Universitat Politècnica de Catalunya

Inteligencia Artificial

Bachelor Degree in Computer Science

Logística en Desastres Naturales

Autores:
Víctor Giménez
Guillem Ferrer
Jordi Armengol

Professor:
Javier Béjar

Q2 Curso 2017/2018

Contents

1 Introducción

Este documento tiene el objetivo de resolver un problema común en logística de desastres: el rescate/evacuación de personas en una zona de terreno de difícil acceso. Para ello implementaremos una inteligencia artificial basada en la estrategia de búsqueda local (los motivos de esta decisión se verán en la siguiente sección del documento).

En concreto, realizaremos una búsqueda local sobre el espacio de soluciones del problema, desplazándonos con un conjunto de operadores y según el criterio de una función de calidad.

Debido a la complejidad problema, también se realizarán una serie de experimentos para encontrar de forma empírica la mejor estrategia de resolución, cambiando operadores, soluciones iniciales y parámetros de la búsqueda.

2 Descripción del problema

2.1 Descripción formal del problema

2.1.1 Elementos de la entrada

Trataremos con un problema que definiremos formalmente como el siguiente:

- El problema considera un área rectangular de 50x50 Km^2
- Existen C centros, con sus respectivas localizaciones en metros (que deben estar en el borde de un rectángulo de 50Km*50Km, formalmente $(\forall i, min(c_i.x, c_i.y) = 0 || max(c_i.x, c_i.y) = 50000m)$).
- Existen G grupos, con un número x de personas t.q $1 <= x <= 12$, una prioridad p, que será 1 o 2, y una localización dentro del área de 50x50Km (en metros). La prioridad nos indica si el grupo está formado por heridos (prioridad 1) o no (prioridad 2).
- Hay un número H de helicópteros en cada centro.

Durante el rescate, un helicóptero podrá salir del centro, rescatar a no más de 3 grupos secuencialmente, y sin pasarse de la capacidad de 15 personas en una salida y regresar al centro. Si posteriormente dicho helicóptero debe realizar otra salida, deberá descansar 10 minutos antes de volver a salir.

2.1.2 Cálculo del tiempo de rescate

Tendremos en cuenta:

- **Tiempo de desplazamiento** por el terreno, teniendo cada helicóptero una velocidad constante de 100 km/h y considerando el cálculo de distancia euclídea.

- **Tiempo de rescate** de grupos, en que por cada persona en el grupo, el helicóptero tardará un minuto en rescatarla si el grupo es de prioridad 2 o dos minutos, si es de prioridad 1.
- **Tiempo de mantenimiento** de helicópteros, en que el helicóptero descansa 10 minutos entre dos salidas.

2.1.3 Objetivo del algoritmo y descripción de la solución

El objetivo es dar una planificación sobre el rescate de los grupos con los helicópteros, optimizando uno de los siguientes criterios:

- **Versión 1**: El tiempo de rescate, definido como la suma de tiempos de rescate de cada helicóptero (no consideramos paralelismo entre helicópteros)
- **Versión 2**: El tiempo de rescate anterior y el tiempo de rescate máximo que se tarda en rescatar a un grupo de prioridad 1 de entre los helicópteros. Es decir, considerando paralelismo total entre helicópteros, el tiempo desde el inicio de la búsqueda hasta que todo grupo de prioridad 1 se encuentre en un centro.

Formalizando un poco más el problema, consideraremos solución, una planificación en que se satisfaga las siguientes restricciones y condiciones:

- Una solución debe rescatar a todos los grupos.
- Todo helicóptero debe rescatar grupos completos.
- Un helicóptero, en un vuelo, puede rescatar máximo 15 personas.
- Un helicóptero, en un vuelo, puede recoger hasta 3 grupos.
- Todo vuelo de un helicóptero comienza y termina en su centro de origen.
- Los grupos no pueden rescatarse parcialmente. Se ha de rescatar a todos los miembros en la misma 'recogida'.

De la misma forma, formalizaremos el cálculo de tiempo ya mencionado de la forma siguiente:

- Un helicóptero viaja a 100 Km/h, siguiendo el recorrido más corto (distancia euclídea).
- El rescate de un grupo tarda 1 minuto por miembro, 2 minutos si el grupo es de prioridad 1.
- Si un helicóptero debe hacer un vuelo después de otro, debe reposar 10 minutos con anterioridad. No contaremos los 10 minutos cuando el criterio de cálculo de tiempo se haya cumplido. Es decir, en el cálculo de tiempo total, no contaremos los 10 minutos de reposo tras el último rescate de cada helicóptero, y en el cálculo de tiempo de rescate de heridos no contaremos los 10 minutos tras el rescate del último grupo de heridos para ese criterio (pero potencialmente sí para el cálculo total).

Nótese que, aunque consideramos el tiempo de rescate como el sumatorio del tiempo de los helicópteros como si fueran en serie, último rescate de heridos lo consideraremos interno de cada helicóptero (cuánto tarda cada helicóptero en rescatar a su último grupo de heridos).

2.2 Justificación del uso de Búsqueda Local

Nos percatamos de que éste problema se trata de una asignación de grupos a helicópteros, con la peculiaridad de que se realizan en 'salidas', que llamaremos a partir de ahora 'vuelos'. Así pues, el problema se puede resolver dando las asignaciones de los grupos a los vuelos, y de los vuelos a los helicópteros, o de los helicópteros a los vuelos, y de éstos a los grupos. En ambos casos, en las asignaciones importa el órden entre grupos dentro de un vuelo, y de vuelos dentro del helicóptero.

Vemos que el espacio de búsqueda es extraordinariamente grande, lo cual nos impediría encontrar la solución óptima directamente, o usar algoritmos de exploración sistemática, pues el tamaño de la memoria no nos lo permitiría.

Observamos también que el espacio de soluciones de este problema, planteado de una forma correcta y con operadores bien seleccionados, puede ser explorado en su totalidad sin salirse de él, asegurando que se mantienen las restricciones.

Nótese también que el tiempo es una buena función de calidad continua y con pocas mesetas. De hecho, con un buen conjunto de operadores hasta podríamos asegurar que no tiene mínimos locales demasiado lejos de la solución real.

Por todas estas razones nos inclinamos a pensar que este problema se puede resolver mediante búsqueda local, en concreto, los algoritmos de Hill Climbing y Simulated annealing.

3 Diseño de los elementos clave de la búsqueda

3.1 Representación del estado

El estado lo consideraremos como una asignación de grupos a vuelos, y de vuelos a helicopteros, estando ambas asignaciones consideradas como una secuencia cronológica. Con algo tan simple como esto podremos reconstruir una solución total. Notemos particularmente que en estas asignaciones, para que el estado sea solución, se dará que un grupo está asociado a un y solo un vuelo, y los vuelos a un y solo un helicóptero. Mientras no deshagamos este hecho y nos aseguremos de cumplir las restricciones de carga del helicóptero y de numero de grupos por vuelo, nuestro estado seguirá siendo una solución, sea mejor o peor.

Notemos, además, que en la segunda versión de función de calidad, se requiere del tiempo máximo de rescate de un grupo de prioridad uno. Consideraremos éste tiempo como el máximo de los tiempos que tardan los helicópteros en rescatar a su último grupo de heridos.

3.1.1 Espacio de soluciones

Examinemos ahora, de manera muy aproximada, cuál es el tamaño del espacio de búsqueda. Tenemos G grupos a distribuir entre vuelos de diversas formas, o bien en grupos de 1 (1 posibilidad), todo grupos de 1 excepto uno de 2... todo grupos de 2, todo grupos de 1 menos un grupo de 3... todo grupos de 1 menos un grupo de 2 y uno de 3... teniendo en cuenta todas las posibles ordenaciones de cada grupo de más de 1 elemento. De hecho, si no tuviéramos en cuenta la posible ordenación de cada subconjunto, tendríamos un subconjunto de la partición del conjunto de grupos. Está demostrado que el tamaño de la partición de un conjunto es el número de Bell, que crece a ritmo $O(e^{e^n-1})$, donde n es el número de elementos del conjunto. Como para cada vuelo V hay un numero constante de permutaciones (máximo 6, permutaciones sobre un vuelo de 3), y habrá máximo un número G de vuelos, esto se multiplica por 6^G. Vagamente vemos que el espacio de búsqueda crece a un ritmo $O(6^G * e^{e^G})$. Además, para cada una de estas posibilidades, debemos asignar cada grupo a un helicóptero. O lo que es lo mismo, dada una de estas distribuciones, debemos dar a cada conjunto un valor entre 0 y C*H. Esto se traduce en $O(C * H^{6^G * e^{e^G}})$. El espacio de soluciones es extraordinariamente grande, y crece muy rápido al aumentar el tamaño de la entrada.

3.1.2 Tipos de representación considerados

Partimos de 4 diferentes representaciones del estado, diferentes entre sí por motivos de eficiencia, no de expresividad:

- Un vector tridimensional travels [i][j][k], donde el elemento i-ésimo representa los vuelos del helicóptero i-ésimo (dimensión C*H). El elemento j-ésimo representa el j-ésimo vuelo de dicho helicóptero. El k-ésimo elemento de un vuelo representa el grupo que se rescata en ese vuelo tras rescatar k grupos antes que él en el mismo vuelo. Por ejemplo, travels[2][3][0] representa el primer grupo en ser rescatado durante la cuarta salida del helicóptero 3 (contamos desde el 0).

  El coste espacial esperado es de O(C*H + G), donde C es el numero de centros, H el numero de helicópteros por centro y G el numero de grupos, si usamos un Array como estructura de datos. Esto sale del hecho de que habrá G grupos en el vector exactamente, sin embargo si C y H son muy grandes debemos considerar que la estructura en sí es la que ocupa (como en los costes de grafos O(V+E)). Nótese que ésta es la más óptima en espacio.

  Sin embargo, el coste de recomputar el tiempo para cada evaluación de la heurística requeriría de recorrer todo el vector, que sería coste O(C*H+G) también. Podríamos encontrar también el tiempo latencia de la segunda función de calidad en el mismo tiempo.

- Igual que 1, sin embargo introduciremos un dato adicional: Tiempo total. Debido al poco tamaño del dato, el tiempo de computación si tuviéramos que hacerlo a cada evaluación y la poca dificultad de actualizarlo en tiempo de 'operación', hemos considerado que este dato podría reducir significativamente el tiempo de ejecución, al menos para la versión uno de la función de calidad.

  El coste espacial esperado es O(C*H+G), igual que en 1.

El coste temporal esperado cambia. El acceso al tiempo es O(1), y el coste es un recorrido por un o dos vectores de máximo 3 posiciones O(1) para la mayoría de operaciones (tan solo requiere cambio el operador swapPilot, de la versión 2, que requiere un recorrido O(G) en peor caso). Si queremos calcular la segunda función de calidad, sin embargo, este tiempo requiere de nuevo de un recorrido por todo el vector ($O(C * H + G)$)

- Igual que 2, pero añadiremos otra estructura de datos adicional: un vector de tamaño C*H en que se guarde, para cada helicóptero, el tiempo que tarda en rescatar al último grupo de prioridad 1. Seguimos el razonamiento de 2, sin embargo un único valor tiempo no tiene suficiente expresividad para facilitar tanto el cálculo de la función de calidad 2. Cada vez que usemos una operación sobre un vuelo o helicóptero tendremos que actualizar todos los helicópteros asociados a la operación. Nótese que ésta es la menos óptima espacialmente.

  El coste espacial es O(C*H+G), aunque ahora tendremos dos vectores de tamaño C*H.

  El coste temporal es el mismo para el cálculo de tiempo global, sin embargo para la latencia requiere de un recorrido por un vector C*H, y para cada operación requiere O(G) en el peor caso (recálculo del peor tiempo de un o dos helicópteros, teniendo entre ellos todos los grupos).

