UNIVERSITAT POLITÉCNICA DE CATALUNYA

INTELIGENCIA ARTIFICIAL

BACHELOR DEGREE IN COMPUTER SCIENCE

PROYECTO DE BÚSQUEDA LOCAL

Authors:
Daniel Moreno
Fabio De Angelis
Solange Palomino

Professor:
Javier Vazquez Salceda

Q1 Curso 2019-2020

Índice

Identificación del problema

Hoy en día el flujo de datos en los negocios es un aspecto muy importante a tomar en cuenta cuando se está en camino al desarrollo de aplicaciones en los que se requiere la interacción directa y fluida con los clientes. Por ejemplo, en caso una entidad financiera necesite que miles de usuarios ingresen al mismo tiempo a su plataforma de consulta de saldos, por detrás, lo que se requiere es que todas las consultas sean atendidas en el menor tiempo posible, lo que implica que miles de consultas se ingresen simultáneamente al mismo servidor de base de datos. Sin embargo, llegará un momento en el que este no pueda atender todas las consultas, al menos no prometiendo el mejor tiempo de respuesta, el cual también es un factor relevante para asegurar la fidelidad del cliente.

Una manera de sobrellevar esta situación es la implementación de un servicio de replicación de la base de datos. Esta técnica consiste en mantener copias de la misma información en varios servidores de bases de datos. Además, se cuenta con un servidor inicial que se encarga de recepcionar las consultas para luego redirigirlas al servidor más cercano que pueda atender la petición. El tiempo de respuesta depende de la cercanía entre el servidor que atiende y el usuario. Por otro lado, el servidor inicial debe buscar una estrategia de tal forma que todas las consultas no sobrecarguen a un solo servidor.

La replicación es una técnica que se utiliza para tener garantizada la accesibilidad, la seguridad, la fiabilidad y disponibilidad de la información, de forma que si ocurre un fallo en un servidor de base de datos, habrán otros servidores que tendrán la misma información para que los usuarios puedan seguir usando la aplicación sin ningún problema. Es por ello que se toma muy en cuenta en los estudios sobre Continuidad de Negocios dentro del campo de la Informática.

Una situación similar es la que se plantea en este proyecto, pero en relación a otro tipo de sistema: un sistema de ficheros. Sin embargo, en este nuevo contexto los ficheros no están replicados en todos los otros servidores, sino que esto depende de la tasa de replicación que tenga cada uno de los ficheros. Asimismo, la relación es de muchos a muchos: un mismo usuario puede solicitar muchos ficheros y un mismo fichero puede ser solicitado por muchos usuarios. A la vez, ese fichero puede estar en muchos servidores (la tasa de replicación de cada fichero no debe superar el 50%).

El problema a resolver trata sobre determinar qué servidor atenderá cada una de las solicitudes de una serie de usuarios (<IDUsuario1, IDFichero2>,<IDUsuario2, IDFichero5>,<IDUsuario1, IDFichero7>), de tal forma que todas sean atendidas y que no todas sean atendidas por un sólo servidor.

Como ya se ha mencionado, la distancia entre los servidores y el usuario marca diferencia en el tiempo de respuesta, lo que justamente debemos minimizar en este tipo de problemas.

Cabe mencionar que el ejemplo de los servidores de bases de datos de la compañía bancaria es una muestra de cuán aplicables son este tipo de problemas en la vida real.

Elementos del problema

Así como existe la replicación para que los servidores que almacenan los ficheros no se sobrecarguen, también podemos utilizar muchos servidores "iniciales" que recepcionen las consultas. Sin embargo, para este problema asumiremos que estamos solucionando la problemática sólo para uno de ellos.

En el mismo sentido, los ficheros están distribuidos en un grupo de servidores de ficheros, los cuales están en distintas ubicaciones físicas. Cada uno de los ficheros puede estar en muchos servidores al mismo tiempo, esto depende de la tasa mínima de replicación que se ingrese como dato al problema. Lo cual determina el número mínimo de copias que tendrá cada uno de los ficheros. Sin embargo, esta tasa de replicación no debe superar el 50% del número de servidores de ficheros que hay en el sistema.

La esencia del problema ronda principalmente sobre el tiempo de respuesta, ya que dos servidores ubicados en la zona A y en la zona B, respectivamente, pueden tener una copia del fichero que solicita el usuario X. No obstante, el servidor de la zona A podría ser la mejor opción ya que este se encuentra más cerca al ordenador del usuario que lo solicitó, por lo que el tiempo de respuesta sería menor. Esto se debe mantener para toda la serie de solicitudes que reciba el servidor de distribución, considerando que los ficheros se transmiten de forma secuencial hacia los clientes.

Cabe mencionar que el servidor de distribución cuenta con información crucial para el problema como son los siguientes:

- La ubicación de un fichero dentro del sistema de servidores de ficheros, es decir, qué servidores tienen una copia del fichero.
- El tiempo de respuesta de cada servidor hacia el dispositivo del usuario, en milisegundos y dentro de un rango de 100-5000 ms.

Búsqueda local

La búsqueda local es uno de los muchos métodos que existen para solucionar problemas de optimización. Este tipo de problemas tienen como característica principal la necesidad de minimizar o maximizar el valor de una variable o función. Gráficamente podemos usar las derivadas para hallar mínimos y máximos en una curva; sin embargo esto tiene un límite: el plano cartesiano sólo tiene 3 dimensiones, con lo que nos vemos restringidos al manejo de sólo tres variables en nuestro problema y a una función que sólo dependa de una.

El método de la búsqueda local nos lleva a otro enfoque del problema. Se trata de empezar en una solución, denominado como estado inicial, y de evaluar a sus vecinos dentro del espacio de soluciones. En caso uno de ellos signifique una mejor solución, se reemplaza el estado inicial por el nuevo, y se continúa con el proceso, hasta que no haya posibilidad de mejorar más, es decir cuando se haya llegado a un máximo local (o mínimo local). La solución ideal es encontrar el máximo global (o mínimo global), no obstante, puede suceder que existan muchos picos dentro del espacio de estados pero que el más cercano no sea necesariamente el mejor, lo cual depende básicamente de donde esté ubicada la solución inicial.

El problema de los ficheros se encuentra dentro de los problemas de búsqueda local ya que se intenta minimizar el tiempo de respuesta de toda la lista de solicitudes de ficheros hacia los usuarios que se envía al servidor de distribución, además del tiempo en el que obtenemos ese valor. Por lo que tenemos un estado con muchas variables, para el cual debemos analizar una serie de posibilidades teniendo en cuenta ciertas heurísticas tal que encontremos una mejor situación, es decir, una mejor solución.

Asimismo, no iremos construyendo un camino a cada paso que demos, sino que comenzaremos con una solución y lo iremos modificando en torno a que encontremos uno mejor. Además que el camino por el que lleguemos a la solución final no es muy importante para nosotros, sino la solución misma.

Descripciones y justificaciones

En el siguiente apartado se detallarán las descripciones de cada uno de los parámetros solicitados para la implementación del problema, así como los criterios que tomamos en cuenta para poder elegirlos como los más indicados para nuestro proyecto.

Implementación del estado

Un estado es la representación de los elementos que describen el problema en un momento específico. Es primordial definirlo en el problema ya que con ello podremos determinar nuestro espacio de soluciones, así como el estado final que representa nuestro objetivo.

En este proyecto decidimos definir a un estado como una clase que contiene tres atributos:

- Servidores: Un arreglo que reúne la información de todos lo servidores que constituyen el sistema completo de almacenamiento de ficheros. Este es un arreglo estático ya que las consultas son información que no debe variar en el tiempo y le pertenece a todos los estados.

- Consultas: Un arreglo con la información de cada una de las solicitudes de ficheros que recibe el servidor de distribución. Este es un arreglo estático ya que las consultas son información que no debe variar en el tiempo y le pertenece a todos los estados.

- Peticiones: Un arreglo de enteros en los que se almacenará el identificador del servidor que atenderá la serie de solicitudes de ficheros.

