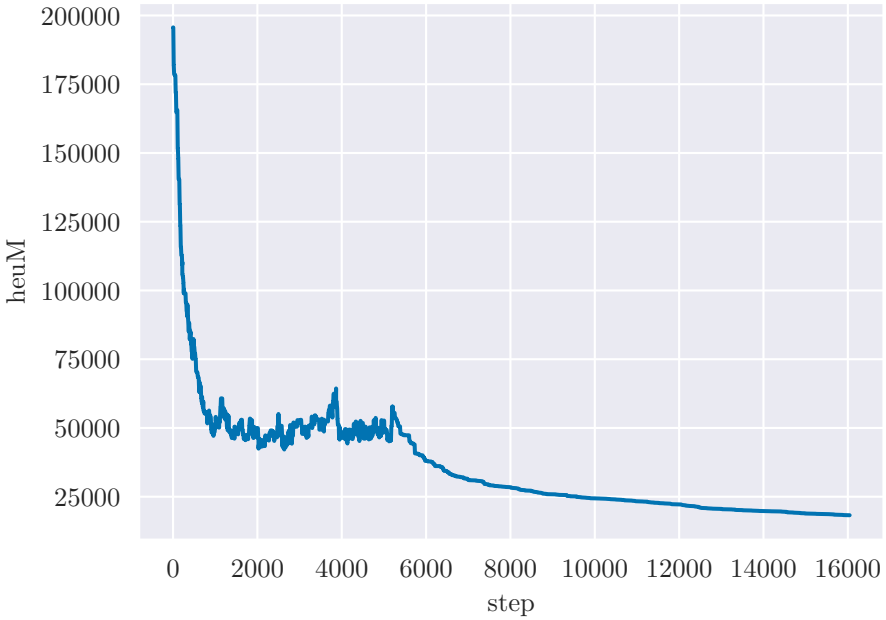

Figura 3: Evolución de la heurística (SA/ $stiter = 10000$, $k = 10000$ y $\lambda = 0{,}01$)

Por otro lado también hemos estudiado la evolución de la heurística utilizada a lo largo del problema. Observando la gráfica 3 y comparándola con la de Hill Climbing (figura 2 vimos que con los parámetros antes mencionados se obtenía un valor de *heuM* mayor (18261 y 11758 respectivamente).

Curiosamente con parámetros que daban peor tiempo total de transmisión que HC se obtuvo mejor valor de *heuM* como en el caso de $stiter = 10000$, $k = 10$ y $\lambda = 0{,}001$ (figura 4) con un tiempo total de transmisión y *heuM* iguales a 568187.7ms y 11359ms respectivamente.

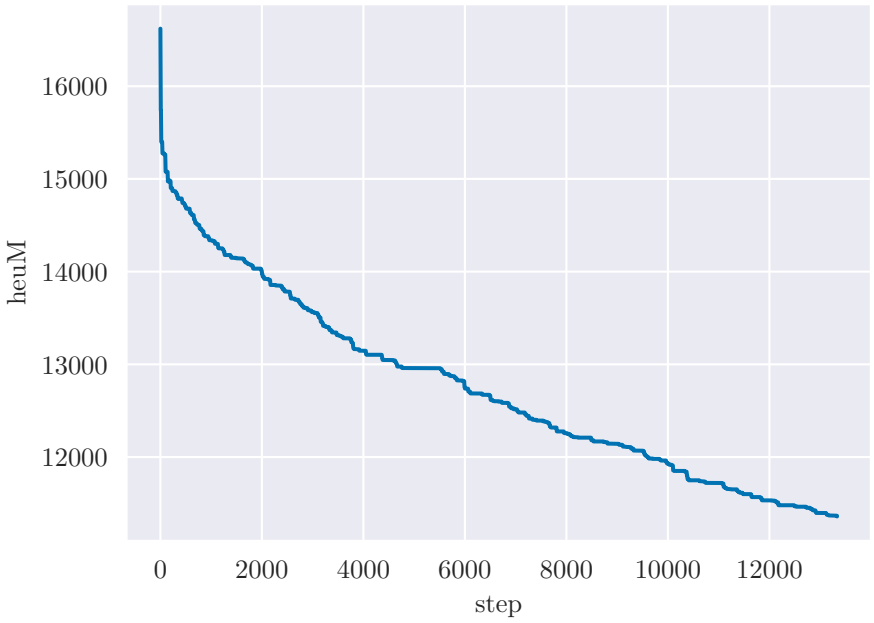

Figura 4: Evolución de la heurística (SA/ $stiter = 10000$, $k = 10$ y $\lambda = 0{,}001$)

6.4. Evolución del tiempo de ejecución para valores crecientes de los parámetros

En estos experimentos estudiamos la evolución del coste de la búsqueda en función del tamaño del problema. Para ello consideramos 2 parámetros: el número de usuarios que piden ficheros y el número de servidores.

Retomando lo explicado en la sección 1.4, el número de estados del problema es: `REQUESTS*USERS*REPLICACIONES`, donde `REPLICACIONES` esta delimitado por el número mínimo de replicaciones de un fichero y el número de servidores.

Hipótesis. *El número de estados crece más rápido al aumentar el número de usuarios que al aumentar el número de servidores. Además tener más servidores implica que es más fácil distribuir la carga de modo equitativo, por lo que es posible que se llegue más rápidamente a los mínimos locales.*

6.4.1. Número de usuarios que piden ficheros

Tabla 5: Parámetros experimento 4 (Usuarios)

Parámetro		valor
USERS	$100 \rightarrow 1000$	(+100)
NSERV		50
NREP		5
REQUESTS		5

Realizamos los experimentos con los parámetros que se muestra en la tabla 5. Incrementamos el número de usuarios dese 100 hasta 1000 en incrementos de 100. Se realizaron 100 repeticiones para cada valor de usuario, cada uno con una semilla distinta. Se midió el tiempo de ejecución y se calculó la media y la desviación estándar. Los resultados se muestran en la tabla 6:

Tabla 6: Resultados del experimento 4 (Usuarios)

USERS	Media	std
100	0.48564	0.093651
200	1.00943	0.304861
300	1.53599	0.509171
400	2.17553	0.870111
500	2.98758	1.185712

USERS	Media	std
600	4.26529	1.776612
700	5.60846	2.478206
800	8.87441	4.385212
900	11.59919	6.073314
1000	14.95716	8.684191