Las representaciones que nos interesan son la 2 y la 3. Usaremos la 2 siempre que sepamos que usaremos solo la primera función de calidad, y la 3 en caso contrario.

3.2 Solución inicial

Para empezar a usar búsqueda local, necesitamos una primera solución inicial.

Bajo el conocimiento del funcionamiento de algoritmos a usar, nos hacemos vagamente a la idea de que soluciones poco óptimas tienen bastante más potencial de ser más óptimas tras correr el algoritmo de búsqueda, pues estarán más lejos de un mínimo local en nuestra función de calidad (nótese que este problema busca minimizar la función, bajo la función que usaremos, definida posteriormente).

Además, y tras examinar nuestras posibilidades de operadores, es importante que las soluciones iniciales permitan el acceso a todo el espacio de búsqueda para asegurar una mínima optimalidad. Bajo este hecho, y ya que nuestros conjuntos de operadores se mantendrán alejados de la posibilidad de 'crear' vuelos nuevos o mantener vacíos, evitaremos soluciones que ya comienzan con más de un grupo en un vuelo.

Dicho esto, partimos de las siguientes opciones:

- El estado inicial es, para cada grupo, un vuelo. Todos los vuelos serán asignados al helicóptero 0, que rescatará a todos los grupos. Ésta es la solución trivial peor, prácticamente maximizando el tiempo total de rescate (podría peor si buscásemos para cada grupo su helicóptero más lejano), y por ello tiene potencial de acercarse más al mínimo general.

  El coste temporal de hallar la solución es O(G), el mínimo posible.

- Similar a 1, sin embargo distribuiremos los vuelos entre helicópteros. Expresado formalmente, el grupo i-ésimo le será asignado al helicóptero ($i\%(C * H)$)-ésimo. El primer grupo

al primer helicóptero, el segundo al segundo, y así sucesivamente hasta no tener más grupos. Si nos quedamos sin helicópteros, vuelta a empezar.

Nótese que evitamos tener dos grupos ya en un mismo vuelo, y que la bondad debería ser mejor, pues ahorramos más descansos al repartirlo entre helicópteros.

El coste es el mismo que el de 1.

- Distribuiremos cada grupo en vuelos de un solo grupo, que será asignado a un helicóptero de su centro más cercano. Los helicópteros se asignarán según el método *Round-Robin*, (similar a la distribución del método 2, pero solo dentro del mismo helicóptero).

  Esta es la solución con mejor bondad inicial.

  El coste se eleva al tener que buscar el centro más cercano, pasando a ser O(G*C)

Avanzando los resultados empíricos recibidos, hemos observado que, dado un conjunto de operadores ya decidido previamente al experimento (detallado en la siguiente subsección y en la sección experimental), ejecutamos el algoritmo 10 veces (con distribución aleatoria de grupos y centros) con cada solución, y sin embargo los datos eran poco consistentes entre repeticiones del experimento.

Para precisar mejor las diferencias entre ellos, aumentamos a 30 el número de iteraciones, y entonces vimos claramente que entre la primera y la segunda soluciones, el 'coste' final era el mismo estadísticamente, y menor que el de la tercera opción. Llegado este punto, escogimos la primera opción, pues experimentalmente el tiempo de ejecución era ligeramente mejor que el del segundo (hallaba la solución final antes).

3.3 Conjunto y subconjuntos de operadores

Necesitaremos, ahora, definir nuestro modo de desplazarnos entre soluciones.

Debido a que el número de helicópteros es fijo, no será necesario añadir o eliminar elementos en una dimensión del vector. Por otro lado, el número de vuelos es variable, y el número de grupos en un vuelo también lo es (aún siendo el número de grupos constante). Esto nos lleva a pensar en la necesidad de distintos tipos de operadores.

Inicialmente, y para potencialmente reducir el número de operadores inútiles, intuitivamente decidimos dividirlos por granularidad:

- **Group-wise** o granularidad grupo, actuará sobre la asignación de grupos a vuelos. Deberían permitirnos no sólo cambiar un grupo de un vuelo a otro, sino también permutar el orden de 'visita' de un vuelo, ya que potencialmente puede reducir tiempos.

- **Flight-wise** o granularidad vuelo, que actuará sobre la asignación de vuelos completos a helicópteros.

  Directamente vemos que para ésta granularidad y la versión 1 de calidad, solo necesitamos un operador que nos permita cambiar un vuelo de un helicóptero a otro. Sin embargo, y debido a la segunda función de calidad, también necesitaremos poder cambiar el orden de los vuelos dentro de un mismo helicóptero.

Aparentemente esta purga de operadores podría reducir nuestro espacio de búsqueda, aislándonos de la posible solución óptima. Tras un mejor vistazo, nos parece que, dados nuestros estados iniciales, esto es imposible. Cualquier grupo puede acabar en cualquier vuelo de cualquier helicóptero aplicando estos operadores (sin tener en cuenta la calidad), y lo único que podría impedir este hecho sería que la función de calidad impidiera el uso de ciertos operadores ya que inicialmente son malos y solo después nos llevan a una solución mejor. (Avanzando resultados empíricos, esta última suposición es cierta).

Ésta fue la idea original, y sin embargo nos planteamos posteriormente que deberíamos poder cambiar grupos entre vuelos de helicópteros del mismo centro.

No nos detuvimos ahí. Dudando de nuestras decisiones intuitivas y prefiriendo dar este trabajo al algoritmo, decidimos incluir también un conjunto de operadores altamente expresivos, con una ramificación explosiva, que nos permitiera mover grupos entre vuelos de dos helicópteros cualesquiera.

Dicho esto, tendremos un conjunto genérico de operadores como el siguiente (nótese que el símbolo % indica parametro implícito en otros parámetros):

- GroupWise:
  - movAndDelete (g, %f1, f2): mueve g de f1 a la última posición de f2. Si f1 queda vacío, lo elimina del conjunto de vuelos.
    Ramificación: $G * O(G)$, pues en el peor caso hay G vuelos, aunque lo esperado es mucho menos. Si además consideramos la versión 'restringida', el esperado es $G*O(G/(C*H))$, pues los vuelos deberían estar repartidos entre helicópteros durante la mayor parte de la ejecución.
  - swap (g1, %f1, g2, %f2): Intercambia g1 y g2 entre los vuelos, manteniendo la posición del intercambiado.
    Ramificación: G*G. Por la misma lógica que antes, se espera $G * O(G/(C * H))$ en la versión restringida.
  - permute(f1, n): Intercambia el orden de visita de grupos dentro del vuelo. Hay 1 permutación para vuelos de 2, y 5 para vuelos de 3.
    Ramificación: $O(G)$.
- FlightWise:
  - switchPilot (f, %h1, h2): delega el vuelo f de H1 a H2.
    Ramificación: $O(G) * H$
  - swapOrder (f1, %f2): intercambia f1 y f2, dentro de los vuelos de un helicóptero. Nota: para reducir ramificación, f2 es el siguiente a f1 en el la planificación, y el último vuelo no puede ser f1.
    Ramificación: $O(G)$

Como nota, antes de aplicar estos operadores, siempre comprobaremos previamente que se mantengan las restricciones de carga del helicóptero.

Dados estos operadores, elegimos una serie de subconjuntos que nos permitan explorar todo el espacio de búsqueda:

- 1. movAndDelete, swap, switchPilot, para la primera versión del problema, sin permitir cambios entre helicópteros (excepto switchPilot)
- 2. movAndDelete, swap, switchPilot, para la primera versión del problema, permitiendo cambios entre helicópteros del centro (excepto switchPilot)
- 3. movAndDelete, swap, switchPilot, para la primera versión del problema, permitiendo cambios cualesquiera.
- 4. movAndDelete, permute, switchPilot, para la primera versión del problema, restringido
- 5. movAndDelete, permute, switchPilot, para la primera versión del problema, no restringido.
- 6. movAndDelete, swap, switchPilot, swapOrder, para la segunda versión del problema, no restringido.

Avanzando resultados experimentales, mucho de los subconjuntos anteriores no llegaron a contrastarse debido a obsolescencia respecto a resultados empíricos previos.

Nuestros experimentos contrastaron 1 con 3, bajo condiciones de un solo helicóptero por centro, y demostraron que 3 era bastante mejor que 1 (del orden de 40 minutos). Esto eliminó a 1 y a 2 de nuestra lista de candidatos, pues son idénticos en estas condiciones y 1 ya falló.

Posteriormente comparamos nuestro ganador, 3, con 5. 3 volvió a destacar, con 20 minutos de diferencia esperada. Este hecho eliminó a todos nuestros candidatos de la versión uno, pues 5 era el segundo más expresivo tras 3 de nuestra lista de candidatos actualizada.

Finalmente, nos decantamos por dos subconjuntos de operadores: 3 y 6, para las versiones 1 y 2 del problema.

3.4 Función de calidad

Bajo el enunciado del problema, tenemos dos criterios ya numéricos a optimizar: el tiempo secuencial y cuánto tardamos en rescatar al último grupo de heridos.

Inicialmente pensamos en buscar criterios más complejos que posiblemente nos permitiera usar operadores con menos ramificación y que el heurístico guiara a la búsqueda mejor, y sin embargo las opciones que surgían no nos convencían, pues podían acabar en zonas que ni siquiera eran mínimo local de la función original a optimizar.

Por ejemplo, pensamos añadir una ponderación añadida para ayudar a que los grupos fueran al centro más cercano (función entropía), pero esto podría separar dos grupos que en un viaje serían rescatados más rápido en dos helicópteros, pues sus centros más cercanos no coincidieran.

Buscamos también encontrar una manera de hacer una función que 'premiara', crear nuevos vuelos, para evitar el castigo en tiempo que es la espera entre vuelos de un helicóptero, pero podría hacer que al terminar la búsqueda, añadiera vuelos vacíos para premiarse a sí misma sin motivo, y no parecía viable bajo nuestros algoritmos.

Sin embargo, sí que encontramos algo ligeramente problemático con la segunda función de calidad. Dada la situación en que tenemos el último vuelo de heridos (del helicóptero que tarda más en

rescatar a su último grupo de heridos) con más de dos grupos de heridos, sería posible que el conjunto de operadores decidiera no priorizar uno de esos grupos de heridos (mover/intercambiar con grupos no heridos asignados antes que él), pues no optimizaría el segundo criterio (el vuelo sigue teniendo grupos de heridos) y aumentaría el tiempo de rescate (porque aumenta la distancia de viaje).

Es decir, la situación en que, por haber dos grupos de heridos en el último viaje, no los priorizamos ya que no optimizaría nada siendo ellos sólos. De hecho, este mismo motivo nos motivó a añadir el operador de swapOrder, pero éste no hace nada si el grupo previo también tiene heridos.

Por ello, se nos ocurrió 'castigar' tener más de un grupo de heridos en la última posición, con 10 minutos por grupo de heridos. Esto forzaría a hacer un swapOrder si el grupo anterior tiene menos grupos de heridos o incluso a mover uno de los dos grupos siempre que 10 * ponderación fuera menor que el tiempo asociado al desvío de otro vuelo para rescatar a este grupo (donde ponderación es el peso asociado al segundo criterio).

Finalmente y bajo la decisión de evitar funciones potencialmente problemáticas, decidimos probar las siguientes funciones heurísticas:

- Sumatorio de los tiempos de los helicópteros.
- Sumatorio de los tiempos de los helicópteros + λ * máx(tiempo en rescatar al último grupo de heridos de cada helicóptero)
- Sumatorio de los tiempos de los helicópteros + λ * (máx(tiempo en rescatar al último grupo de heridos de cada helicóptero)+10*(grupos de heridos en ultimo vuelo-1))

El parámetro λ, a elegir en tiempo de ejecución, decide qué importa más: el tiempo total del rescate o la prioridad que se da a salvar heridos.

$\lambda = 0$ es lo mismo que la primera función de calidad, no consideramos a los heridos prioritarios.