  Escogimos este tipo de dato ya que nos da la libertad de no requerir almacenar un objeto más complejo que almacene una relación de <Servidor, Fichero>, sino que podemos relacionarlo directamente con los índices. Así, el servidor que esté en la posición i en nuestro arreglo, será el que atienda la petición del fichero en el arreglo de Consultas en la posición i, y así para todos.

  Asimismo, al no tener que crear un objeto de <Servidor, Fichero> y almacenarlo en una lista o en un arreglo, ahorraremos espacio en nuestra memoria, el cual podríamos utilizarlo de forma más eficiente para crear más nodos hijos dentro del problema y encontrar una mejor solución.

Por otro lado, ¿cómo se determina si un estado es finalmente el estado objetivo? En este caso, no podemos establecer condiciones específicas puesto que no conocemos del todo al estado final al que llegaremos. Sólo debemos tener en cuenta que el estado siguiente debe ser mejor respecto al actual, de acuerdo a las heurísticas que estamos utilizando. En caso sea así, entonces actualizaremos el estado, pero en caso negativo, o analizaremos más vecinos o nos quedaremos con ese último como el máximo (o mínimo) local.

Respecto al espacio de búsqueda podemos mencionar que toma un valor significativamente alto: CxS (Cada una de las consultas recibidas por el servidor de distribución x Número de servidores que almacenan el fichero solicitado).

Operadores elegidos

Para poder navegar dentro del espacio de soluciones y transformar nuestro estado actual en uno mejor necesitamos definir una serie de operadores que creamos convenientes. Dentro del proyecto hemos escogido implementar sólo dos operadores:

- SwapServer: Este operador intercambia los servidores que atenderán a dos ficheros distintos, siempre que los servidores a intercambiar estén dentro del conjunto de servidores que pueden atenderlos. Este fichero requiere de dos pares e intercambia el servidor que atiende la petición i para toda la lista de ficheros solicitados, con uno de la posición j elegido aleatoriamente.

  En un primer momento decidimos intercambiar cada uno de los servidores en posición i con otro de la posición i+1 (i+1 mod tamaño de peticiones, para no excluir del intercambio al último servidor); sin embargo, esto no ayudaba a la generación de nuevos hijos ni a la de nuevos movimientos para llegar a un óptimo ya que sólo se comparaba con el del costado. Fue por ello que consideramos usar un aleatorio.

  El factor de ramificación es menor o igual al tamaño de la lista de peticiones de ficheros ya que no siempre se pueden intercambiar dos servidores

- cambiar_servidor: Este operador es una función que retorna una lista de objetos de la clase Estado. Este recorre la lista de servidores que van a atender cada una de las peticiones y los cambia por los siguientes de la lista de cada fichero.

  Podemos decir que el factor de ramificación es mediano ya que depende de la cantidad de solicitudes que se recepcionen, y por cada uno de ellos elegirá el siguiente servidor que pueda atender la petición.

Estrategias para la solución inicial

El estado inicial es donde se empieza las búsqueda de las soluciones y es primordial para determinar en qué zona del espacio de soluciones caemos y hacia qué máximo (o mínimo) nos podemos estamos más próximos.

En el presente proyecto planteamos dos estrategias para generar esta solución inicial, claramente uno mejor que el otro y son los siguientes.

- Primera solución: En esta primera generación se escoge como respuesta al primer servidor de la lista de servidores donde se encuentra el fichero.

  Consideramos este como la función que nos da un peor inicio frente al problema ya que cogemos el primero de la lista de servidores posibles para cada una de las peticiones, pero queremos experimentar si realmente sucede lo que esperamos. Sin embargo podemos decir que su costo es mucho menor que de la siguiente función de generación.

- Segunda solución: Para cada petición se selecciona el servidor con menor tiempo de respuesta dentro de la primera mitad de la lista de servidores donde se encuentra el fichero.

  Ya que en esta función se requiere un recorrido de al menos la mitad de la lista de posibles servidores que pueden atender la petición, el coste es mayor al de la primera función de generación planteada. Sin embargo, consideramos que esta solución es mucho más favorecedora ya que nos da un mejor comienzo teniendo en cuenta que estamos cogiendo uno de los mejores del conjunto total de servidores.

  Llegamos a la conclusión de lo anterior con la premisa de que, si escogemos al mejor de la primera mitad, es muy posible que también sea el mejor de la segunda mitad o al menos que sea uno de los mejores.

Funciones heurísticas

Las funciones heurísticas que hemos utilizado van acorde a la estrategia planteada para el problema:

- Heurística 1: Esta función acumula el tiempo que se demora en responder cada uno de los servidores de la lista de peticiones del estado que se está analizando. Con ello, elegimos el valor del tiempo total que se demora el servidor más lento y lo retornamos. El objetivo es minimizar este valor.

- Heurística 2: Esta función devuelve el tiempo total de transmisión de los servidores penalizando el hecho de que el tiempo de transmisión por cada servidor difiera mucho con los demás. Lo conseguimos calculando la media de tiempo de transmisión y si se desvía bastante sumamos una penalización de 5000 al tiempo total.

Experimentación

El siguiente apartado está destinado a la presentación detallada de cada uno de los experimentos realizados en el proyecto, donde se pone a prueba cada uno de los ítems desarrollados en torno a la aplicación de las estrategias de búsqueda.

Experimento 1

Observación

Para este proyecto hemos determinado la aplicación de dos operadores que nos ayuden a movernos dentro del espacio de búsqueda para encontrar la mejor solución. Sin embargo, debemos entender su comportamiento de forma individual como en equipo para poder determinar cuál es el que nos aporta más en la búsqueda del óptimo, respecto a la heurística que optimice el primer criterio. A partir de estos resultados fijaremos los operadores para el resto de experimentos.

Planteamiento del problema

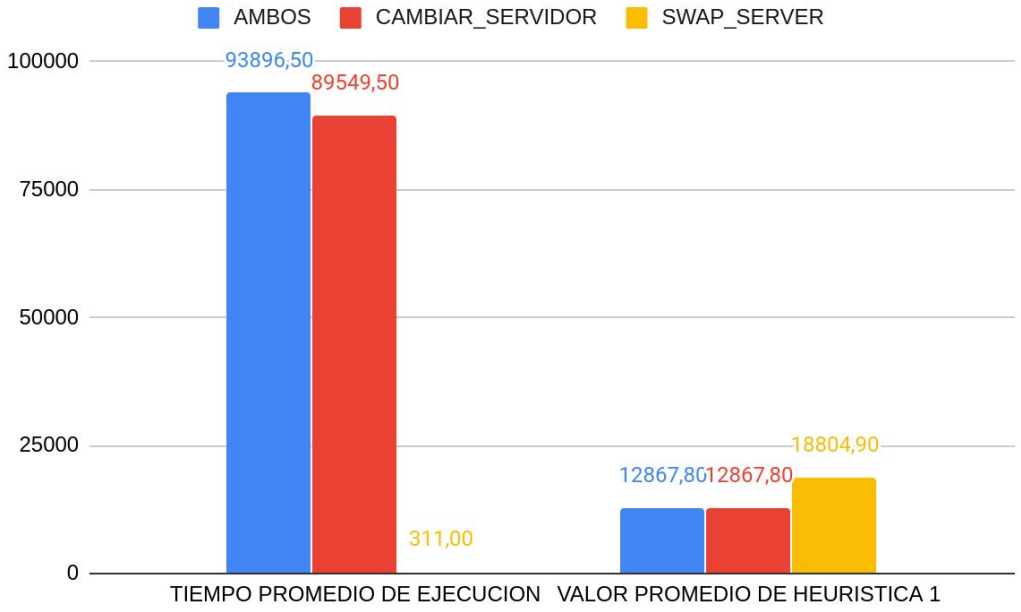
Ejecutamos el programa desarrollado con los parámetros correspondientes usando sólo uno de los operadores (o ambos) y observamos el estado final, el valor de la heurística N° 1 y el tiempo de ejecución.