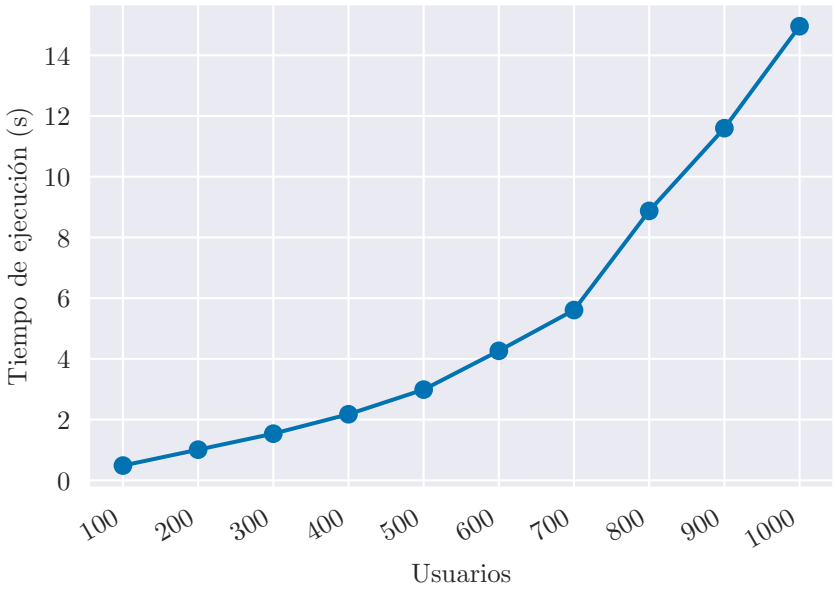

Figura 5: Media Experimento 4 (Usuarios)

A partir de estos resultados generamos una gráfica con las medias obtenidas para cada valor de usuarios (figura 5). Se puede apreciar un crecimiento que aparentemente parece cuadrático.

En el boxplot que realizamos (figura 6) podemos observar como el tiempo de ejecución varia mucho a medida que se van incrementando el número de usuarios.

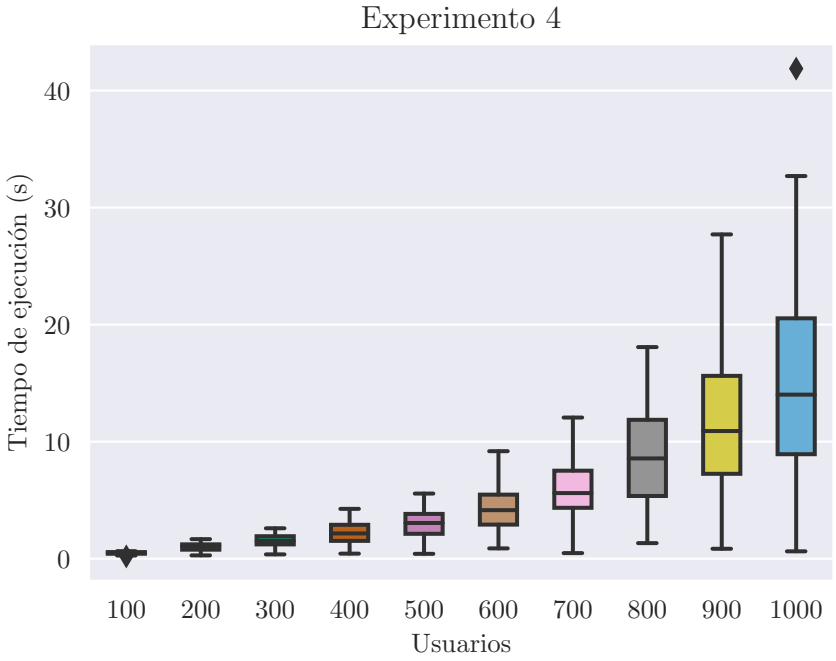

Figura 6: Boxplot T_{ej} del experimento 4 (Usuarios)

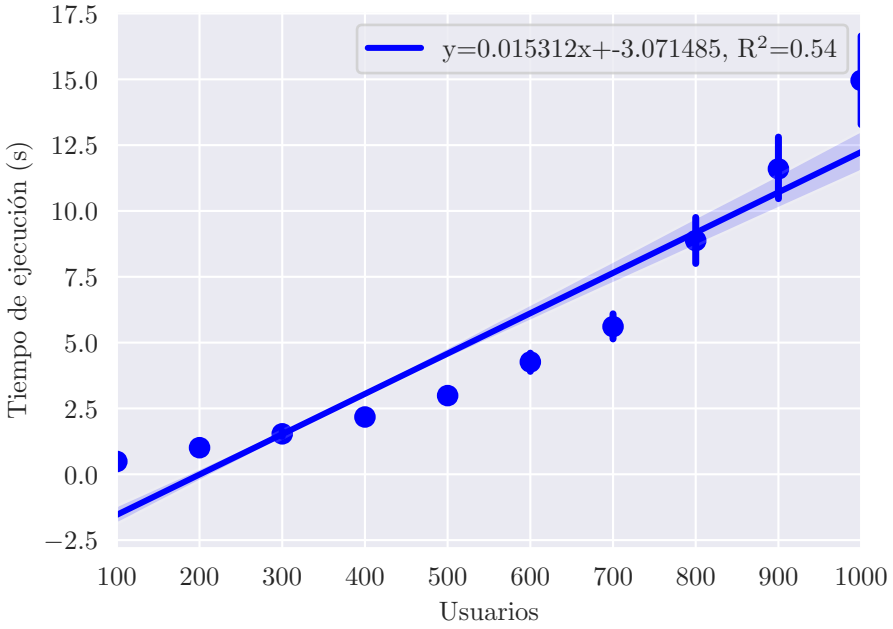

Figura 7: Regresión lineal T_{ej} del experimento 4 (Usuarios)