$\lambda = 1$ implica que nos importa tanto el tiempo total como el tiempo máximo de rescate.

λ superior a 1 forzará al algoritmo a sacrificar más tiempo total de rescate siempre que podamos rescatar a todos los heridos contra antes mejor.

Avanzando resultados experimentales se descubre que no hay diferencia real entre la segunda y la tercera función de calidad. Esto nos sorprendió inicialmente, pues pensamos que el algoritmo se quedaría atascado en la situación mencionada. Intuimos que la razón de que no haya diferencia entre ambas es que el escenario dado no llegue a suceder por alguna razón, como el modo de búsqueda que se realiza, o que sea tan improbable que nuestros experimentos no lo hayan detectado.

4 Experimentación

4.1 Entorno de experimentación

Los experimentos se han realizado en las siguientes condiciones:

- **Máquina de ejecución**

  - **Procesador:** Intel i7-2600K de 3.4 GHz.
  - **Memoria RAM:** 16 GB
  - **S.O.:** Windows 7 Professional

- **Otros datos**
  - **IDE:** Netbeans 8.2
  - **Memoria asignada** 1024MB (a no ser que el experimento especifique lo contrario
  - **Cálculo de tiempo de ejecución:** Librería System de Java (método currentTimeMillis)

4.2 Experimento 1: Operadores

La selección de un conjunto de operadores es crítica en búsqueda local. Operadores poco expresivos pueden quedar atascados en mínimos locales con más facilidad que operadores muy ramificados, pero una ramificación explosiva aumenta mucho el tiempo de ejecución del programa, potencialmente llegando a agotar la memoria.

Por ello, realizamos un experimento para encontrar el conjunto que sea el mejor en optimizar los criterios de calidad. Si hay dos que no presentan diferencia, escogeremos aquel que tenga menor tiempo de ejecución estimado.

Para los siguientes experimentos, tendremos el siguiente escenario:

G=100, C=5, H=1. Situación de los grupos y los centros sobre el terreno: aleatoria. Algoritmo: Hill Climbing.

Dado este escenario, decidimos, inicialmente, enfrentar los conjuntos de operadores 1 y 3. Recordemos:

- 1. movAndDelete, swap, switchPilot, para la primera versión del problema, sin permitir cambios entre helicópteros.
- 3. movAndDelete, swap, switchPilot, para la primera versión del problema, permitiendo cambios cualesquiera (no restringido).

También fijaremos el método de inicialización y usaremos el método 1: Para cada grupo, un vuelo. Todos los vuelos serán asignados al helicóptero 0

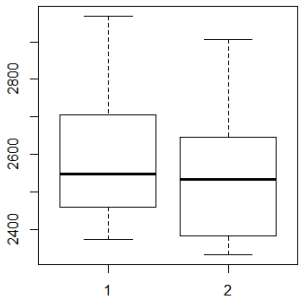
La siguiente tabla resume el primer experimento que realizaremos:

Observación	Pueden haber conjuntos de operadores mejores que otros
Planteamiento	Compararemos los conjuntos de operadores 1 y 3 entre ellos
Hipótesis	• 1 y 3 son iguales en tiempo de rescate (H0) o uno es mejor que el otro • En caso de igualdad, 1 y 3 son iguales en tiempo de ejecución (H0) o uno es más rápido que el otro
Método	• Elegimos 10 semillas aleatorias • Para cada semilla, ejecutamos el algoritmo HC y obtenemos el tiempo de rescate y el de ejecución con ambos operadores. • El escenario del problema considera 100 grupos, 5 centros y 1 helicóptero por centro.

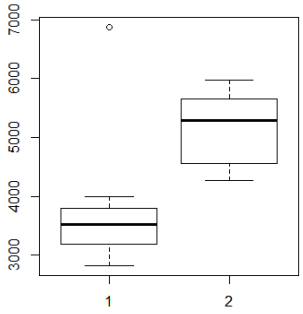
Resultados experimentales: (NOTA: todos los datos se encuentran adjuntos al proyecto en los ficheros de datos correspondientes)

seeds	time1	time2	exec1	exec2
283	2645.49	2583.71	6877	5572
127	2546.3	2528.84	3988	5262
239	2968.49	2905.53	2820	4263
228	2549.25	2539.88	3110	4479
290	2909.79	2900.35	3181	4562
33	2373.53	2339.85	3542	5311
13	2706.88	2646.48	3398	5276
238	2460.62	2382.68	3508	5980
53	2503.36	2465.18	3676	5938
345	2410.85	2333.74	3798	5651

Veamos estos datos en boxplot:



(a) Comparación del tiempo de rescate



(b) Comparación del tiempo de ejecución, en ms

Nótese que fue necesario desplazar el origen de coordenadas para poder mostrar las figuras y aparenta haber más diferencia de la que hay realmente

Se nos ocurre comparar la diferencia real entre ambos tiempos de rescate, pues un boxplot está ligeramente por debajo del otro.

Paired t-test
data: time1 and time2
t = 5.3064, df = 9, p-value = 0.0004897
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
25.71975 63.94425
sample estimates: mean of the differences 44.832

Paired t-test
data: time1 and time2
t = 5.3064, df = 9, p-value = 0.9998
alternative hypothesis: true difference in means is less than 0
95 percent confidence interval:
-Inf 60.31939
sample estimates: mean of the differences 44.832

Vemos empíricamente que hay diferencia entre los datos. Time2 (asociado al tiempo de rescate del conjunto de operadores 3) es notablemente mejor (44 minutos), ya que rescata a todos los grupos antes.

Esto, como ya hemos mencionado anteriormente, nos lleva a descartar varios grupos de operadores.

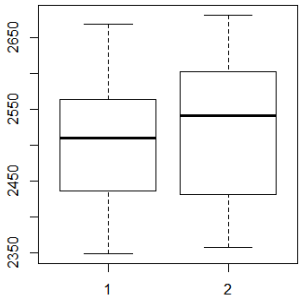
4.2.1 Experimento 1.2

Tenemos otro candidato que potencialmente podría ser igual que el conjunto 3 y tener mucha menos ramificación. Se trata del conjunto 5. Repetimos el experimento 1 bajo exactamente las mismas condiciones, esta vez con 3 y 5 como conjuntos a probar.

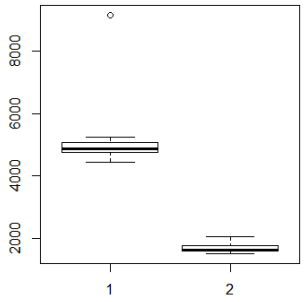
Los resultados empíricos fueron los siguientes

Resultados experimentales:

seeds	time1	time2	exec1	exec2
386	2561.98	2602.68	9161	2001
350	2669.25	2678.46	4428	1598
372	2348.92	2383.13	4828	1765
200	2436.96	2431.57	4478	1536
32	2351.08	2358.19	5241	2042
368	2475.6	2512.31	4848	1580
43	2662.18	2681.55	4942	1629
385	2507.8	2503.11	5079	1740
92	2564.04	2583.39	4871	1635
34	2512.56	2570.35	4753	1505



(a) Comparación del tiempo de rescate, a la izquierda el set 3



(b) Comparación del tiempo de ejecución, a la izquierda el set 3, en ms

```
Paired t-test
data: time1 and time2
t = -3.2756, df = 9, p-value = 0.009597
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-36.241636 -6.632364
sample estimates: mean of the differences -21.437
```

De nuevo, vemos que el conjunto de operadores 3 gana (por un margen de 20 minutos). Como añadido adicional, vemos que el conjunto 5 es muchísimo más rápido en términos de tiempo de ejecución.

4.3 Experimento 2

Ahora que ya disponemos de un conjunto de operadores para las circunstancias del escenario planteado, nos planteamos encontrar cuál es la mejor manera de iniciar la búsqueda.

En una búsqueda local, la solución inicial influye en los resultados finales y el tiempo de ejecución, así que se nos ocurre contrastar las diferentes opciones propuestas en la sección de elementos de la búsqueda.

Dados los tres métodos de inicialización mencionados con anterioridad:

- El estado inicial es, para cada grupo, un vuelo. Todos los vuelos serán asignados al helicóptero 0, que rescatará a todos los grupos.
- Similar a 1, sin embargo distribuiremos los vuelos entre helicópteros. Expresado formalmente, el grupo i-ésimo le será asignado al helicóptero $(i\%(C*H))$-ésimo. El primer grupo al primer helicóptero, el segundo al segundo, y así sucesivamente hasta no tener más grupos. Si nos quedamos sin helicópteros, vuelta a empezar.
- Distribuiremos cada grupo en vuelos de un solo grupo, que será asignado a un helicóptero de su centro más cercano. Los helicópteros se asignarán según el método *Round Robin*, (similar a la distribución del método 2, pero solo dentro del mismo helicóptero).

Notamos que están ordenados por complejidad de construcción y bondad ascendientes.

Partiremos del mismo escenario que en el experimento 1:

G=100, C=5, H=1. Situación de los grupos y los centros sobre el terreno: aleatoria. Algoritmo: Hill Climbing.

Usaremos el conjunto de operadores 3, el mejor empíricamente según el experimento 1.

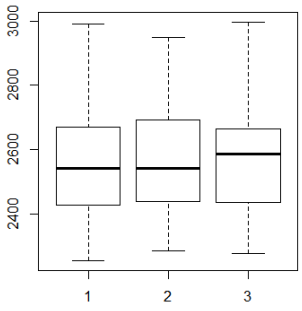
Avanzamos ahora los datos reales del experimento 2 tras una revisión. Inicialmente, hicimos el experimento con 10 semillas, pero al repetirlo los datos eran inconsistentes. Tuvimos que aumentar el número de semillas para obtener datos más fiables, de 10 a 30.

La siguiente tabla detalla el experimento a realizar:

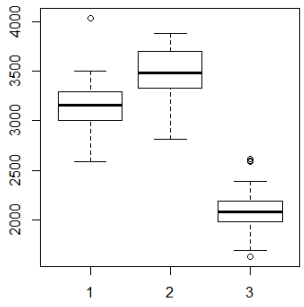
Observación	Es posible que soluciones iniciales diferentes nos lleven a soluciones mejores que otras
Planteamiento	Compararemos las 3 inicializaciones detalladas en la descripción de la sección 3.2
Hipótesis	• Todos los estados iniciales son iguales (H0) o hay unos mejores que otros • En caso de empate entre los mejores, también son iguales en tiempo de ejecución (H0) o uno es más rápido que otros
Método	• Elegimos 30 semillas aleatorias • Para cada semilla, ejecutamos el algoritmo HC y obtenemos el tiempo de rescate y el de ejecución probando todas las inicializaciones. • El escenario del problema considera 100 grupos, 5 centros y 1 helicóptero por centro, usando el conjunto de operadores 3

Los resultados experimentales son los siguientes:

seeds	time1	time2	time3	exec1	exec2	exec3
91	2253.62	2283.38	2274.61	4037	3804	2309
204	2594.0	2615.11	2608.5	2981	3387	1995
164	2790.31	2777.17	2766.17	3000	3141	2073
218	2666.64	2650.52	2639.06	3068	3471	2612
324	2588.68	2554.57	2598.58	3341	3702	2112
1	2670.06	2644.12	2662.93	3207	3756	1951
76	2655.07	2692.92	2654.91	2780	2905	1940
136	2404.64	2437.6	2435.75	3292	3845	2280
318	2426.29	2442.7	2457.74	3225	3788	2364
72	2725.81	2704.23	2733.94	2587	2814	2072
199	2792.71	2802.92	2871.88	3210	3331	1630
195	2541.78	2568.9	2603.75	2822	3344	2187
34	2512.56	2528.33	2586.73	3343	3476	2592
186	2342.41	2386.17	2400.09	3153	3885	2072
306	2427.66	2439.59	2432.59	3500	3665	2599
242	2622.83	2565.6	2585.36	3064	3635	1931
173	2486.43	2482.51	2542.13	3213	3529	1978
357	2438.96	2416.63	2470.81	3079	3547	2095
157	2989.81	2941.41	2996.97	2859	2976	1953
287	2730.55	2710.91	2739.71	3338	3118	2102
253	2939.15	2948.59	2971.52	3207	3090	1689
148	2394.72	2403.21	2433.65	3274	3557	2054
76	2655.07	2692.92	2654.91	2874	2987	2004
221	2465.44	2506.34	2496.99	3031	3351	1865
245	2311.16	2323.85	2376.13	3095	3796	2066
397	2431.37	2408.39	2418.39	3074	3474	2390
55	2714.83	2652.17	2678.41	3432	3373	2136
100	2439.94	2438.42	2426.72	3166	3752	2184
254	2411.94	2388.29	2422.45	3353	3483	2139
52	2539.0	2522.44	2522.2	2942	3506	2160



(a) Comparación del tiempo de rescate



(b) Comparación del tiempo de ejecución, en ms

De nuevo vemos como apenas hay diferencia entre los costes, pero tras una

Paired t-test
data: time1 and time2
t = 0.20768, df = 29, p-value = 0.8369
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-9.889203 12.124537
sample estimates: mean of the differences 1.117667
Conclusión: no hay diferencia real entre inicializaciones 1 y 2 respecto al tiempo de rescate

Paired t-test
data: time1 and time3
t = -2.8255, df = 29, p-value = 0.008456
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-28.738670 -4.603997
sample estimates: mean of the differences -16.67133
Conclusión: La inicialización 1 lleva a mejor tiempo de rescate que la inicialización 3.