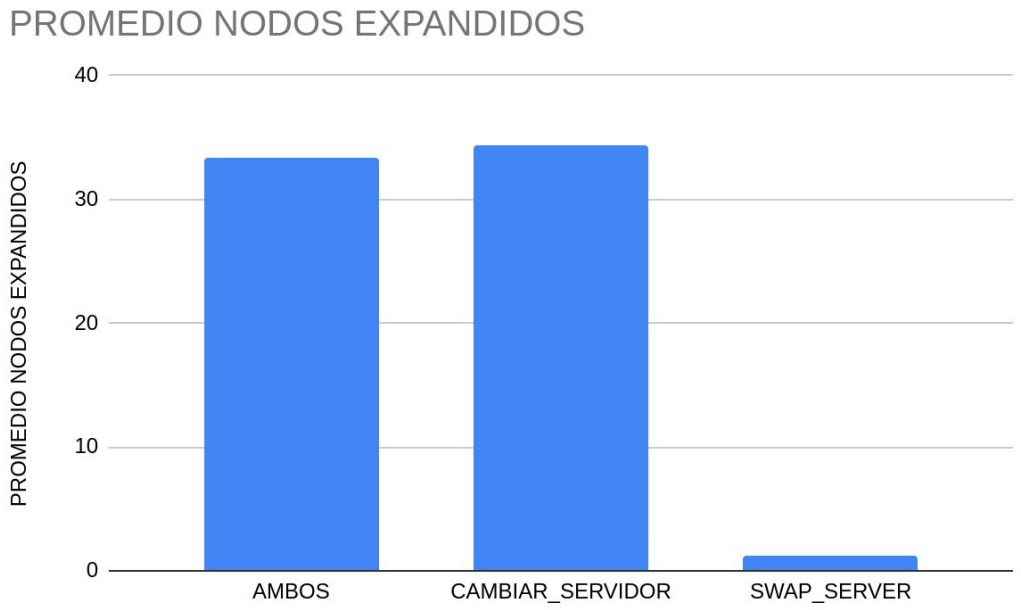
Hipótesis

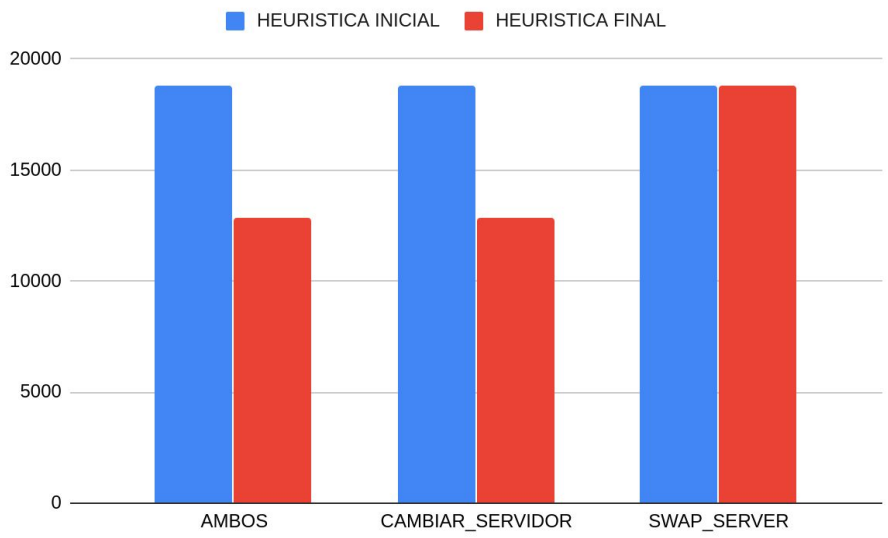
El trabajo conjunto de las dos funciones de cambiar_servidor y swap_Server será la configuración que mejor resultados nos brinde.

Método

- Estableceremos un escenario del problema principal 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. Esta configuración será la misma para todas las réplicas.
- Usaremos el algoritmo Hill climbing.
- Aplicaremos la estrategia de generación del estado inicial N° 2.
- Estableceremos a la heurística N° 1 como la heurística a usar en el problema.
- Elegiremos una de las dos funciones o el trabajo paralelo de ambas dentro de la función la generación de sucesores.
- Ejecutaremos 10 experimentos para cada función escogida (o ambas).
- Mediremos diferentes parámetros para realizar la comparación.


AMBOS
CAMBIAR_SERVIDOR
SWAP_SERVER
100000
75000
50000
25000
0
93896,50
89549,50
311,00
12867,80
12867,80
18804,90
TIEMPO PROMEDIO DE EJECUCION
VALOR PROMEDIO DE HEURISTICA 1


PROMEDIO NODOS EXPANDIDOS
40
30
20
10
0
PROMEDIO NODOS EXPANDIDOS
AMBOS
CAMBIAR_SERVIDOR
SWAP_SERVER



Discusión

Se esperaba que los valores del tiempo de ejecución aumenten al usar ambos operadores en paralelo, lo cual es precisamente lo que se observa en los resultados. Lo curioso de ello es que la mayor fracción de ese tiempo es realmente lo utilizado por el operador cambiar_servidor.

Esto podría significar dos cosas, o el operador Swap_server es mucho más eficiente por lo que demora menos en dar una respuesta, o demora menos porque no da muchas alternativas para que podamos movernos dentro del espacio de soluciones del problema. Lo que observamos por el número de nodos expandidos es que este operador en realidad no nos está ayudando mucho a generar nuevos nodos hijos, ya que la cantidad es considerablemente menor.

Por otro lado, si analizamos el valor de las heurísticas obtenido en el estado final de los nodos generados, el operador cambiar_servidor ofrece una mejor actuación respecto a Swap_server.

En general, Swap_server no ha disminuido en nada a la heurística que estábamos evaluando. Esto se podría deber a que lo que el operador hace es intercambiar el servidor que tiene asignado un fichero i por el que tiene otro j si es que ambos comparten una serie de servidores en común. Al parecer eso ha reducido a casi nula la posibilidad de intercambiar el servidor y con ello generar sucesores.

Conclusiones

A pesar de que el tiempo de ejecución es mucho más grande, el operador cambiar_servidor tiene un factor de ramificación mucho más grande y nos ofrece muchas más alternativas para la creación de nodos hijos. Por tanto, este será el operador que utilizaremos para los siguientes experimentos.

Experimento 2

Observación

Los algoritmos de búsqueda local son los que nos ayudan a buscar una solución óptima a partir de la transformación de un estado determinado hacia otro dentro del espacio de soluciones. Sin embargo, para tratar de garantizar obtener una buena solución final es necesario partir de un buen estado inicial, por lo que requerimos determinar qué estrategia de las dos que planteamos en el presente proyecto es la que da mejores resultados.

Planteamiento del problema

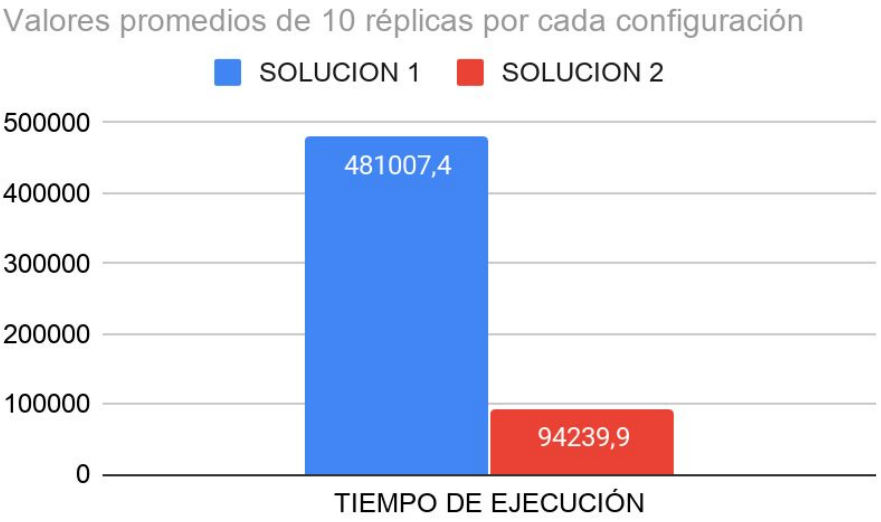
Ejecutamos el programa desarrollado con los parámetros correspondientes usando cada una de las funciones generadores de estado inicial y observamos el estado final, el valor de la heurística N° 1 y el tiempo de ejecución.