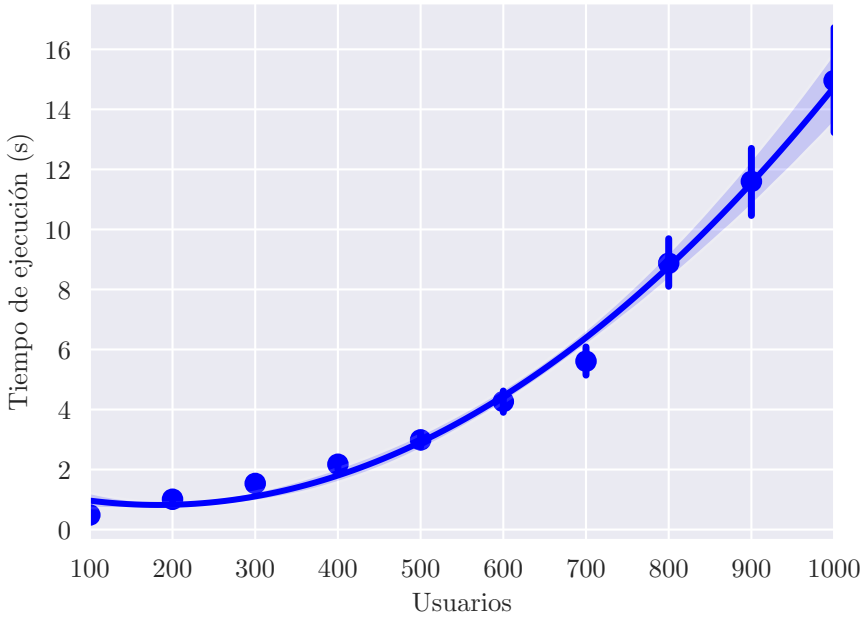

Figura 8: Fit polinómico de orden 2 del experimento 4 (Usuarios)

En las figuras 7 y 8 podemos ver los resultados de una regresión lineal sobre los datos obtenidos en el experimento 4 y un *fit* polinómico de grado 2. Se puede observar que la regresión lineal no modela los datos obtenidos y el valor de $R^2 = 0{,}54$ también lo indica. El fit polinómico se ajusta mejor a los datos aunque los valores residuales para los valores de usuarios entre 100 y 500 son bastante significativos.

6.4.2. Número de servidores (Manteniendo el número de replicaciones)

Tabla 7: Parámetros experimento 4 (Servidores)

Parámetro		valor
USERS		200
NSERV	$50 \rightarrow 1000$	(+50)
NREP		5
REQUESTS		5

Realizamos los experimentos de modo análogo al experimento anterior pero variando el número de servidores desde 50 hasta 1000 en incrementos de 50, los parámetros se muestran en la tabla 7. Se realizaron 100 repeticiones para cada valor del número de servidores, cada uno con una semilla distinta. Se midió el tiempo de ejecución y se calculó la media y la desviación estándar. Los resultados se muestran en la tabla 8:

Tabla 8: Resultados del experimento 4 (Servidores)

NSERV	Media	std
50	0.94622	0.270108
100	1.13820	0.375877
150	1.07949	0.391840
200	1.24774	0.382903
250	1.13636	0.348386
300	1.04336	0.341065
350	1.05570	0.344303
400	0.92419	0.318070
450	0.94206	0.283875
500	0.91905	0.339851

NSERV	Media	std
550	0.92473	0.304312
600	0.87153	0.268109
650	0.89010	0.322946
700	0.83640	0.294466
750	0.79192	0.258547
800	0.80678	0.301987
850	0.78347	0.264913
900	0.70329	0.230744
950	0.70912	0.222591
1000	0.69929	0.224391

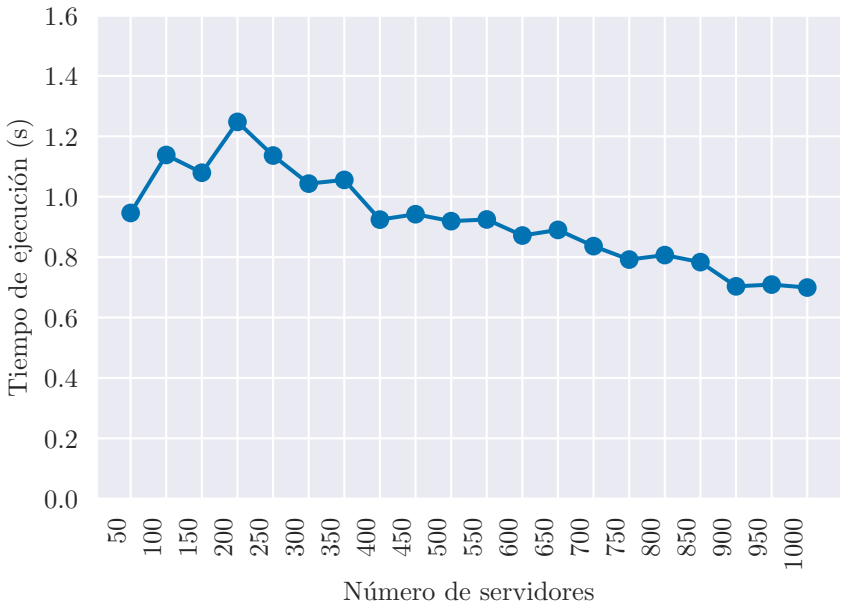

Figura 9: Medias obtenidas en el experimento 4 (Servidores)

En las figuras 9 y 10 se puede observar que el tiempo de ejecución tiende a disminuir al aumentar el número de servidores. No obstante, hay mucha variabilidad en el tiempo de ejecución, por lo que los resultados no son completamente concluyentes.

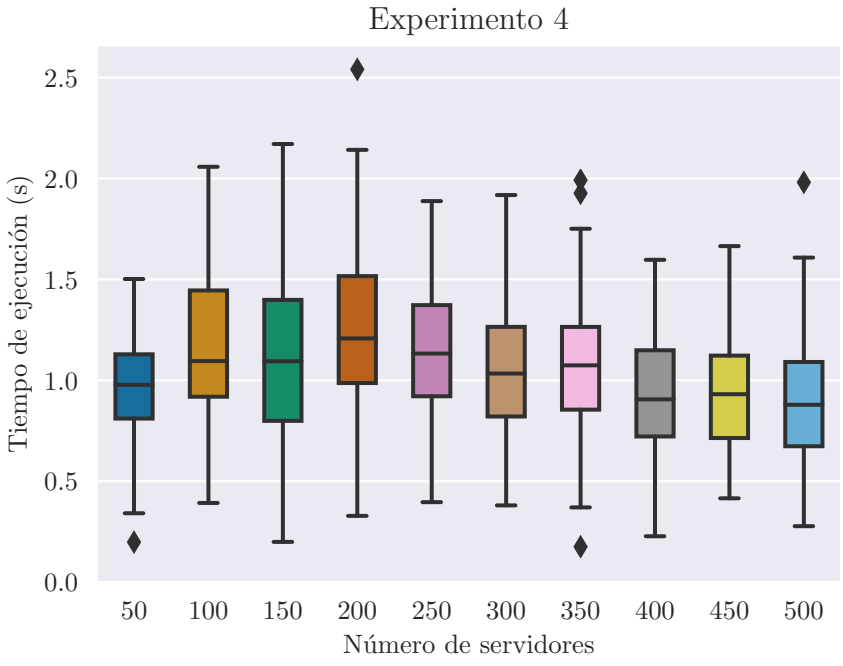

Figura 10: Boxplot resultados experimento 4 ($NSERV \leq 500$)

6.5. Comparación de las dos heurísticas al calcular el Tiempo total de transmisión y tiempo para hallar la solución utilizando Hill Climbing

El objetivo de este experimento es analizar las diferencias entre las dos heurísticas comparando tanto el tiempo total de transmisión como el tiempo de ejecución.