Paired t-test
data: time2 and time3
t = -3.3899, df = 29, p-value = 0.002033
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-28.521569 -7.056431 sample estimates: mean of the differences -17.789
Conclusión: La inicialización 2 lleva a mejor tiempo de rescate que la inicialización 3.

En resumen, los métodos 1 y 2 resultan en tiempos de rescate muy parecidos y mejores que los de 3.

En consecuencia, usaremos la inicialización 1, pues es visiblemente más rápida que la 2, en el gráfico de tiempo de ejecución.

4.4 Experimento 3

Debemos ajustar ahora los parámetros para realizar la búsqueda con el algoritmo de Simulated annealing:

Conocimientos previos: hay que tener en cuenta la función de energía y la de la probabilidad de aceptar estados peores, así como el comportamiento de Simulated Annealing.

Objetivo: Optimizar SA (parámetros: steps, stiter, k y lambda), partiendo del escenario y condiciones de los experimentos anteriores, esto es, 100 grupos, 5 centros, 1 helicóptero por centro, los operadores mov, swap y switch sin restricciones y el método de inicialización 1 (dentro de los códigos, inicialización 0).

Observación: Pueden existir parámetros que consigan un coste mucho menor. En particular, hay que tener en cuenta que k y lambda influyen en la función de energía y la probabilidad de aceptar un estado peor, y se intuye que dependiendo de la heurística podrían ser mejores unos valores de otros; steps y stiter determinan las iteraciones y su distribución, y se intuye que la función va a tender a ser mejor como más steps tenga. La influencia de stiter dependerá de los parámetros k y lambda.

Hipótesis: Los parámetros son indiferentes (H0) o influyen decisivamente.

Método:

- Seguiremos el ejemplo del enunciado adaptándolo a las circunstancias de nuestro problema.
- Dada la complejidad del espacio de la función (4 variables), empezaremos con k y lambda, ya que el número de iteraciones (steps y stiter) dependerán de estos parámetros. La importancia de stiter depende de los valores de estos parámetros.
- Fijaremos, pues, unos valores determinados de steps y stiter. Steps deberá ser suficientemente grande para que haya convergencia. El valor por defecto de la clase es 10000, y es el usado en el ejemplo.

  Lo que haremos será realizar un paso previo, que será realizar ejecuciones con distintos valores de steps, stiter = 100 (ver explicación a continuación), y los valores por defecto de k (20) y lambda (0.005), para comprobar que haya suficientes iteraciones para converger. En este paso previo también ejecutaremos HC para los mismos casos, como referencia.

  El método de ejecución será el mismo que en los otros pasos y está explicado posteriormente. Fijaremos así el steps incial, pero igualmente es arbitrario y volveremos a steps cuando lo optimicemos al final. Stiter será arbitrario al principio, fijaremos el valor de 100 (el usado por defecto en la clase de Simulated Annealing y divisor de steps).
- Dada la dificultad de realizar una búsqueda exhaustiva, probaremos una serie de valores distintos considerados extremos para k y lambda, calcularemos las medias, observaremos el gráfico 3D resultante y tendremos una idea de hacia donde se minimiza la función. Concretamente, el gráfico nos mostrará los costes en tiempo según los distintos valores de k y lambda.
- Volveremos a repetir acotando los valores a la parte donde se minimice el coste, y tendremos en cuenta los resultados en caso de que sean significativos (esto es, las medias sean suficientemente diferenciadas).
- Teniendo en cuenta lo que tarda en converger y la forma del gráfico (si observamos que para un solo parámetro determinado la solución es buena, por ejemplo), será indicado fijar uno de los dos parámetros y ver cómo varía la función variando el otro.
- Las ejecuciones se harán del siguiente modo:
  - Elegiremos 10 semillas aleatorias, una para cada réplica.
  - Realizaremos 3 repeticiones de cada ejecución para una misma semilla ya que, al ser un método estocástico, existe una cierta variabilidad. No fueron necesarias más repeticiones debido a que la variabilidad es pequeña y además el coste en tiempo de ejecución es considerable.

- Guardaremos los parámetros k y lambda usados en la réplica, el coste en tiempo (de la heurística), y el tiempo de ejecución. La métrica para medir cómo de buena es una solución será en todo momento su coste en tiempo (de la heurística).
- Generaremos una tabla con las medias de las distintas semillas para cada par de valores k y lambda.
- La gráfica 3D se dibujará partiendo de la tabla anterior.
- Una vez hayamos escogido k y lambda, optimizaremos stiter utilizando el mismo método que antes (parámetros fijados, 10 semillas, 3 repeticiones...), aunque esta vez nos moveremos en una única dimensión. Es posible que sea poco importante (como explica el enunciado). Lo comprobaremos con el gráfico en 2 dimensiones y la tabla.
- Finalmente, optimizaremos steps. La función irá subiendo y bajando, inicialmente mucho y al final poco, pero no va a tener sentido poner un nombre de iteraciones absurdamente grande para conseguir mejores de una unidad, por ejemplo. Habiendo fijado los demás tres parámetros, variaremos valores "razonables" de steps y comparemos con la ejecución de HC (que también guardaremos, esta vez), como referencia.

Dada la gran cantidad de datos generados en este experimento, solo incluiremos las tablas y los gráficos. Los datos usados serán accesibles en un documento aparte.

Los resultados de cada paso fueron los siguientes:

Ejecutamos la función experiment3_prevSteps(), con distintos rangos de steps (entre 5k y 150k). Llegamos a la conclusión, observando la tablas de medias y el gráfico ampliado de que 20k será un número suficientemente grande para que haya convergencia (a partir de 20k prácticamente no varía y en cambio aumenta el tiempo de ejecución) en los experimentos. Volveremos a steps una vez tengamos k, lambda y stiter.

steps	costSA	costHC
5000	2758.782667	2653.544
10000	2666.029	2653.44
20000	2639.389	2653.544
30000	2635.437667	2653.544
60000	2631.751667	2653.544
120000	2637.463333	2653.544
150000	2637.049667	2653.544

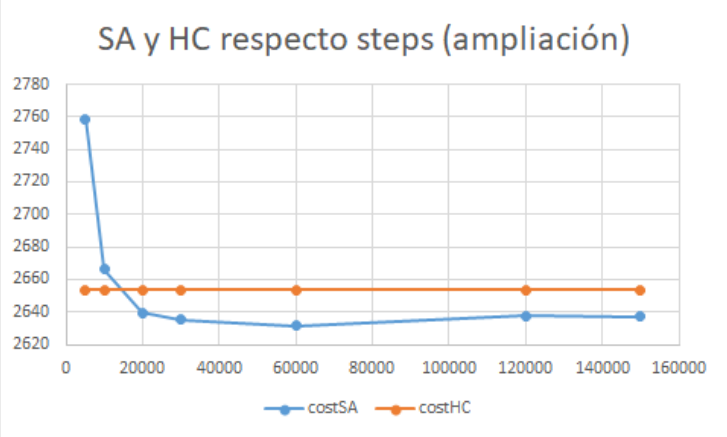


Figure 4: Gráfico ampliado de la comparación previa de los valores finales del heurístico de SA y HC, con respecto al número de pasos de SA (valores de SA por defecto).

Procedemos a ejecutar la función experiment3_lambdaK2(), con steps ya en 20k. Primera versión: valores extremos de lambdas = 1,0.01,0.0001; ks = 1,5,25,125, los usados en el ejemplo. Observando el gráfico 3D de medias detectamos un mínimo en k = 5, lambda = 0.0001. Cabe destacar que las diferencias en el gráfico son muy significativas (el máximo es 3622.81, el mínimo es 2582.56). Para todo lambda = 1 parece haber malas soluciones. Con 0.01 obtenemos buenos resultados y a medida que va aumentando k parece ir minimizándose lentamente. Sin embargo, alrededor de lambda = 0.0001 y k = 5 parece haber un mínimo.

lambda	k	value
1	1	3599.553
1	5	3619.289333
1	25	3602.340333
1	125	3622.807667
0.01	1	2613.098667
0.01	5	2610.945
0.01	25	2610.245333
0.01	125	2607.778
0.0001	1	2597.017667
0.0001	5	2582.562667
0.0001	25	2715.493667
0.0001	125	3283.568333

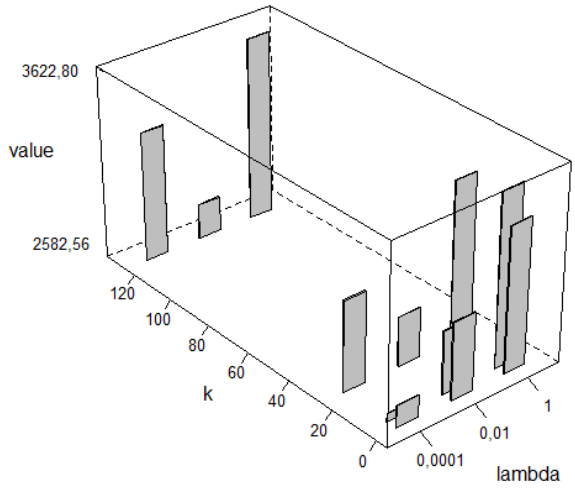


Figure 5: Valores finales del heurístico con respecto a los parámetros k y lambda.