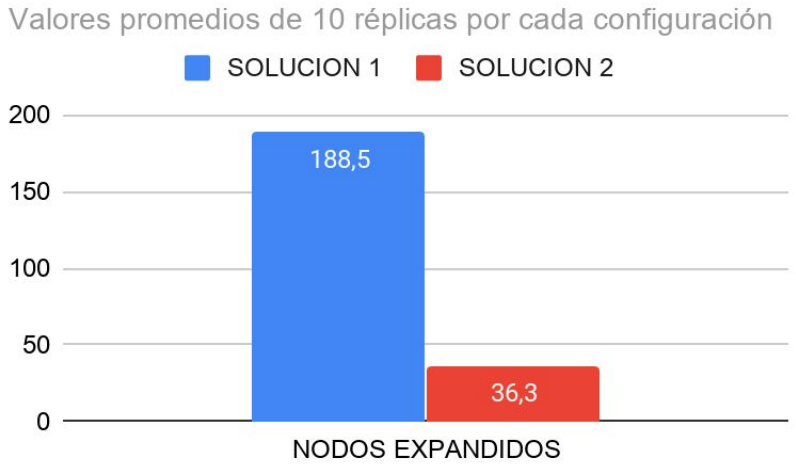
Hipótesis

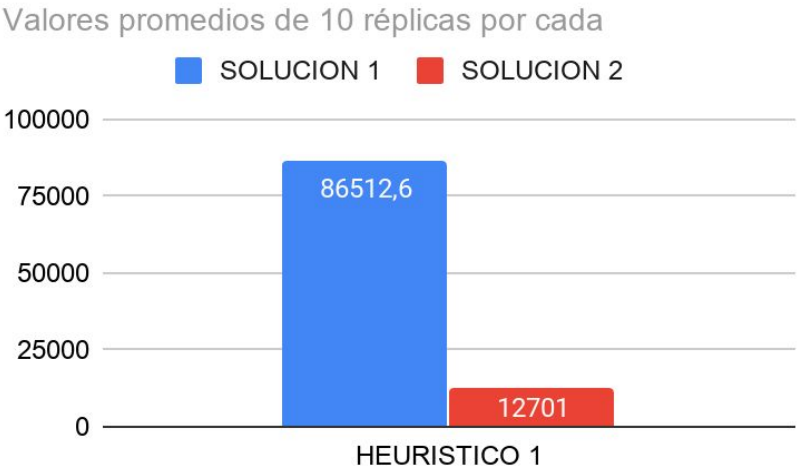
La segunda función generadora de estado inicial mostrará una mejor perfomance en cuanto a la mejora de los heurísticos del problema planteado.

Método

- Estableceremos un escenario del problema principal 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. Esta configuración será la misma para todas las réplicas.
- Usaremos el algoritmo Hill climbing.
- Estableceremos a la heurística N° 1 como la heurística a usar en el problema.
- Elegiremos sólo la función cambiar_servidor para la generación de sucesores.
- Probaremos las dos funciones de generación de estado inicial.
- Ejecutaremos 10 experimentos para cada función escogida.
- Mediremos diferentes parámetros para realizar la comparación.


Valores promedios de 10 réplicas por cada configuración
SOLUCION 1
SOLUCION 2
500000
400000
300000
200000
100000
0
481007,4
94239,9
TIEMPO DE EJECUCIÓN


Valores promedios de 10 réplicas por cada configuración
SOLUCION 1
SOLUCION 2
200
150
100
50
0
188,5
36,3
NODOS EXPANDIDOS


Valores promedios de 10 réplicas por cada
SOLUCION 1
SOLUCION 2
100000
75000
50000
25000
0
86512,6
12701
HEURISTICO 1

Discusión

Como se observa en los gráficos donde se muestra el promedio de los valores obtenidos en todas repeticiones de los experimentos, la función generadora de la solución inicial 1 es la que en promedio más consume en el problema si hablamos respecto al tiempo de ejecución. Asimismo, está función es la que da más probabilidad de nodos expandidos, pero debido al heurístico del estado inicial no hemos avanzado mucho en el camino a optimizar nuestra solución.

En cambio, la segunda función generadora presenta otra situación un poco más favorable a nuestro problema ya que buscamos tener un gran avance hacia la solución más óptima y, al mismo tiempo, lograrlo en poco tiempo; y es esta función la que, según las experiencias, las está cumpliendo.

Conclusiones

Podemos concluir que no hay mucho que comparar entre estas dos funciones al observar su gran diferencia. Una es mejor que la otra directamente tal y como se había propuesto en la hipótesis del experimento, por tanto, fijaremos a la función N° 2 como la estrategia de generación de la solución inicial para los siguientes experimentos.

Experimento 3

Observación

Los problemas de búsqueda local tienen una amplia variedad de algoritmos a aplicar para poder solucionarlos. Uno de ellos es Simulated Annealing que se basa en el método de enfriamiento de los metales y básicamente es una combinación entre Hill Climbing y el seguimiento de un camino aleatorio, con el que ganamos eficiencia y completitud. Sin embargo, se requiere determinar el valor de sus cuatro parámetros que son variables a cada problema y son los siguientes: El número total de iteraciones, número de iteraciones por cada cambio de temperatura (ha de ser un divisor del anterior), parámetro k de la función de aceptación de estados, parámetro λ de la función de aceptación de estados

Planteamiento del problema

Ejecutamos el programa desarrollado con la configuración del escenario correspondiente variando cada vez los parámetros del algoritmo y observamos el estado final, el valor de la heurística N° 1 y el tiempo de ejecución para determinar la influencia de los cuatro factores dentro de la ejecución del algoritmo. Nos aseguramos de que el número de iteraciones que usaremos sea suficiente para que la búsqueda llegue a converger.

Hipótesis

El ajuste de los parámetros para el algoritmo tiene una gran influencia respecto al tiempo de ejecución, las heurísticas y la búsqueda del óptimo local del problema.

Método

- Estableceremos un escenario del problema principal 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. Esta configuración será la misma para todas las réplicas.
- Usaremos el algoritmo Simulated Annealing.
- Estableceremos a la heurística N° 1 como la heurística a usar en el problema.
- Elegiremos sólo la función cambiar_servidor para la generación de sucesores.
- Probaremos la segunda función de generación de estado inicial.
- Variaremos los valores de los parámetros en cada iteración.
- Ejecutaremos 10 experimentos para cada función escogida.
- Mediremos diferentes parámetros para realizar la comparación.

Resultados

En las siguientes tablas presentaremos los promedios de los tiempos de ejecuciones en segundos:

Con steps = 1000, stiter = 100

	K=1	K=5	K=25
Λ = 0,0001	5,813	5,856	5,851
Λ =0,001	5,869	5,794	5,781
Λ =0,01	5,893	6,014	5,78
Λ = 0,1	5,862	5,795	5,776

Con steps = 2000, stiter = 100

	K=1	K=5	K=25
Λ = 0,0001	10,798	10,735	10,719
Λ = 0,001	11,025	10,721	10,912
Λ = 0,01	10,656	11,043	10,672
Λ = 0,1	10,863	10,855	10,755

Discusión

Tal y como lo muestran las tablas, los resultados no muestran diferencias significativas al variar de los parámetros k y Λ. Se intentó repetir las experiencias muchas más veces de las que nos lo habíamos propuesto pero aún así no obtuvimos más alteración.

El único factor que se ha visto afectado es el tiempo de ejecución, y ello sucedió al variar el número de iteraciones de Simulated Annealing.