Hipótesis. *La segunda heurística será más rápida y encontrará mejores resultados que la primera.*

Los parámetros usados son los mismos del experimento 1. Se ejecutaron 100 repeticiones usando la primera heurística y 100 más usando la segunda. Los parámetros y resultados obtenidos en el experimento se muestran en la tabla 9.

Tabla 9: Parámetros y resultados del experimento 5

Parámetro	valor
USERS	200
NSERV	50
NREP	5
REQUESTS	5

Heuristica	T_{ej}		TTT	
	media	std	media	std
Max	0.97251	0.300749	543084.94	37647.23
Total	0.63535	0.082964	478000.77	22884.30

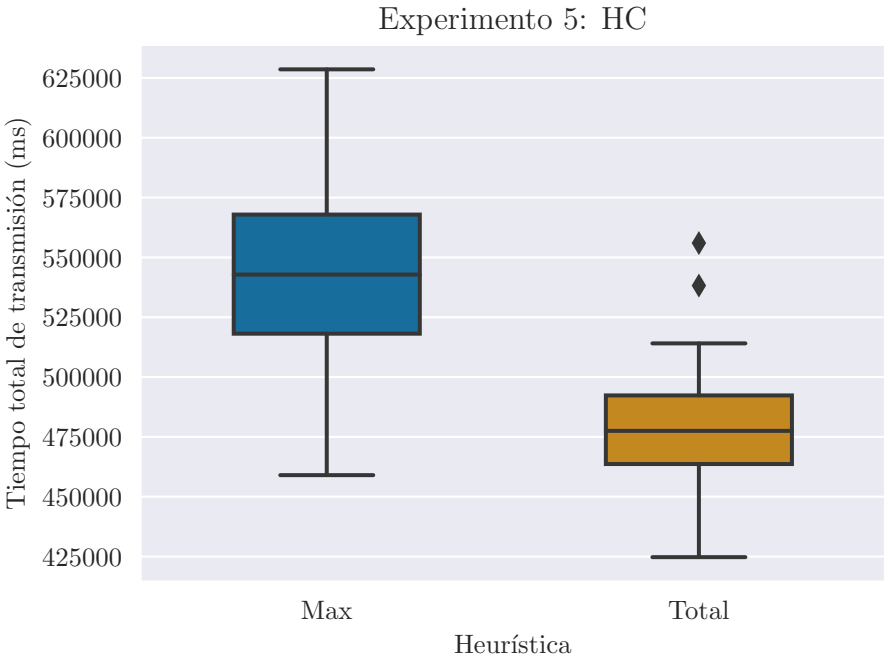

Figura 11: Experimento 5: Tiempo total de transmisión

En la figura 11 podemos ver los boxplots del tiempo de ejecución obtenido en el experimento 5. Podemos ver que el tiempo total de transmisión es menor con la segunda heurística y tiene menos variación.

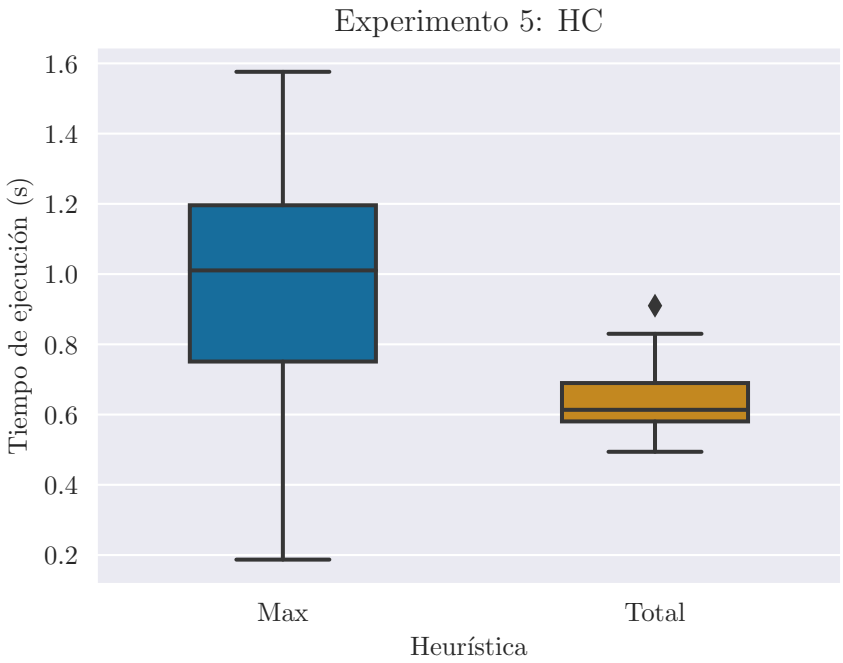

Figura 12: Experimento 5: Tiempo de ejecución

La figura 12 muestra los boxplots del tiempo de ejecución obtenidos en el experimento 5. En esta figura, se puede ver como el tiempo de ejecución es menor con la segunda heurística (Total en la figura) respecto a la primera (Max en la figura). También se puede apreciar como hay una variación de los tiempos de ejecución mucho mayor con la primera heurística que con la segunda.

Los resultados de este experimento nos indican que la segunda heurística es más rápida y encuentra mejores resultados que la primera como esperábamos.

6.6. Comparación de las dos heurísticas al calcular el Tiempo total de transmisión y tiempo para hallar la solución utilizando Simulated Annealing

Este experimento es análogo al anterior (6.5) pero usando Simulated Annealing en vez de Hill Climbing.