Repetiremos el procedimiento anterior acotando valores alrededor del último caso, tal y como sugiere el enunciado: lambdas = 0.001, 0.0001,0.00001; ks = 1,5,10,15. Notamos que aún hay cierta variabilidad (rango de 500 aproximadamente), pero menos que antes, la desviación estándar se ha reducido. Curiosamente, el mínimo de esta ejecución es exactamente el mismo que antes: k = 5, lambda = 0.0001. No nos parece oportuno continuar acotando porque las diferencias cada vez son menores y el mínimo encontrado parece ser un buen valor.

lambda	k	value
0.001	1	2627.355667
0.001	5	2621.755
0.001	10	2620.081667
0.001	15	2614.149
0.0001	1	2604.548333
0.0001	5	2593.021667
0.0001	10	2617.342
0.0001	15	2652.516333
0.00001	1	2606.39
0.00001	5	2717.610333
0.00001	10	2916.38
0.00001	15	3095.872

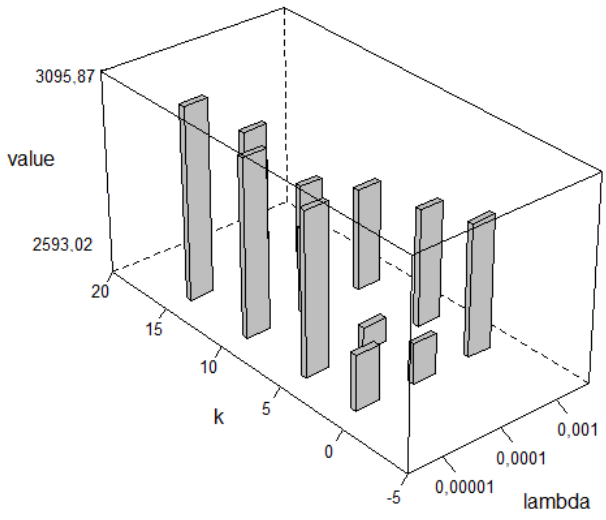


Figure 6: Valores finales del heurístico con respecto a los parámetros k y lambda, acotando los valores.

Así pues, ya tenemos que k = 5 y lambda = 0.0001. Ahora procederemos a realizar las ejecuciones (experiment3_stiter2()) con distintos stiters (todos ellos divisores de steps): 50, 80, 100, 125, 160, 200, 250, 400, 500, 625, 800, 1000, 1250. Como ya advertía la guía, podría pasar que dependiendo de los parámetros k y lambda escogidos stiter fuera poco relevante. De hecho, nos es imposible detectar ninguna tendencia, las medias dan valores muy parecidos. Para una media de 2407.67, la desviación estándar es de apenas 2.46. No se aprecia ninguna tendencia en particular. El mínimo es 160, así que vamos a coger este valor, aunque perfectamente nos podríamos haber quedado con el 100 porque la diferencia es del orden de unidades. Lo que pasa es que para probar los nuevos steps habrá que ir con cuidado de que sean divisibles por 160.

stiter	value
50	2410.959
80	2409.833333
100	2408.945
125	2408.779333
160	2402.707333
200	2405.309667
250	2405.475333
400	2405.723333
500	2407.122333
625	2408.243
800	2407.444333
1000	2411.491
1250	2407.637

Finalmente, volvemos a steps. Esencialmente repetiremos el paso previo, con SA y HC, para asegurarnos de que converja correctamente y sea mejor que HC con los nuevos valores. También usaremos la función experiment3_prevSteps(), fijando los parámetros ya determinados y con

posibles valores de steps de 5120,10400,20000,30400,40000. Observando la tabla de medias y su correspondiente gráfica ampliada, notamos que en 5120 y 10400 steps de SA, HC sigue siendo mejor, pero de 20k para adelante no hay duda alguna de que SA se comporta mejor. Además observamos que SA se va estabilizando. De 30400 a 40k varía del orden de menos de una unidad. Concretamente, alrededor de 3 décimas. De 20k a 30,4k el cambio es de alrededor de 11 unidades.

Cabe decir que, aunque no es nuestro objetivo en este experimento concreto, el factor tiempo de ejecución es siempre una limitación a tener en mente para poder ejecutar estos algoritmos y no tiene sentido realizar un número gigantesco de pasos para obtener una mejora tan pequeña (del orden de minutos, en este problema).

steps	costSA	costHC
5120	2675.645667	2509.477
10400	2545.084667	2509.477
20000	2489.163333	2509.477
30400	2476.440333	2509.477
40000	2476.773	2509.477

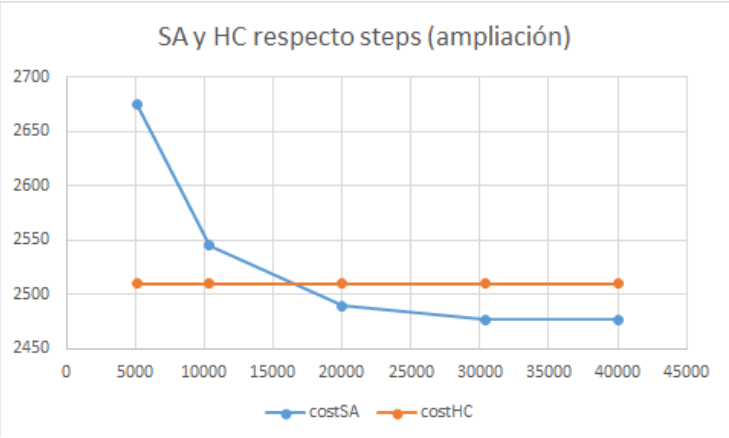


Figure 7: Gráfico ampliado de la comparación final de los valores finales del heurístico de SA y HC, con respecto al número de pasos de SA (parámetros de SA ya optimizados).

Experimentalmente hemos optimizado la función de búsqueda de SA a los parámetros:

steps = 30400, stiter = 160, k = 5 y lambda = 0.0001.

Incrementado mucho steps parece que solo conseguiríamos mejorar la función del orden de las unidades. a costa de aumentar el tiempo de ejecución. Refutamos H0, aunque algunos parámetros han resultado ser mucho más determinantes que otros (k y lambda mucho más importantes que stiter).

4.5 Experimento 4

En el experimento 4 vamos a comprobar cómo cambia el comportamiento de SA y HC cuando aumentamos la complejidad del problema. Concretamente, para el caso en que el número de centros y el de grupos aumente proporcionalmente (proporción 5 centros por cada 100 grupos),

vamos a mirar cómo cambia el tiempo de ejecución.

Cabe destacar que en estas condiciones y para el caso de HC, se hace menos viable ir corriendo ejecuciones, por el elevado tiempo de ejecución y RAM requerida.

El resumen del experimento es el siguiente:

Observación: Aumentando el número de centros y grupos HC seguramente generará muchos más sucesores, lo que debería aumentar un aumento importante del tiempo de ejecución. Por otro lado, en principio SA ejecuta un número de pasos fijo, pero es posible que con más complejidad no tenga tiempo de converger. El valor final del heurístico podría aumentar porque hay más grupos a transportar y el problema es más difícil.
Objetivo: Determinar cómo evoluciona el tiempo de ejecución de HC y SA cuando aumentamos proporcionalmente el número de centros y grupos a transportar.
Hipótesis: El tiempo de ejecución de HC aumenta según el número de centros y grupos y SA permanece constante (H0), o hay otros comportamientos de los tiempos de ejecución.
Método: Generaremos problemas con seeds aleatorias (debido al tiempo de ejecución de casi 15 minutos para HC, lo haremos de una en una y colocaremos tan solo el valor medio, al contrario del resto de experimentos). Ejecutaremos ambos algoritmos (SA haciendo 3 repeticiones y calculando la media) para los siguientes parámetros: (nCentros, nGrupos) : (5,100), (10,200) y (15,300). Mediremos el tiempo de ejecución y tiempo de rescate obtenidos con SA y HC para los mismos pares de valores, y esta será nuestra medida principal. Como ya se ha explicado, en este experimento va a ser menos viable ir corriendo ejecuciones. Comparemos los valores de los tiempos de ejecución y los expondremos en gráficos para observar las tendencias.

nCentros	nGrupos	costSA	costHC	exectimeSA	exectimeHC
5	100	2319.9500	2335.59	687.0	4247.0
10	200	4475.7966	4471.01	255.0	85278.0
15	300	6692.69	6426.36	325.0	851790.0

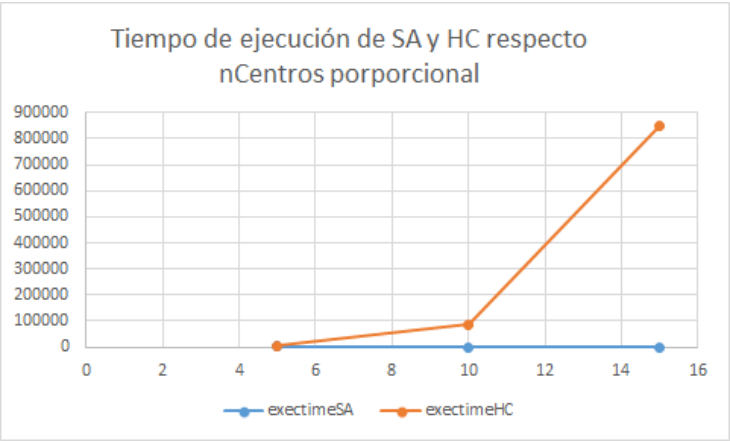


Figure 8: Gráfico de la comparación del tiempo de ejecución de SA (azul) y HC (naranja) según el número de centros, aumentados proporcionalmente al número de grupos.

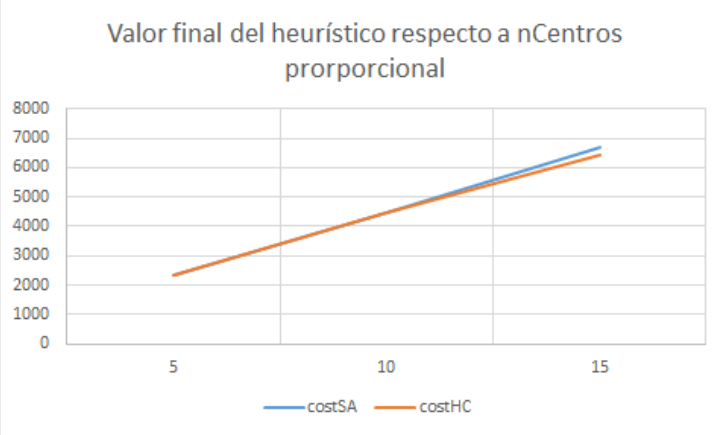


Figure 9: Gráfico de la comparación del tiempo de rescate total de SA (azul) y HC (naranja) según el número de centros, aumentados proporcionalmente al número de grupos.

Conclusión: Mirando el gráfico confirmamos H0: HC aumenta exponencialmente en tiempo de ejecución y SA permanece constante, por las razones que habíamos aventurado en las observaciones. Fijándonos en la tabla, vemos que la primera ejecución de SA es algo más alta, y nuestra explicación es que esto es debido a optimizaciones de la JVM (durante la primera ejecución compila el bytecode y llena la cache, por ejemplo), pero esto no afecta a la conclusión.

4.5.1 Recálculo de los parámetros de Simulated Annealing

En el experimento 4, hemos observado que aumentando la complejidad del problema (concretamente, aumentando el número de centros y de grupos proporcionalmente), SA sigue dando resultados buenos, pero ligeramente peores a HC. Es posible que con el aumento de la complejidad SA ya no tenga tiempo de converger, o incluso que el conjunto de parámetros haya dejado de ser el óptimo. Cabe recordar que se había optimizado para las condiciones del experimento 3, que ya no son las mismas.

Así, se considera oportuno repetir el experimento 3 para comprobar si SA sigue siendo óptima. Lo haremos de la forma explicada en el experimento 3, pero con alguna modificación para adaptar el método a las nuevas circunstancias. De entrada, no será viable ir comparando con HC, por el aumento de complejidad. Por el otro, tendremos que tener en cuenta ejecuciones con más centros y grupos.

Observación: Para ejecuciones de SA con más complejidad (experimento 4), SA da valores ligeramente superiores que HC. Es posible que con el aumento de complejidad SA no tenga tiempo de converger. o que el conjunto de parámetros entero haya dejado de ser el óptimo.
Objetivo: Optimizar los parámetros de SA partiendo de una mayor complejidad más centros y grupos.
Hipótesis: Los parámetros óptimos determinados en el experimento 3 vuelven a ser los óptimos (H0) o con el aumento de complejidad han cambiado.
Método: El explicado en el experimento 3, pero sin ir comparando con HC en el paso previo.

Sin más dilación, los resultados:

Con los parámetros por defecto de SA, corremos varias ejecuciones (con sus respectivas seeds) aumentando nGrupos y nCentros con la proporción 100:5. Esta vez probamos valores de steps más altos: 1000,5000,10000,30000,60000,120000,500000,1000000,2000000.