Conclusiones

Llegamos a la conclusión de que la hipótesis que fue planteada al inicio de la experiencia no es correcta, al menos según lo que demuestran nuestros experimentos. Podemos afirmar que los parámetros para este algoritmo dependen directamente del problema en el que va a ser aplicado, pero al menos en este proyecto no ha habido mucha diferencia al modificarlos.

Experimento 4

Observación

Una vez que hemos ido descubriendo cómo se comportan los algoritmos en un escenario fijo con el objetivo de establecer mejor nuestra metodología para encontrar una solución óptima, vemos conveniente el hecho de saber cómo se va comportando la misma configuración en escenarios distintos, es decir, con un problema variable. Lo importante aquí será analizar cómo varía el tiempo de ejecución del algoritmo con valores crecientes de los parámetros.

Planteamiento del problema

Ejecutamos el programa desarrollado con la configuración correspondiente pero variando los siguientes parámetros:

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

Luego, observamos el tiempo de ejecución para su variación acorde al cambio del escenario.

Hipótesis

Lo que se espera es que el tiempo de ejecución sea directamente proporcional al número de usuarios, y, al mismo tiempo, inversamente proporcional al número de servidores.

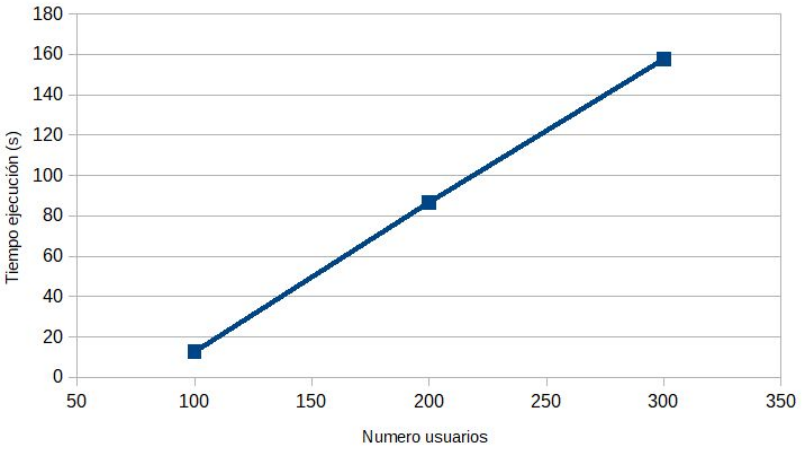
Método

- ➢ Usaremos el algoritmo Hill Climbing.
- ➢ Estableceremos a la heurística N° 1 como la heurística a usar en el problema.
- ➢ Elegiremos sólo la función cambiar_servidor para la generación de sucesores.

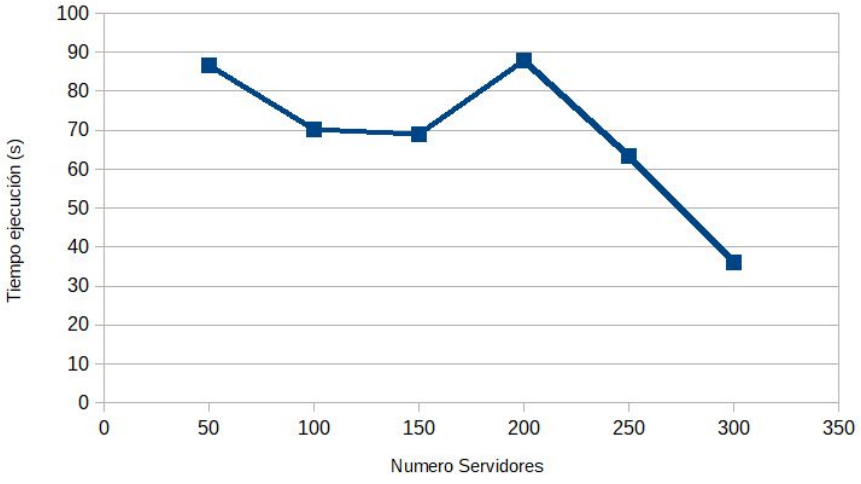
- Probaremos la segunda función de generación de estado inicial.
- Estableceremos un escenario del problema principal en el que el número de usuarios que piden ficheros es 100, el cual iremos incrementando el número de 100 en 100 y el número de servidores es 50.
- El número máximo de peticiones por usuario estará fijo en 5.
- El número mínimo de replicaciones estará fijo en 5.
- Ejecutaremos 10 experimentos para cada configuración escogida.
- Mediremos el tiempo de ejecución para ver la tendencia.
- Luego, cambiaremos a 200 usuarios e iremos variando la cantidad de servidores iniciando en 50 e incrementándolo de 50 en 50.
- Ejecutaremos 10 experimentos para cada configuración escogida.
- Mediremos el tiempo de ejecución para ver la tendencia.

Resultados

Usuarios	t (s)
100	12,762
200	86,665
300	157,824



N servidores	T (s) (promedio)
50	86,665
100	70,17
150	68,918
200	87,85
250	63,277
300	36,059



Discusión

Lo que se esperaba obtener era un tiempo de ejecución que suba al subir del número de usuarios, y que se baje al aumentar el número de servidores. El experimento ha confirmado nuestra hipótesis como se puede ver desde las tablas y los gráficos.

Conclusiones

Es normal que el tiempo de ejecución sea mayor cuando hay un número de usuarios más grande porque esto se traduce en estados que ocupan más memoria y que entonces necesitan más tiempo para ser generados.

Por lo contrario cuando el número de servidores es más grande, es muy probable que cada petición pueda ser servida en un tiempo menor porque tiene más servidores que la puedan sodisfar.

Experimento 5

Observación

En este proyecto debemos evaluar el comportamiento de todos los parámetro posibles a nuestro alcance, uno principal es la heurística que aplicamos dentro del problema para evaluar qué camino tomamos respecto a los nodos hijos que estamos generando. Dado el escenario del primer apartado, debemos estimar la diferencia entre el tiempo total de transmisión y el tiempo para hallar la solución, usando las dos heurísticas, implementando los criterios definidos. Para este experimento vamos a utilizar el algoritmo de Hill climbing. En este caso, experimentaremos cómo debéis incorporar la penalización del segundo criterio.

Planteamiento del problema

Analizaremos el comportamiento de ambas heurísticas planteadas para la evaluación de los nodos sucesores. Se va a ejecutar el Hill Climbing utilizando ambas las heurísticas para ver la diferencia entre los tiempos totales de respuesta y la desviación estándar de cada una, y para ver las diferencia entre los tiempos de ejecuciones.

Hipótesis

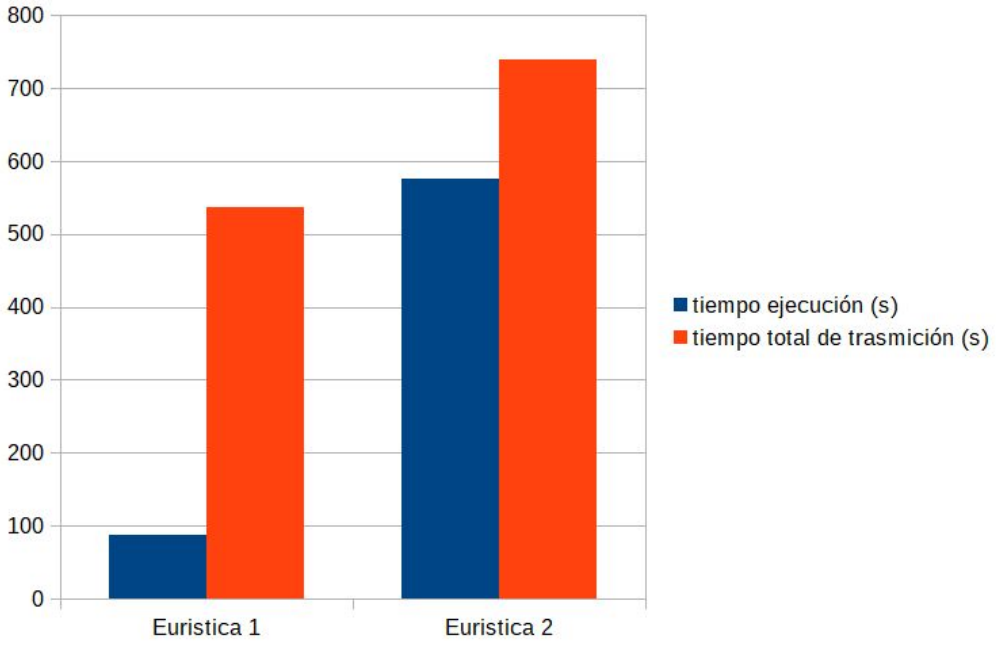
La heurística 2 es el que tiene mayor influencia sobre la generación de las soluciones sucesoras, y, por lo tanto, sobre el problema.