Hipótesis. *La segunda heurística será más rápida y encontrará mejores resultados que la primera .*

La tabla 10 muestra los parámetros usados y los resultados obtenidos.

Tabla 10: Parámetros y resultados del experimento 6

Parámetro	valor
USERS	200
NSERV	50
NREP	5
REQUESTS	5

Heuristica	T_{ej}		TTT	
	media	std	media	std
Max	1.74435	0.293189	697912.55	165272.92
Total	1.63954	0.132577	476274.87	23504.87

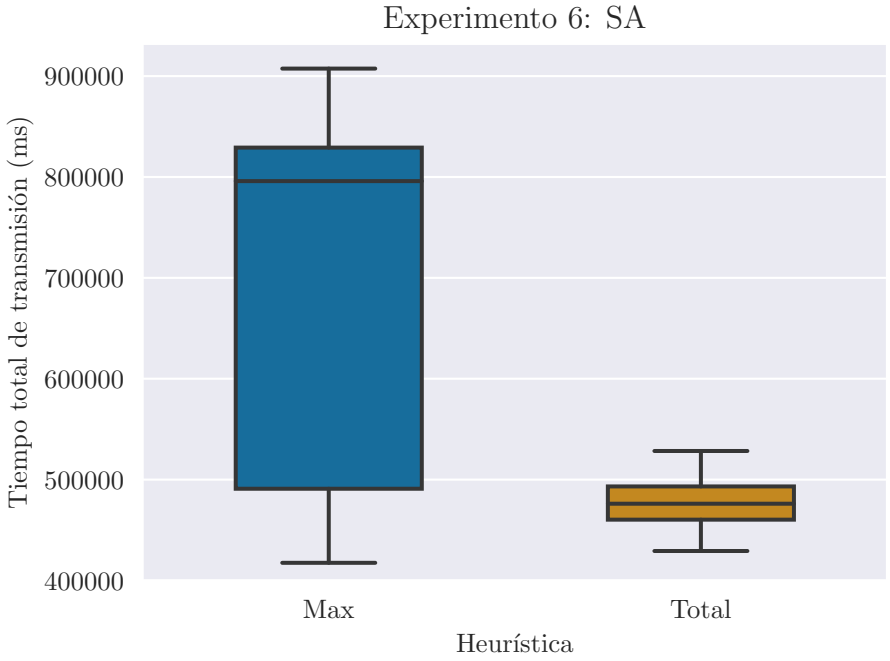

Figura 13: Experimento 6: Tiempo total de transmisión

En la figura 13 podemos observar que, al igual que con Hill Climbing, con Simulated Annealing el tiempo total de transmisión es menor usando la segunda heurística.

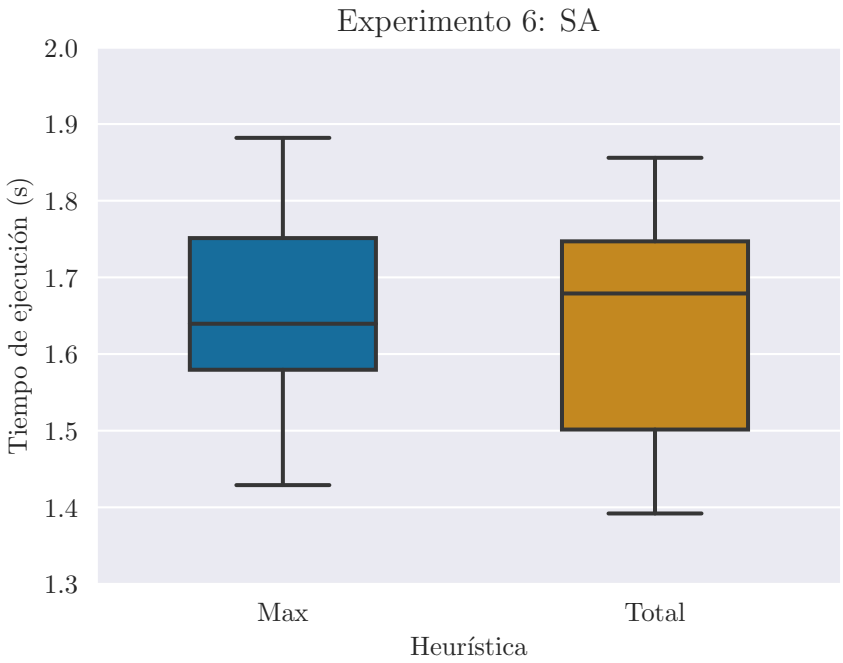

Figura 14: Experimento 6: Tiempo de ejecución

En la figura 14 podemos observar que, al contrario que con Hill Climbing, con Simulated Annealing el tiempo de ejecución es mayor usando la segunda heurística. También varia mucho mas el tiempo de ejecución con la segunda heurística.

6.6.1. Comparación con Hill Climbing

Tabla 11: Resultados del experimento 5 (Hill Climbing)

Heuristica	T_{ej}		TTT	
	media	std	media	std
Max	0.97251	0.300749	543084.94	37647.23
Total	0.63535	0.082964	478000.77	22884.30

Tabla 12: Resultados del experimento 6 (Simulated Annealing)

Heuristica	T_{ej}		TTT	
	media	std	media	std
Max	1.74435	0.293189	697912.55	165272.92
Total	1.63954	0.132577	476274.87	23504.87

Si comparamos las tablas de los experimentos 5 y 6, podemos observar que Simulated Annealing obtiene resultados similares a Hill Climbing pero tarda bastante más. En los dos casos la heurística que produce mejor TTT es la segunda.

6.7. Influencia del número de replicaciones de los ficheros

Tabla 13: Parámetros experimento 7

Parámetro		valor
USERS		200
NSERV		50
NREP	$5 \rightarrow 25$	(+5)
REQUESTS		5

Para este experimento variamos el número de repeticiones de usuarios de 5 a 25 en incrementos de 5. Se realizaron 100 repeticiones para cada caso. Además, realizamos dos experimentos distintos, uno para cada heurística. En las siguientes secciones se muestran y analizan con detalle los resultados obtenidos para cada heurística y se hace una comparación.