Nos fijamos en los dos casos extremos: nCentros = 5, nGrupos = 100; nCentros = 35, nGrupos = 700 y calculamos una tabla de medias y mostramos gráficamente los resultados.

steps	costSA
1000	3304.538
5000	2588.536
10000	2495.48
30000	2463.836
60000	2460.876
120000	2454.606
500000	2469.584
1000000	2463.916
2000000	2470.9

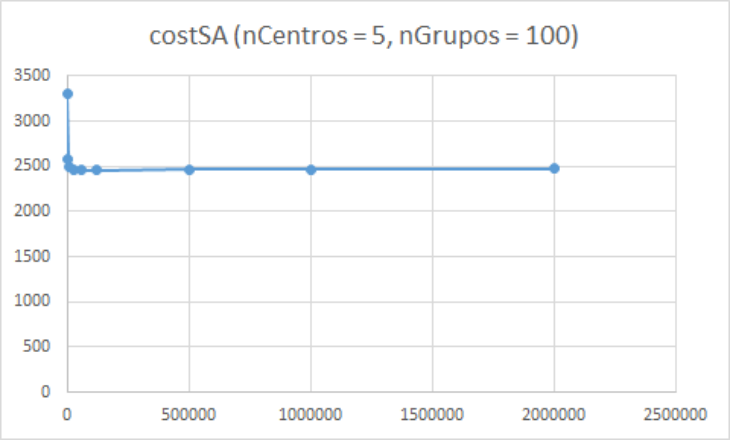


Figure 10: Valores finales del heurístico de SA según el número de steps, con los parámetros por defecto y con 5 centros y 100 grupos.

steps	costSA
1000	32414.446
5000	24767.228
10000	21378.194
30000	16828.48
60000	15091.096
120000	14298.48
500000	14121.758
1000000	14119.37
20000000	14134.286

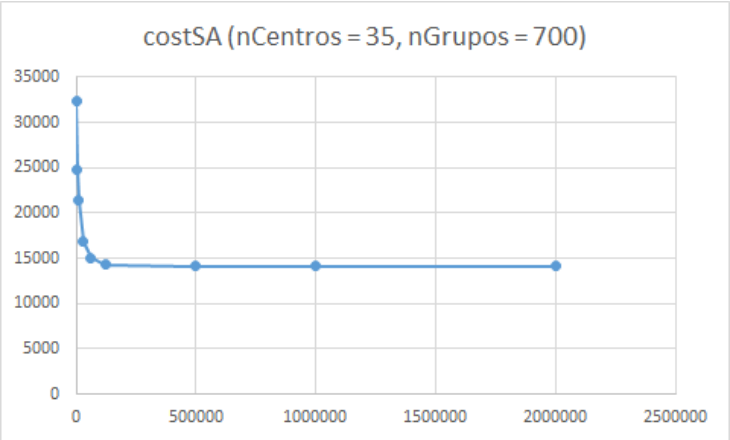


Figure 11: Valores finales del heurístico de SA según el número de steps, con los parámetros por defecto y con 35 centros y 700 grupos.

Confirmamos que con 30k da tiempo a converger en el caso inicial, pero en el grande aún varía demasiado. En 500k ya observamos una cierta estabilidad, y aún es un número razonable de pasos, así que tomaremos 500k para la experimentación.

Fijando steps = 500k, stiter = 100 (por defecto), y tomando valores extremos de k y lambda como en el experimento 3, expuestos sobre un gráfico 3D para buscar cómo varía el coste según k y lambda. Las condiciones serán las del caso de nCentros = 35, nGrupos = 700. Valores de lambda: 1,0.01,0.0001. Valores de k: 1,5,25,125.

lambda	k	value
1	1	34017.504
1	5	33921.362
1	25	33907.674
1	125	33804.586
0.01	1	14749.298
0.01	5	14739.092
0.01	25	14785.302
0.01	125	14820.21
0.0001	1	13684.862
0.0001	5	13674.054
0.0001	25	13675.986
0.0001	125	13646.514

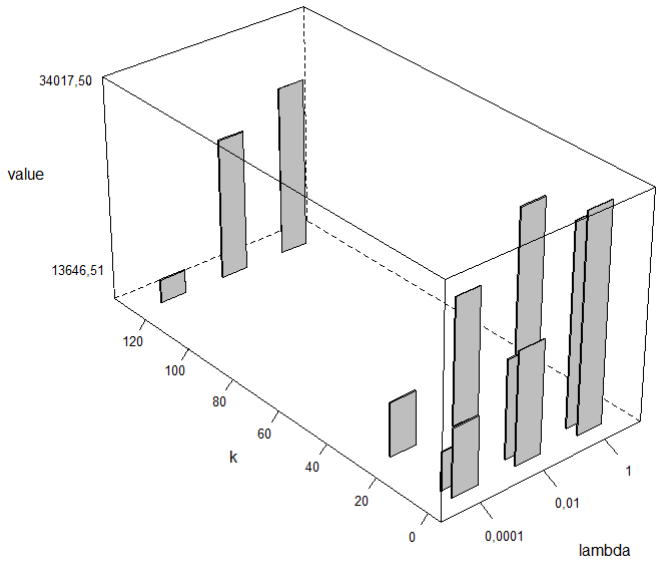


Figure 12: Valores finales del heurístico con respecto a los parámetros k y lambda.

Los resultados tienen una desviación alta: rango de entre 13600 y 34000 aproximadamente. Observamos que los parámetros k y lambda determinados en el experimento 3 siguen dando buenos resultados, lo que es muy buena señal. Concretamente, para k = 5 y lambda = 0.0001, tenemos el segundo valor más pequeño. Sin embargo, a diferencia del experimento 3, en que había más irregularidades pero habíamos detectado una zona que minimizaba en una esquina de los valores probados, esta vez observamos que los resultados son buenos para todo lambda = 0.0001 (en el 3, con esta lambda con K alta daba un resultado no muy bueno). Así, esta vez fijaremos lambda = 0.0001 y variaremos k. Escogeremos el mínimo.

Valores de lambda: 0.0001. Valores de k: 5,[25-200] de cinco en cinco.

k	value
5	13624.654
25	13626.556
50	13608.418
75	13616.688
100	13596.564
125	13612.45
150	13601.304
175	13594.928
200	13627.266

Observamos una variabilidad pequeña y sin que parezca haber tendencia remarcable alguna. Simplemente, escogeremos el mínimo, k = 175. Notamos, sin embargo, que está muy cerca del valor escogido en el experimento 3, así que también nos podríamos haber quedado con él. De nuevo, cabe recordar que el componente aleatorio se ha intentado controlar ejecutando varias ejecuciones con seeds distintas y, para una misma seed, corriendo alguna repetición para tomar la media.

Una vez fijadas k y lambda (k = 175, lambda = 0.0001), nos fijaremos en stiter. Cabe recordar que este parámetro es posible que sea más o menos importante según los parámetros k y lambda fijados (en el 3 ha resultado ser poco relevante).

stiter	value
50	13767.754
100	13754.932
125	13759.006
200	13751.634
250	13740.328
500	13741.828
800	13747.856
1000	13735.916
1500	13737.598

De nuevo, fijándonos en la tabla de medias el stiter parece muy poco relevante en este contexto. La función varia muy poco y no se aprecia ninguna tendencia. Puestos a escoger vamos a tomar el que permite conseguir el mínimo, que es 1000.

Finalmente, vamos a volver a medir diferencias con el número de pasos para ver cómo converge con los nuevos parámetros. La media de 500k, el valor inicial que habíamos escogido para optimizar los demás parámetros, comprobamos que SA ya se ha estabilizado, aunque en 1M y 2M de pasos sigue bajando algo. Si cuadriplicar (aproximadamente) el tiempo de ejecución no es un problema y queremos afinar mucho la optimización, podríamos quedarnos con el valor de 2M, que sigue siendo medianamente razonable, pero en 500k ya ha convergido.

steps	costSA
1000	33458.656
100000	14463.79
500000	13745.266
1000000	13723.23
2000000	13692.368

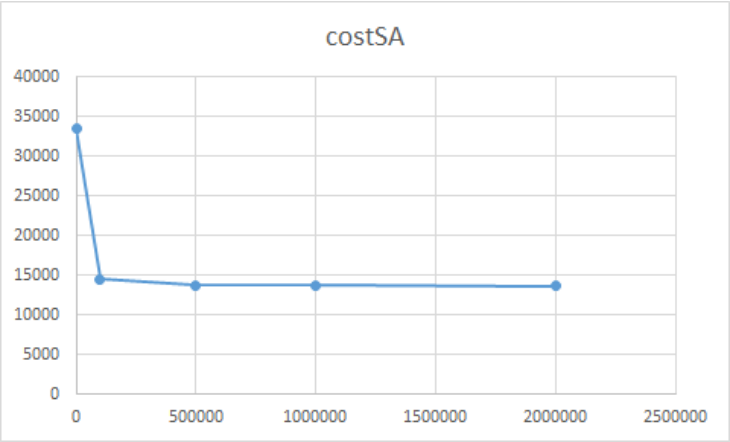


Figure 13: Valores finales del heurístico de SA, con respecto al número de pasos de SA (parámetros ya optimizados).

CONCLUSIÓN:

Esta vez hemos determinado los siguientes parámetros óptimos: lambda = 0.0001, k = 125, stiter = 1000, steps = 500k.

En este sentido, refutamos la hipótesis nula porque no hemos determinados los mismos parámetros, pero cabe decir que lambda ha permanecido igual y que con k = 5 los resultados también eran bastante buenos. Respecto a la relevancia, lambda es muy relevante, porque la función varía enormemente según este parámetro.

Los parámetros k y, sobre todo, stiter son poco determinantes. Respecto a steps, en 500k apreciamos que SA tiene tiempo de converger, pero si quisiéramos afinar la optimización y no nos importara el tiempo de ejecución podríamos llegar a 2M.

4.6 Experimento 5

En el experimento 5, nos interesa estudiar el comportamiento de la IA con más complejidad, pero esta vez solo para el caso de HC y aislando los dos parámetros a aumentar. El experimento está dividido en dos apartados: el 5.1 y el 5.2.

En el experimento 5.1, aumentaremos solamente el número de grupos, de 50 en 50, hasta apreciar la tendencia. En este caso, volveremos a tener dificultades para correr las ejecuciones, debido a la RAM (1GB) y el tiempo de ejecución.

En el escenario 5.2, haremos lo propio con el número de centros.

4.6.1 Experimento 5.1

Observación: Si aumentamos el número de grupos pero mantenemos el número de centros constante, igualmente HC seguirá creando muchos sucesores, porque habrá muchos grupos a transportar.
Objetivo: Estudiar el comportamiento del tiempo de ejecución de HC cuando aumentamos el número de grupos a transportar pero mantenemos constante el número de centros (y helicópteros).
Hipótesis: El tiempo de ejecución aumentará con respecto al número de grupos (H0), o no aumentará.
Método: Ejecutaremos instancias del problema manteniendo constante el número de centros a 5 y aumentando el número de grupos de 50 en 50, obteniendo el tiempo de ejecución. A partir del gráfico y la tabla sacaremos las conclusiones.

nCentros	nGrupos	value	exectime
5	100	2526.11	8330
5	150	3687.46	28869
5	200	4843.26	230776
5	250	6042.71	593624

Figure 14: Tiempo de ejecución de HC respecto al número de grupos, con el número de centros constante a 5.

Conclusión: Si bien no hemos podido usar una n demasiado grande, pues el tiempo escalaba extraordinariamente rápido, sí que podemos apreciar una tendencia exponencial con relación al numero de grupos, en la figura 14.