Método

- Estableceremos un escenario del problema principal 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. Esta configuración será la misma para todas las réplicas.
- Usaremos el algoritmo Hill climbing.
- Definimos a cambiar_servidor como el operador de generación de sucesores.
- Aplicaremos la estrategia de generación del estado inicial N° 2.
- Estableceremos a la heurística N° 1 como la heurística a usar en el problema.
- Ejecutaremos 10 experimentos para cada configuración.
- Mediremos diferentes parámetros y calcularemos sus promedios para realizar la comparación.
- Luego, estableceremos a la heurística N° 2 como la heurística a usar en el problema.
- Ejecutaremos 10 experimentos para cada configuración.

➢ Mediremos diferentes parámetros y calcularemos sus promedios para realizar la comparación.

Resultados



La heurística 1 produce un tiempo total de respuesta de 537,18 segundos y un tiempo de ejecución promedio de 86,77 segundos.

La heurística 2 produce un tiempo total de respuesta de 739,96 segundos y un tiempo de ejecución promedio de 575,85 segundos.

Discusión

Lo que se espera de obtener es un tiempo total de trasmisión que sea menor para el algoritmo que utiliza la heurística 2, por el hecho que esta heurística debería optimizar el siguiente criterio: minimizar el tiempo total de trasmisión de los ficheros pero con la restricción de que los tiempos de transmisión de los servidores han de ser lo más similares posible entre ellos.

Cabe mencionar que hemos elegido la desviación estándar como medida de similaridad.

Con respecto a los tiempos de ejecuciones, rescatamos el hecho de que ya en los experimentos precedentes hemos verificado que la heurística dos es más lenta que la número 1. Esto probablemente sea porque tiene una complejidad mayor.

En el mismo sentido, el experimento ha demostrado también cómo en verdad la heurística dos no es efectiva para optimizar este criterio, probablemente hay errores de proyectación de la función. Finalmente, podemos mencionar que los resultados del experimento no han confirmado nuestra hipótesis.

Experimento 6

Observación

En este experimento trataremos de estimar la diferencia entre el tiempo total de transmisión de los ficheros a los usuarios y el tiempo para encontrar la solución con la ejecución del algoritmo Simulated Annealing con los parámetros que anteriormente hemos calculado utilizando las dos heurísticas.

Planteamiento del problema

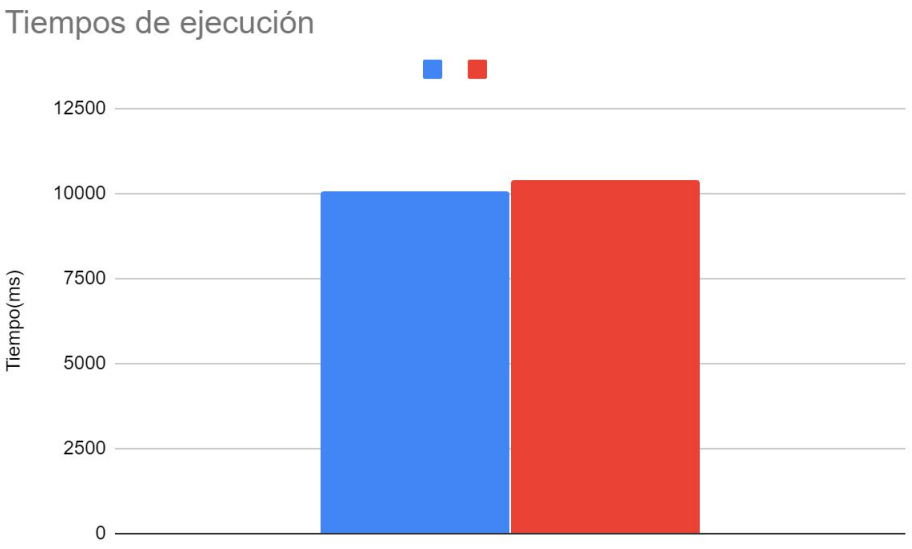
Ejecutaremos el programa desarrollado con el algoritmo Simulated Annealing con los parámetros adecuados con cada uno de los heurísticos y compararemos el tiempo de transmisión y el tiempo de ejecución

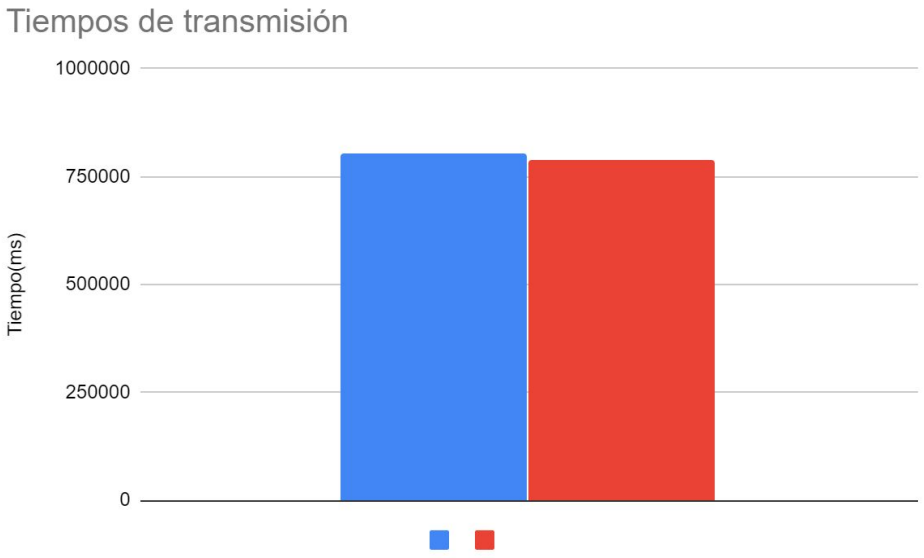
Hipótesis

La ejecución de Simulated Annealing con la heurística 2 se espera que tenga un tiempo de ejecución menor que con la heurística 1.
La ejecución de Simulated Annealing con la heurística 1 se espera que tenga un tiempo de transmisión menor que con la heurística 2.

Método

- Estableceremos un escenario del problema principal en el que el número de usuarios es 200, el número máximo de ficheros que pide un usuario es 5, el número de servidores es 50 y el número mínimo de replicaciones es 5.
- Utilizaremos el algoritmo Simulated Annealing con los parámetros siguientes:
  - Número de iteraciones = 2000
  - Iteraciones por cada cambio de temperatura = 100
  - k = 25
  - $\lambda = 0.0001$
- Aplicaremos la estrategia de generación del estado inicial N°2.





Azul: Heurística 1
Rojo: Heurística 2

Discusión

Se esperaba que los tiempos de ejecución con la heurística 2 fueran menores que con la heurística 1, pero no ha sido así. Los tiempos han sido muy parejos.
También se esperaba que los tiempos de transmisión fueran superior con la heurística 2 ya que se intenta que los tiempos de transmisión de cada servidor fueran similares. Pero los tiempos de transmisión también han sido parejos.

Conclusiones

La conclusión de este experimento es que las dos heurísticas tienen unos tiempos de ejecución y tiempos de transmisión muy parecidos ejecutando el algoritmo Simulated Annealing.