Hipótesis. *Un mayor número de replicaciones implica un espacio de búsqueda más grande, por lo que el tiempo de ejecución aumentará. Así mismo tener más replicaciones hará que los servidores puedan distribuir mejor sus cargas por lo que obtendremos tiempos totales de transmisión menores.*

6.7.1. Primera heurística

Tabla 14: Resultados del experimento 7 con la primera heurística

NREP	T_{ej}		TTT	
	Media	std	Media	std
5	1.01077	0.332965	541711.65	34079.55
10	1.43681	0.480345	342630.67	20683.48
15	1.96248	0.807076	260552.10	19601.19
20	2.16906	0.928984	215671.28	16023.59
25	2.62371	1.245368	191069.57	14962.04

La tabla 14 muestra los resultados obtenidos en el experimento 7 con la primera heurística. Los datos se muestran en el gráfico de la figura 15 en el que se puede apreciar como el tiempo de ejecución incrementa linealmente con el número de replicaciones y el tiempo total de transmisión disminuye. [3].

[3]Notase que para poder visualizar bien las pendientes, los ejes verticales no empiezan en 0

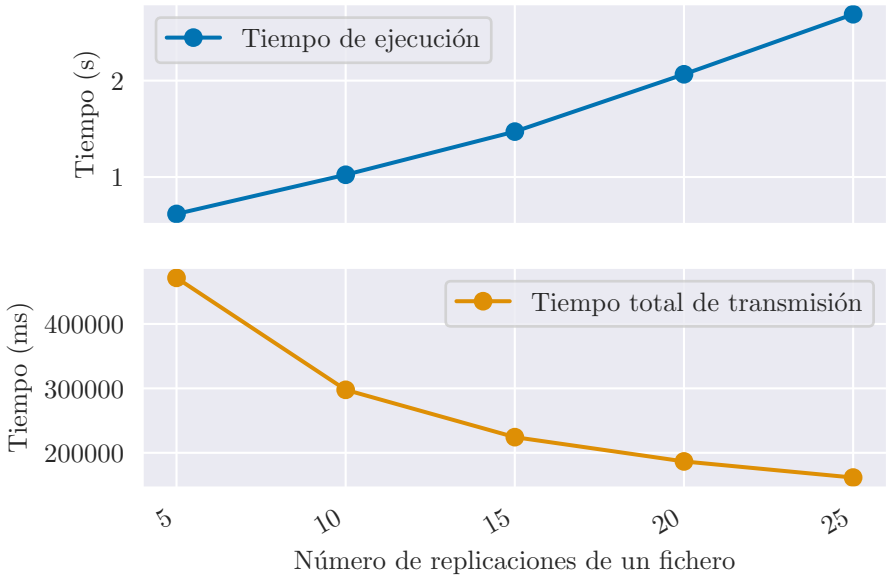

Figura 15: Medias del experimento 7 con la primera heurística

El boxplot de la figura 16 nos permite analizar con más detalle los resultados obtenidos. Vemos que hay mucha variación en el tiempo de ejecución con outliers que llegan hasta 3 veces más de la media.

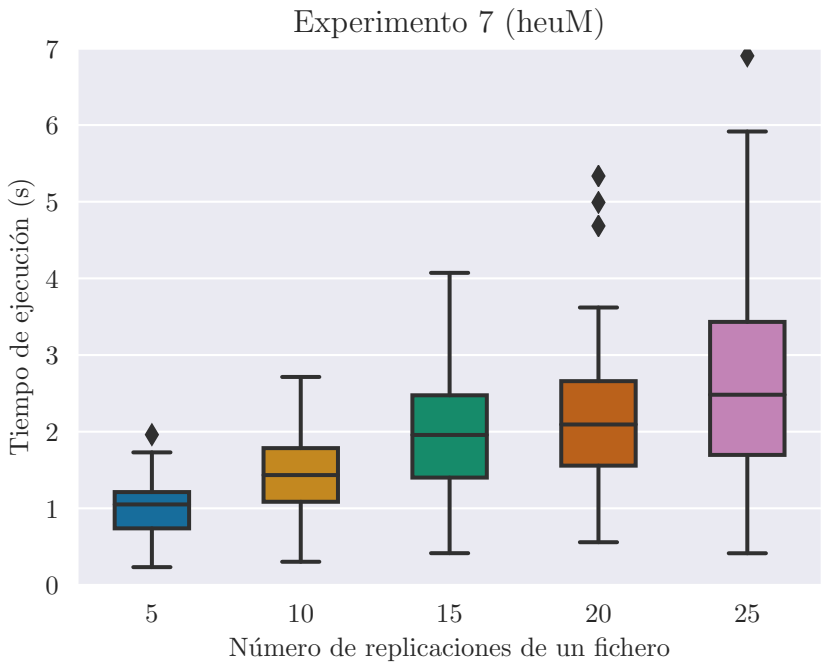

Figura 16: Boxplot del tiempo de ejecución con la primera heurística

La figura 17 muestra la regresión lineal sobre los datos obtenidos para el tiempo de ejecución con la primera heurística. A pesar de lo que parecía a priori, la regresión tiene un valor de R^2 muy bajo que nos indica que no es un buen modelo.

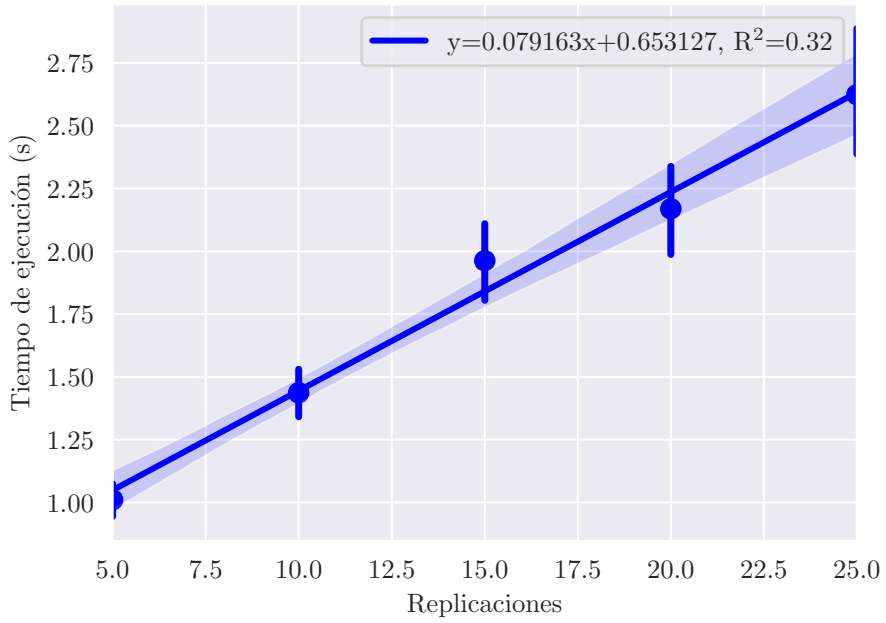

Figura 17: Regresión del tiempo de ejecución con la primera heurística

El boxplot del tiempo total de transmisión que se muestra en la figura 18 nos indica que el número de replicaciones de un fichero y el tiempo total de transmisión de la solución obtenida sigue una relación de proporcionalidad inversa. A diferencia de con el tiempo de ejecución el tiempo total obtenido parece muy consistente con poca variación entre las 100 repeticiones realizadas.