4.6.2 Experimento 5.2

Observación: Si aumentamos el número de centros pero mantenemos constante el número de grupos, parece que HC generará también más sucesores, pero no muchos más, porque los grupos a transportar son los mismos y no hay tantas combinaciones (habrá un momento en el que todos los grupos ya estarán asignados eficientemente). El valor del heurístico debería disminuir porque a más centros (ergo, más helicópteros) menos tiempo se debería tardar en transportar los grupos.
Objetivo: Estudiar el comportamiento de HC referido al tiempo de ejecución si aumentemos el número de centros pero mantenemos constante el número de grupos.
Hipótesis: El tiempo de ejecución aumenta con respecto al número de centros (H0), o no aumenta.
Método: Elegiremos 10 semillas aleatorias para cada tamaño de problema. Mantendremos constante el número de grupos. Aumentaremos el número de grupos de 5 en 5, hasta 50. Para cada instancia concreta ejecutaremos 3 repeticiones. Elaboraremos una tabla de medias y lo expondremos en tablas para llegar a la conclusión.

nCentros	nGrupos	value	exectime
5	100	2616.308	3325.9
10	100	2323.777	3694.3
15	100	2199.415	4242.3
20	1006	2106.454	4552.9
25	100	2026.041	5042.6
30	100	1967.998	5685.4
35	100	1921.363	6009.5
40	100	1871.64	6684.8
45	100	1825.219	7149.3
50	100	1795.581	7457.3

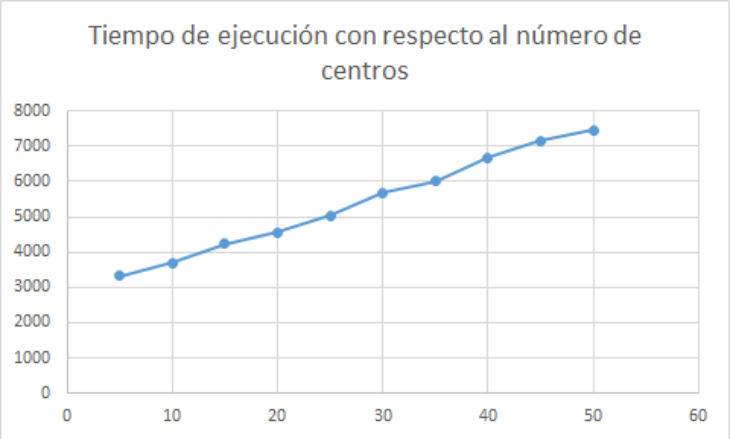


Figure 15: Tiempo de ejecución de HC para número de centros creciente, con el número de grupos constante a 100.

Conclusión: Confirmamos H0: el tiempo de ejecución aumenta con respecto al número de centros. Concretamente, lo hace de forma lineal, en cierta forma como intuíamos con las observaciones. Es decir, el tiempo de ejecución aumenta solo linealmente, muchísimo menos que en el escenario 1 (y que en el experimento 4).

4.7 Experimento 6

Para el experimento 6 usaremos valores de H diferentes de 1. Es decir, en cada centro habrá más de un helicóptero.

Creemos que es bastante obvio que la calidad de las soluciones aumentará incrementando dicho valor, pues contra más helicópteros, más podremos repartir los vuelos y ahorrarnos tiempos de descanso.

Sin embargo, al haber más helicópteros, el espacio de soluciones aumentará como está detallado en la sección 3.1. Dicho esto, también esperamos más sucesores a cada paso del algoritmo Hill Climbing, así que el tiempo de ejecución potencialmente aumentará.

Por otro lado, no creemos intercambiable el número H y el número C, aunque el número de

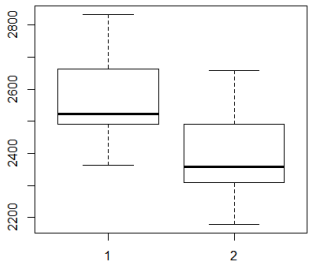
helicópteros sea el mismo. Al haber más centros, potencialmente deberíamos tener más grupos cerca de un centro, así que nos interesa contrastar si para el mismo número de helicópteros totales es mejor tener más centros o es lo mismo.

4.7.1 Experimento 6.1

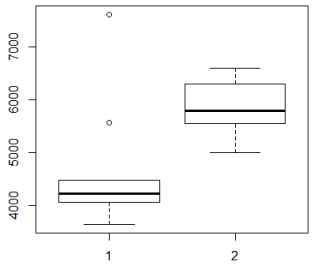
Observación: El parámetro H puede afectar a la calidad de la solución y al tiempo de ejecución.
Objetivo: Estudiar el comportamiento de HC referido al tiempo de ejecución y de rescate si aumentemos el de helicópteros por centro.
Hipótesis: El tiempo de ejecución y la calidad de la solución son constantes respecto a H (H0) o no.
Método: Elegiremos 10 semillas aleatorias y generaremos el problema con los parámetros (C=5, G=100), pero haremos dos ejecuciones diferentes: Una con H=1 y otra con H=5. Obtendremos los cuatro tiempos y los contrastaremos.

Los resultados obtenidos son los siguientes:

seeds	time1	time2	exec1	exec2
341	2363.43	2178.62	7617	5548
43	2662.18	2489.59	4233	6595
170	2490.47	2362.04	5568	5830
131	2506.25	2315.81	4058	5745
324	2588.68	2415.57	4416	6053
397	2431.37	2270.3	4200	5732
7	2540.04	2352.86	4140	5480
198	2502.36	2308.36	4478	6370
220	2673.89	2516.72	3788	6295
56	2832.67	2658.85	3637	5002



(a) Comparación del tiempo de rescate, Hill climbing, H=1(izq.) vs H=5(der.)



(b) Comparación del tiempo de ejecución, Hill climbing, H=1(izq.) vs H=5(der.)

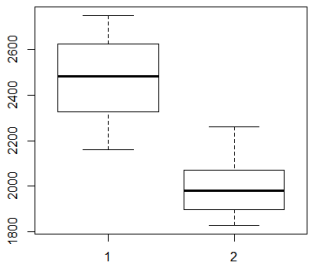
Visualmente vemos como nuestra primera intuición era correcta.

4.7.2 Experimento 6.2

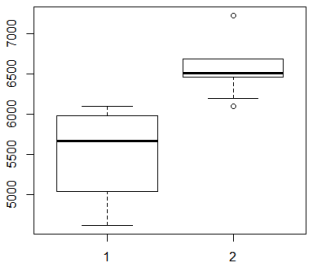
Observación: La distribución de helicópteros entre el parámetro H y el parámetro C puede dar soluciones de calidad diferente, aún manteniendo el número de helicópteros total constante.
Objetivo: Estudiar el comportamiento del tiempo de ejecución y de rescate según la distribución de helicópteros.
Hipótesis: Para valor constante de helicópteros totales, la calidad y el tiempo no cambian aunque sí que lo haga la distribución de helicópteros entre centros (H0) o son diferentes.
Método: Elegiremos 10 semillas aleatorias y generaremos el problema dos veces, con los parámetros (H=5, C=5, G=100) y (H=1, C=25, G=100). Nótese que el número de helicópteros total es 25 en ambos problemas. Obtendremos los cuatro tiempos y los contrastaremos.

Los resultados obtenidos son los siguientes:

seeds	time1	time2	exec1	exec2
26	2544.17	2147.65	4732	6461
119	2543.5	1958.54	5040	6106
165	2750.72	1975.82	5117	6486
260	2625.44	1886.88	4621	6479
113	2161.67	1827.2	5740	6539
201	2326.58	2071.01	5982	6665
393	2694.23	2260.16	5598	6692
379	2408.84	1982.62	6037	7231
204	2425.53	2003.23	6099	6198
149	2192.61	1895.46	5742	6695



(a) Comparación del tiempo de rescate, Hill climbing, set 1 de parámetros (izq.) vs set 2 (der.)



(b) Comparación del tiempo de ejecución, Hill climbing, set 1 de parámetros (izq.) vs set 2 (der.)

De nuevo, nuestra intuición inicial se confirma y vemos que la distribución sí que importa para la calidad.

Un mayor número de centros es mucho mejor que un mayor número de helicópteros por centro.

4.8 Experimento 7

Introduciremos ahora la segunda heurística, y con ella, cambios generales sobre el problema (ya mencionados en secciones anteriores):

- Usaremos la tercera representación del problema, en que tenemos un vector auxiliar con el tiempo de rescate del último grupo de heridos de cada helicóptero.
- Añadiremos el operador swapOrder, efectivamente cambiando del conjunto de operadores 3 al 6.
- Usaremos la función heurística 2.

Obviamente los resultados a esperar serán diferentes con respecto a los experimentos anteriores. Ahora no solo optimizamos el tiempo de búsqueda total sino que nos importa también el tiempo de rescate de heridos.

Balancearemos estos dos criterios con un parámetro λ, lo cual da lugar a muchas heurísticas diferentes con el cambio de dicho valor.

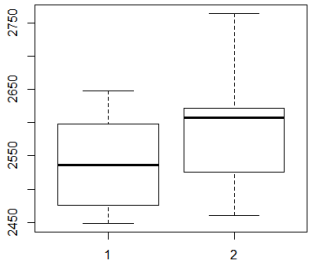
4.8.1 Experimento 7.1

Comparamos ahora el resultado de comparar la heurística 1 con la heurística 2.1 (heurística 2 con $\lambda = 1$).

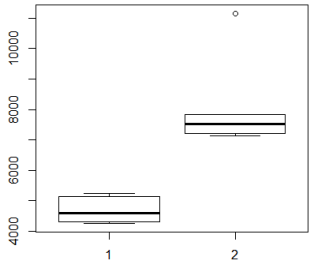
Formalmente:

Observación	Al añadir un segundo valor a optimizar a la heurística el valor del primero puede cambiar
Planteamiento	Compararemos la heurística 1 con la heurística 2
Hipótesis	El tiempo de rescate en ambos casos es igual (H0) En caso de no igualdad, el primer valor en la primera heurística es mejor que en la segunda heurística
Método	• Usaremos 10 semillas aleatorias • Para cada semilla, ejecutamos HC dos veces, una con cada función heurística y obtenemos valores h1, h2 y el tiempo de ejecución, donde h1 y h2 son el tiempo de rescate total de cada función • Para cada semilla, ejecutaremos SA 6 veces, 3 para cada versión de la heurística, cambiando la semilla aleatoria para SA y calculando valores medios. • Según el escenario con 100 grupos, 5 centros y 1 helicóptero por centro.

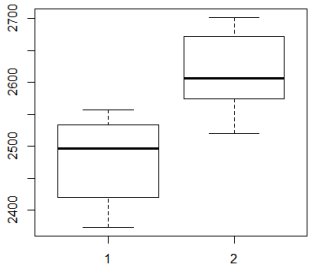
Tenemos 4 boxplots, dos que comparan **la parte de tiempo de rescate total** y 2 que comparan el tiempo de ejecución, para los algoritmos de Hill Climbing y Simulated Annealing.



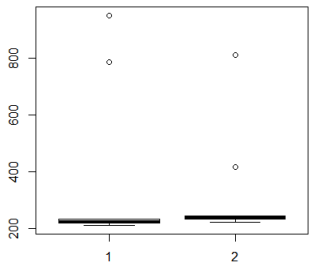
(a) Comparación del tiempo de rescate, Hill climbing, h1(izq.) vs h2.1(der.)



(b) Comparación del tiempo de ejecución, Hill climbing, h1(izq.) vs h2.1(der.)



(a) Comparación del tiempo de rescate, SA, h1(izq.) vs h2.1(der.)



(b) Comparación del tiempo de ejecución, SA, h1(izq.) vs h2.1(der.)

No podemos distinguir bien en el gráfico la diferencia entre tiempos de ejecución del SA, así que realizamos un t-test para comprobar si hay diferencia:

data: SAExec_Old and SAExec1
t = 0.73545, df = 9, p-value = 0.4808
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-83.65761 164.25761
sample estimates: mean of the differences 40.3
Conclusión: No hay diferencia real entre los tiempos de ejecución para las dos heurísticas usando SA.