Experimento 7

Observación

Uno de los elementos que influyen en el problema es el número mínimo de replicaciones que puede tener un fichero dentro del sistema de ficheros. La idea de la técnica de replicación que explicamos al inicio de este informe tiene como objetivo el disminuir la posibilidad de que un sólo servidor atienda todas las peticiones de ficheros y se sobrecarga, de tal forma que detenga todo el proceso o la vuelva extremadamente lenta. Entonces el camino que tomamos fue duplicar el fichero en varios servidores al mismo tiempo, con la restricción de que esa cantidad no supere el 50% del número de servidores.

Lo que se piensa naturalmente es que mientras más replicaciones, menor es el tiempo de total de transmisión; sin embargo, puede que el algoritmo se comporte de manera que eso no sea lo que sucede.

Planteamiento del problema

Evaluaremos el comportamiento del tiempo total de transmisión de las peticiones y del tiempo en el que se obtiene dicha solución cuando se varía el número mínimo de replicaciones de los ficheros dentro del sistema de servidores.

Hipótesis

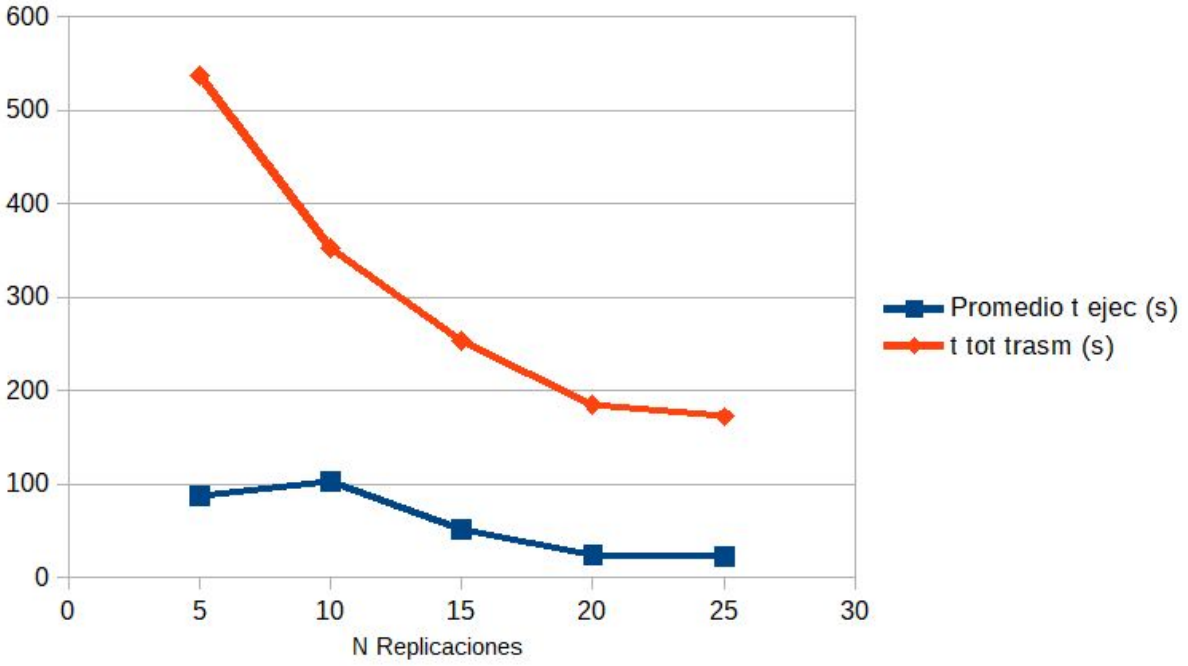
Mientras mayor sea el número mínimo de replicación, menor será el tiempo total de transmisión de los ficheros, así como el tiempo de ejecución en el que se obtenga una solución óptima.

Método

- Usaremos el algoritmo Hill Climbing.
- Estableceremos en primer lugar a la heurística N° 1 como la heurística a usar en el problema, pero luego de conseguir 10 réplicas lo modificaremos a la heurística N° 2.
- Elegiremos sólo la función cambiar_servidor para la generación de sucesores.
- Probaremos la segunda función de generación de estado inicial.
- Estableceremos un escenario del problema principal 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. Esta configuración será la misma para todas las réplicas.
- Respecto al número mínimo de replicaciones, experimentaremos primero con el valor en 5 e iremos incrementándolo en pasos de 5 hasta llegar a 25.
- Ejecutaremos 10 experimentos para cada configuración escogida.
- Mediremos el tiempo de ejecución y el tiempo total de transmisión para ver la tendencia.

Resultados

N Rep.	Tiempo promedio de ejecución (s)	Tiempo total de transmisión (s)	Desviación estándar (ms)
5	87,85	537,19	767
10	103,507	352,08	580
15	52,162	253,91	352
20	23,896	185,24	226
25	23,285	172,12	277



Discusión

Los resultados de este experimento muestran cómo lo que esperábamos se ha verificado. Entonces los tiempos de ejecución y lo tiempos total de trasmisión de los ficheros son indirectamente proporcionales a el factor de replicación de los ficheros. También la desviación estándar entre los tiempos de respuesta sigue esta tendencia.

Conclusiones

Con este experimento hemos demostrado cómo tener un número de replicaciones más alto para cada fichero hace que las respuestas a las peticiones sea más rápidas porqué para cada fichero hay más servidores que lo tiene y entonces tiene más posibilidad que hay un tiempo de respuesta más bajo y que la búsqueda del servidor sea más rápida.

Claramente estos aspectos se reflectan también sobre el tiempo de ejecución del programa.

Experimento 8

Observación

Los problemas de búsqueda local tienen una amplia variedad de algoritmos a aplicar para poder solucionarlos. El presente proyecto tiene como uno de los objetivos principales el poder comparar el funcionamiento de los mismos ante un mismo escenario. Por ello se ha escogido dos algoritmos que ven el problema desde dos ángulos diferentes: Hill Climbing y Simulated Annealing. Vamos a aprovechar este último experimento para poder comparar, finalmente, la performance de cada uno de estos algoritmos frente a nuestro problema de los servidores de ficheros.

Debemos tener en cuenta que no necesariamente el algoritmo que ofrece un resultado en menor tiempo es el que ofrece el mejor resultado, ya que depende de muchos otros factores.

Planteamiento del problema

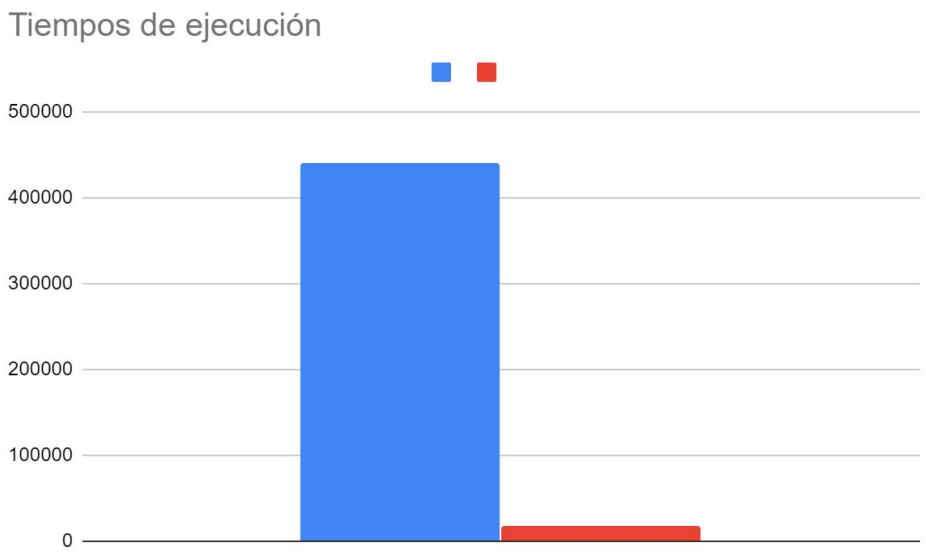
Teniendo en cuenta todas las experimentaciones previas, compararemos el comportamiento de ambos algoritmos para encontrar una solución óptima, respecto a las heurísticas, las funciones de generación utilizadas, los operadores, el tiempo total de transmisión y el tiempo de ejecución.