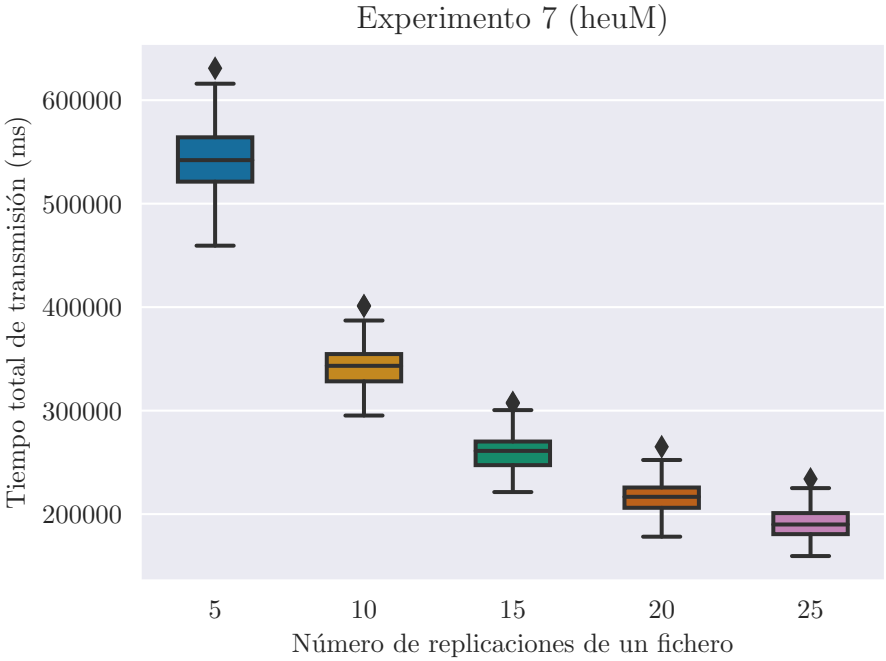

Figura 18: Boxplot del tiempo total de transmisión con la primera heurística

6.7.2. Segunda heurística

Los resultados con la segunda heurística son similares a los obtenidos con la primera. A continuación se incluyen las tablas y gráficos correspondientes.

Tabla 15: Resultados del experimento 7 con la segunda heurística

NREP	T_{ej}		TTT	
	Media	std	Media	std
5	0.61688	0.126308	471577.81	22913.49
10	1.02330	0.122715	297764.73	12791.75
15	1.47118	0.195260	224288.30	10553.78
20	2.06579	0.325079	186614.04	8792.76
25	2.68870	0.414751	161579.74	8144.09