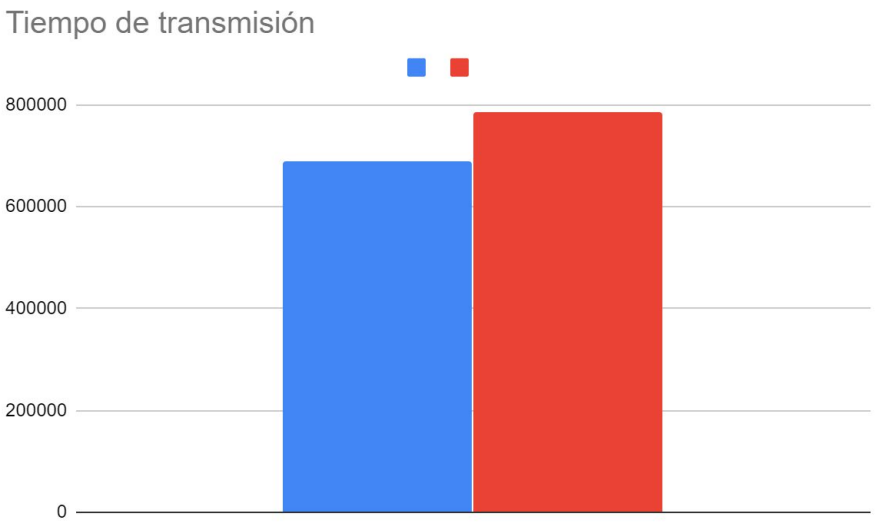
Hipótesis

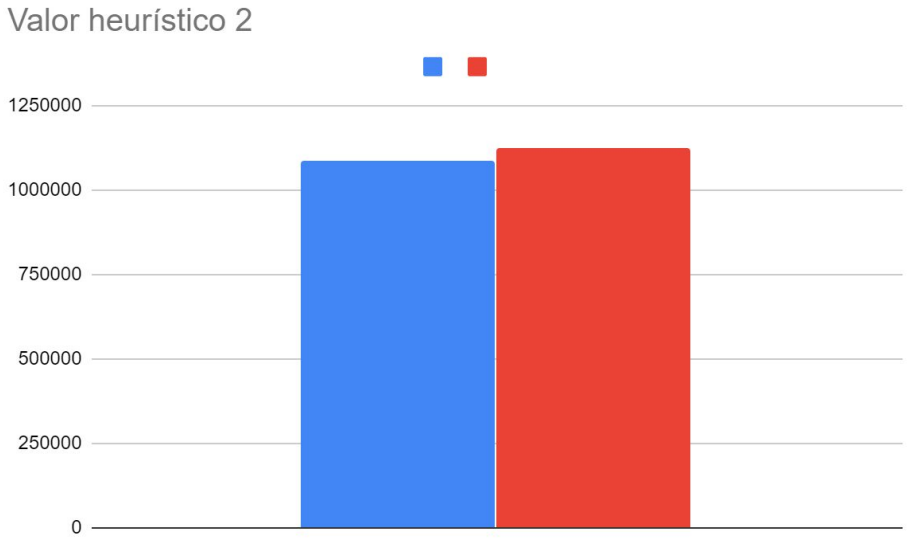
Esperamos que el problema desarrollado obtenga mejores resultados ejecutado con el algoritmo Simulated Annealing que con algoritmo Hill Climbing.

Método

- Estableceremos un escenario del problema principal 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. Esta configuración será la misma para todas las réplicas.
- Usaremos los algoritmos Hill climbing y Simulated Annealing.
- Aplicaremos la estrategia de generación del estado inicial N° 2.
- Estableceremos a la heurística N° 2 como la heurística a usar en el problema.


Tiempos de ejecución
500000
400000
300000
200000
100000
0


Tiempo de transmisión
800000
600000
400000
200000
0



Azul: Hill Climbing
Rojo: Simulated Annealing

Discusión

Esperábamos que el tiempo de ejecución con Simulated Annealing fuera inferior al del Hill Climbing pero no esperábamos que la diferencia fuera tan grande. Creemos que la diferencia es debido a que el Hill Climbing calcula todos sus posibles estados a través de su operador, en cambio Simulated Annealing solo opera sobre un estado.

En cambio, los tiempos de transmisión y los valores heurísticos 2 han sido mejores para Hill Climbing que para Simulated Annealing.

Conclusiones

Aunque Hill Climbing de mejores resultados en aspectos como los siguientes: Tiempo de transmisión y en los valores heurísticos; concluimos nos elegimos al algoritmo Simulated Annealing como la mejor para el problema ya que priorizamos el hecho de que un algoritmo que se aplique a la Inteligencia Artificial presente tiempos de ejecución considerablemente mejores.

Comparaciones entre métodos de búsqueda

El presente apartado contiene una especie de debate que pretende presentar una serie de comparaciones entre los resultados obtenidos con Hill Climbing y Simulated Annealing.

Los resultados de la experimentación señalan que la solución mejor al problema de los servidores de ficheros es el algoritmo Simulated Annealing con el uso de la heurística 1 y 1000 iteraciones, porque garantiza mejores tiempos de ejecución y un tiempo total de respuesta bastante eficiente.

Respecto a los valores de los parámetros de este algoritmo podemos mencionar que requiere de una experiencia adicional a las que se deben hacer para otros algoritmos, ya que requiere un ajuste previo de sus parámetros, teniendo en cuenta que estos valores no pueden ser los mismos para todos los problemas. En ese sentido, en el presente proyecto nos apoyamos en el experimento N° 3 para poder definir esos parámetros.

Después de llevarlo a cabo no hemos encontrado diferencias significativas ante la variación de los parámetros k y lambda.

Por otro lado, sabemos que el algoritmo Hill Climbing no siempre tiene la mejor perfomance respecto al anterior algoritmo mencionado, ya que tiende a llevar su solución al óptimo en el que caiga más cerca, por lo que depende mucho de su función de generación del estado inicial. En el mismo sentido, en los experimentos realizados se ha mostrado que su tiempo de ejecución sube demasiado rápido al aumentar el número de usuarios, entonces no la consideramos como una solución bastante eficiente.

Conclusiones

Con este trabajo hemos propuesto una solución al problema que explota la librería AIMA para solucionarlo a través de funciones que implementan técnicas de Inteligencia Artificial. Asimismo, hemos implementado técnicas de generación de soluciones iniciales, de generación de hijos, y de búsqueda diferentes para confrontar los resultados y encontrar la que sea mejor.

También hemos implementado los operadores, el cual probablemente sea el aspecto menos eficiente de este trabajo porque ambos operadores tienen factores de ramificación que seguramente son mejorables, y que se traducen en un algoritmo que no es bastante rápido y eficiente, como se puede ver en la experimentación de Hill Climbing.

Hemos intentado desarrollar operadores con factores de ramificación menores, pero entonces el algoritmo Hill Climbing no lograba crear nodos sucesores para lograr que se expanda dentro del espacio de soluciones, para lo cual decidimos seguir trabajando con los operadores propuestos, de tal forma que podamos garantizar el correcto funcionamiento del programa.

Por otro lado, como equipo hemos llegado a la conclusión de que los problemas de este tipo son muy aplicables a la vida cotidiana, tal y como lo explicamos en la introducción del proyecto. Además son complicados de resolver sin ayuda de las soluciones de la Inteligencia Artificial.

Asimismo, entendimos que cada uno de los factores que trabajamos en los experimentos son factores que influyen en diversas escalas al comportamiento de los algoritmos en la búsqueda de una solución óptima, y que debemos aplicar una serie de conocimientos de las matemáticas, la lógica y la estadística para poder tomar decisiones que nos ayuden a asegurar una propuesta lo suficientemente buena. De tal forma que podamos conseguir ahorro de los recursos que tenemos a disposición y una respuesta en un tiempo considerable.