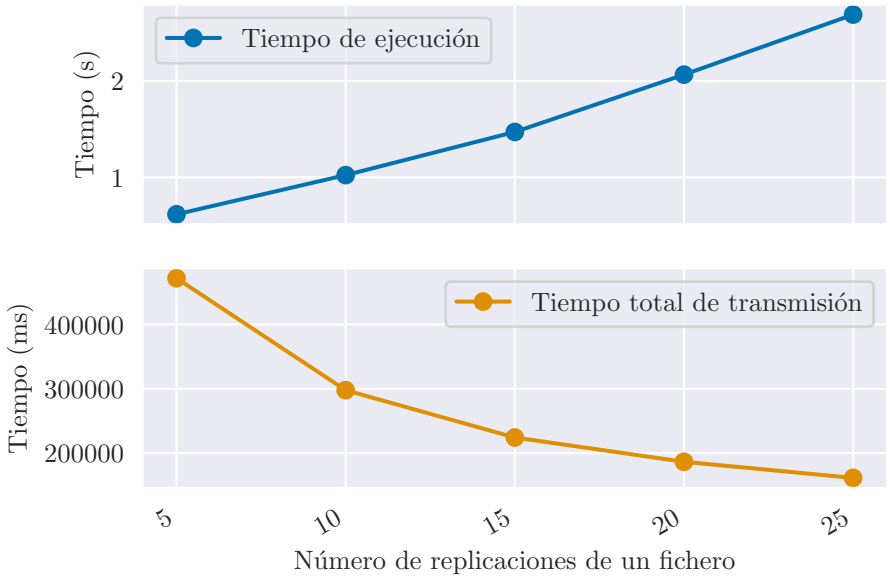

Figura 19: Medias del experimento 7 con la segunda heurística

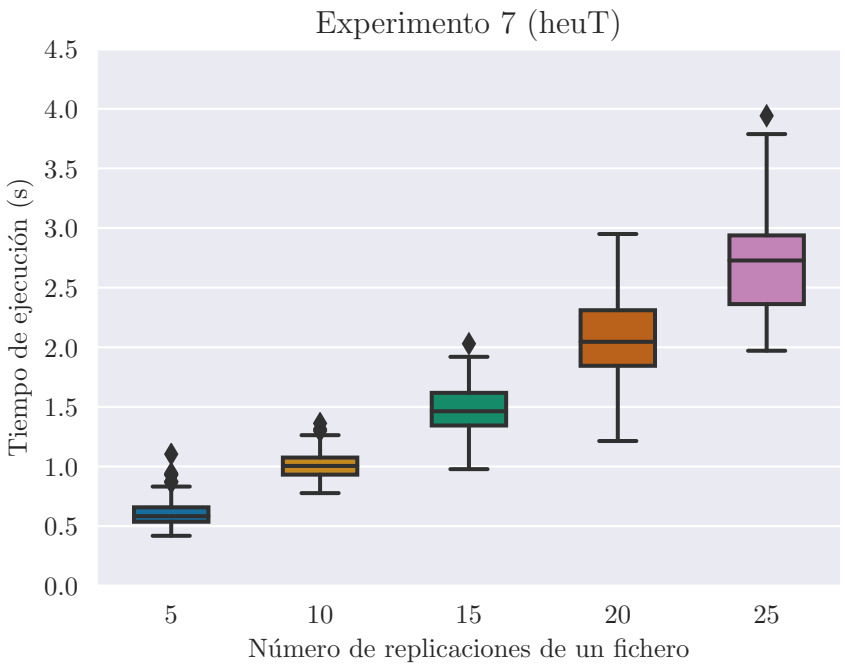

Figura 20: Boxplot del tiempo de ejecución con la segunda heurística

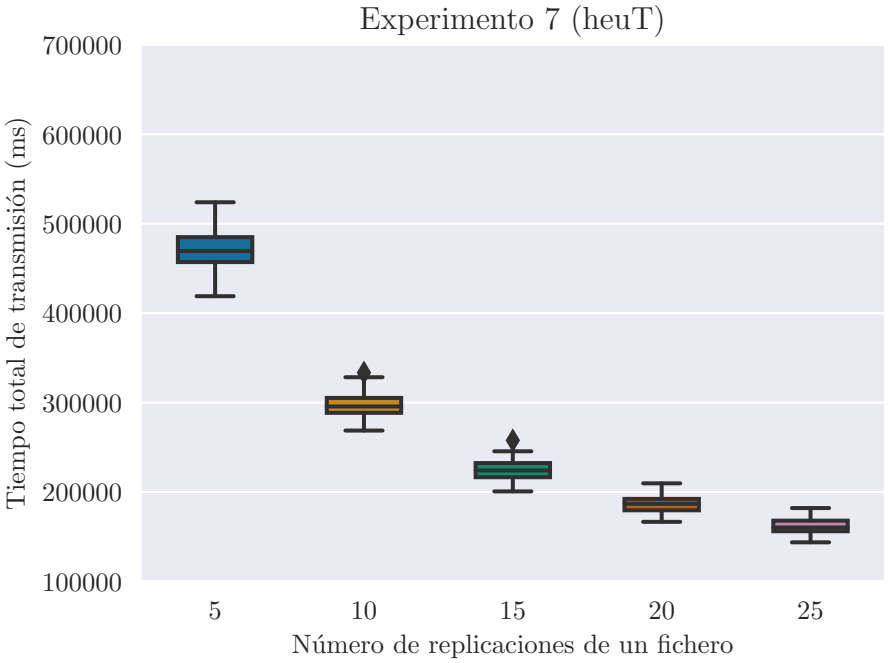

Figura 21: Boxplot del tiempo total de transmisión con la segunda heurística

6.8. Comparación de las heurísticas

Como se ha podido ver en el experimento 7 (sección 6.7, a nivel del tiempo de ejecución las dos heurística obtienen resultados similares, aunque la primera heurística tiene una desviación estándar mayor (los tiempos tienden a variar más).

Donde si se aprecia una diferencia notable es en el tiempo total de transmisión. Los resultados con la segunda heurística son consistentemente más bajos que con la primera tal y como se muestra en la figura 22:

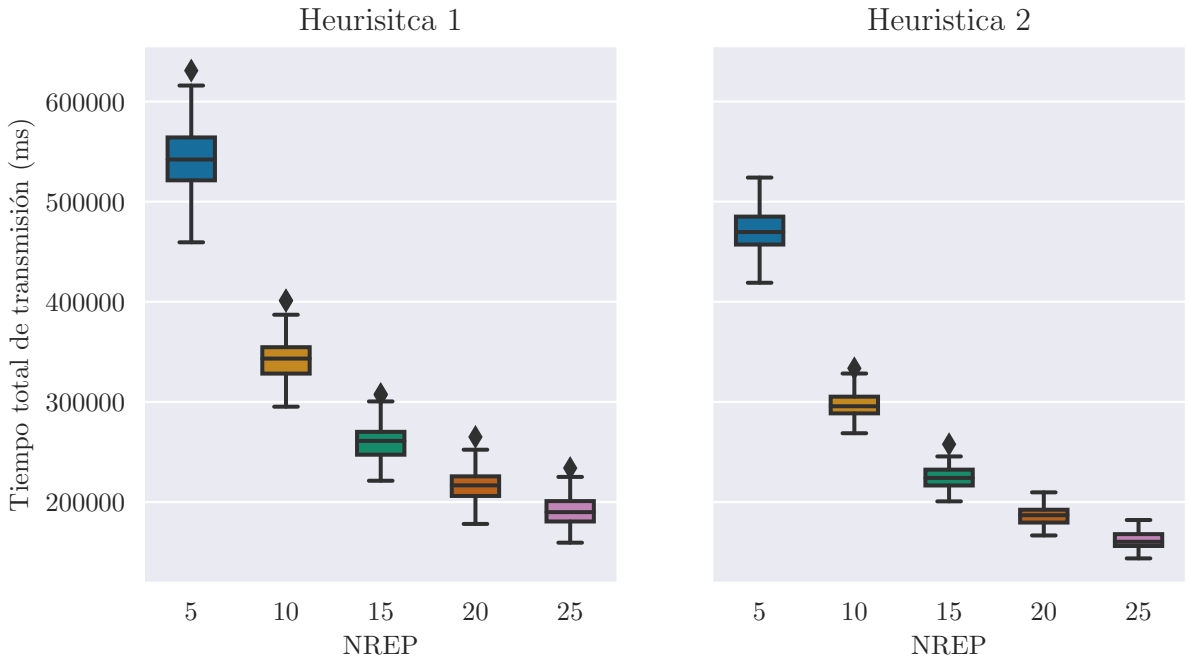

Figura 22: Boxplots del tiempo total de transmisión con de las dos heurísticas

Los resultados de los experimentos 5 y 6 (secciones 6.5 y 6.6) nos muestran que la segunda heurística da mejores resultados Usando tanto Hill Climbing como Simulated Annealing.

7. Conclusión

Los resultados de los experimentos 5 y 6 (secciones 6.5 y 6.6) que comparan las distintas heurísticas con Hill Climbing y Simulated Annealing nos indican que con la segunda heurística se obtiene un mejor resultado usando Simulated Annealing, pero con la primera se obtiene uno mejor con Hill Climbing. Esto no concuerda con lo que esperábamos inicialmente: que el Simulated Annealing obtuviera resultados significativamente mejores que Hill Climbing con ambas heurísticas.