En resumen, vemos que en HC el primer criterio a optimizar (tiempo de rescate) y el tiempo de ejecución empeoran ligeramente y que en SA solo el primer criterio a optimizar empeora.

4.8.2 Experimento 7.2

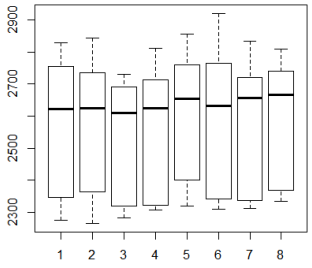
Nos gustaría ver cómo evoluciona el tiempo de rescate total, el tiempo de rescate de heridos y la heurística total y el tiempo de ejecución para valores de λ diferentes.

Para ello, nos disponemos a repetir diversas ejecuciones con valores de $\lambda = 1, 2, 4, 8... = 2^{n-1}$ para $n \geq 1$

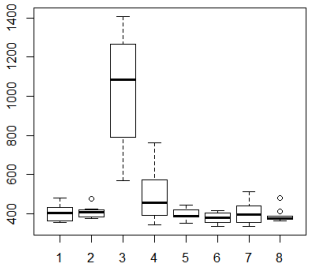
Observación	Nos interesa ver cómo afecta el parámetro λ a la heurística
Planteamiento	Realizaremos diversas ejecuciones para valores de $\lambda = 2^{n-1}$ para n de 1 a 8
Hipótesis	Los resultados son similares (H0) se dan cambios notables entre valores de λ
Método	• Elegimos 10 semillas aleatorias • Para cada semilla, ejecutamos el algoritmo HC con cada valor de λ y obtenemos la heurística total, el tiempo de rescate total, el tiempo de rescate de heridos y el tiempo de ejecución. • Para cada semilla, ejecutamos el algoritmo SA 3 veces con cada valor de λ y obtenemos la heurística total, el tiempo de rescate total, el tiempo de rescate de heridos y el tiempo de ejecución, todo valores medios. • El escenario del problema considera 100 grupos, 5 centros y 1 helicóptero por centro, usando el conjunto de operadores 3

A continuación se dan los gráficos obtenidos con el experimento detallado anteriormente:

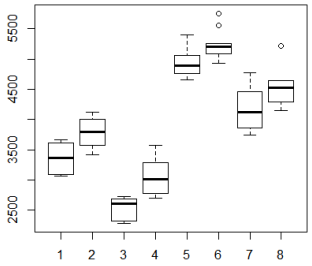
Para Hill Climbing:



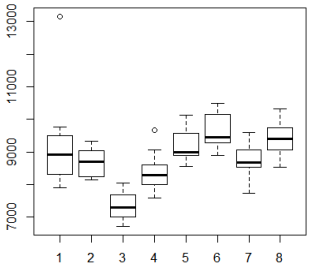
(a) Tiempo de rescate total, HC



(b) Tiempo de rescate de heridos, HC

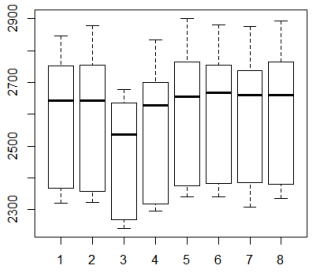


(c) Heurística total, HC

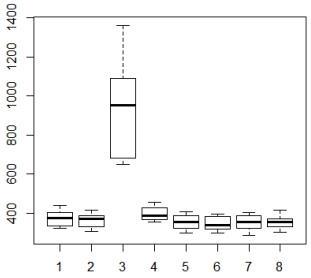


(d) Tiempo de ejecución, HC

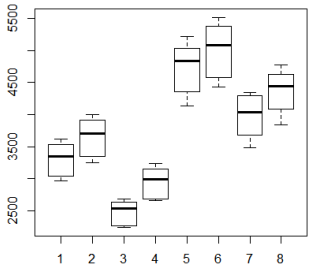
Para Simulated Annealing:



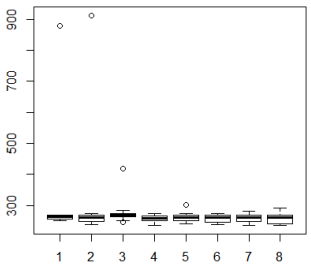
(a) Tiempo de rescate total, SA



(b) Tiempo de rescate de heridos, SA



(c) Heurística total, SA



(d) Tiempo de ejecución, SA

Observando los boxplots donde el axis inferior representa N en la formula de ponderación heurística $= h1 + h2 * 2^{n-1}$, siendo h1 el tiempo total de rescate y h2 el tiempo de rescate de heridos, vemos que tanto HC como SA cambian de forma similar conforme aumentamos N.

El tiempo total de rescate se mantiene relativamente constante, al igual que el tiempo de rescate de heridos sin ponderar y el tiempo de ejecución.

Notamos un outlier para N=3, donde el tiempo de rescate de heridos es casi 3 veces peor que en cualquier otro valor de N, la cual se mantiene aun repitiendo el experimento con seeds diferentes.

La heurística total ($h1+h2*2^{n-1}$) muestra un patrón para cada cuatro valores de N, por ejemplo entre N=1 y N=2 hT empeora, pero en N=3 se vuelve mejor que N=1 y en N=4 empeora respecto a N=3, pero sigue siendo ligeramente mejor que N=1.

Aparte de este patrón, hT parece aumentar linealmente con N, en vez de exponencialmente como inicialmente suponíamos. Esto se verá más claramente en el experimento 7.3.

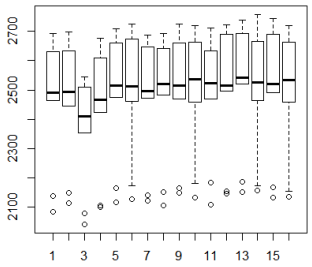
4.8.3 Experimento 7.3

Debido a los resultados anteriores, queremos examinar con mayor detalle la progresión del heurístico $h1 + h2 * 2^{n-1}$ con respecto a n.

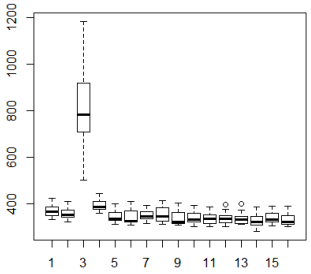
Como la búsqueda Hill Climbing toma demasiado tiempo en ejecutarse, solo usaremos SA para

este experimento, con exactamente el mismo procedimiento de antes y n = 1..16.

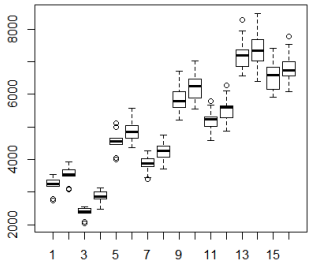
Exponemos los resultados:



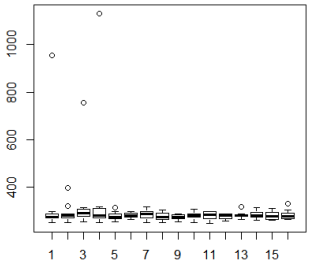
(a) Tiempo de rescate total, SA



(b) Tiempo de rescate de heridos, SA



(c) Heurística total, SA



(d) Tiempo de ejecución, SA

De nuevo observamos el outlier en n=3 y la inesperada linealidad de la heurística total, que esperábamos crecería exponencialmente.

4.8.4 Experimento 7.4

Nos referimos ahora a la sección 3.4. Aparentemente, podríamos tener un problema si se da la siguiente situación:

- El helicóptero h es el último en rescatar a su último grupo de heridos
- El vuelo i del helicóptero h es el último vuelo con heridos.
- El vuelo i del helicóptero h tiene dos grupos de heridos.
- El vuelo i-1 del helicóptero h tiene un grupo de heridos.
- Estamos en la situación óptima de tiempo de rescate total, y cualquier cambio la empeorará
- Podemos optimizar muchísimo el tiempo de rescate de heridos si priorizamos ambos vuelos

Podemos ver que, en esta situación, el algoritmo se detendrá. No sacará ningún grupo del último vuelo, pues empeoraría el tiempo de rescate total sin mejorar el tiempo de rescate de heridos. No moverá el vuelo i-1 hacia adelante porque no aporta nada. No intercambiará el vuelo i-1 con el i pues tampoco hará nada.

Llegado este punto, nos planteamos si sería interesante añadir un 'castigo' a la función heurística, en función del número de vuelos del último helicóptero.

Si hacemos esto, y añadimos un castigo de, pongamos 10 minutos, por cada grupo de heridos en el último vuelo de heridos, en la situación anterior:

El algoritmo puede intentar mover un grupo de heridos del último vuelo a otro, siempre que no empeore el tiempo de rescate en más de 10 minutos. Sin embargo, antes intentará intercambiar el vuelo i y el i-1, pues no empeora el tiempo de rescate total y reduce el castigo. Al hacerlo, dejará disponible un intercambio entre este vuelo y uno de no heridos cualquiera, reduciendo el tiempo de rescate de heridos en la siguiente operación.

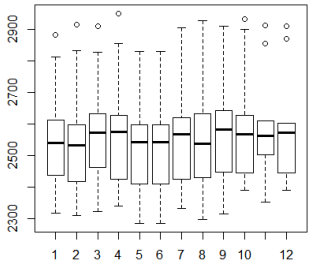
Por ello, nos planteamos ahora las dos funciones heurísticas:

- **Heurístico 2** $h1 + h2 * 2^{n-1}$, donde h1 es el tiempo de rescate total, h2 es el tiempo de rescate del último grupo de heridos y 2^{n-1} es λ
- **Heurístico 3** $(h1 + (h2 + c) * 2^{n-1})$, donde h1, h2 y 2^{n-1} es lo mismo que en la otra función heurística, y c es nuestro castigo. c toma por valor $c = 10*$ #numero de grupos de heridos en el último vuelo de heridos (global)

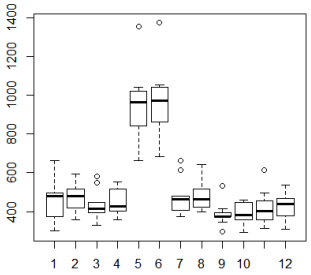
Formalmente, tenemos el siguiente experimento:

Observación: La función heurística estándar (2) tiene potencial de quedarse atascada en ciertas situaciones.
Planteamiento: Compararemos dicha función con una diferente que podría potencialmente resolver dichos atascos, bajo las mismas condiciones que los otros experimentos del apartado 7
Hipótesis: No hay diferencia entre el Heurístico 2 y el Heurístico 3 (H0), o sí que la hay
Método: Obtendremos 10 semillas y generaremos 10 problemas diferentes con parámetros (C=5, H=1, G=100). No sólo esto, sino que generaremos 6 lambdas diferentes (2^{n-1} n = 1..6). Para cada uno de estos problemas, ejecutaremos HC dos veces, una con cada heurístico; y SA 6 veces, 3 con cada heurístico.

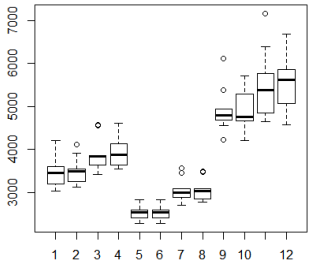
Debido al descomunal tamaño de los datos obtenidos, solo presentamos las gráficas resultado:



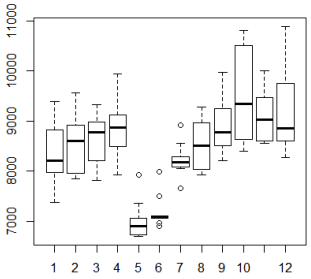
(a) Tiempo de rescate total, HC



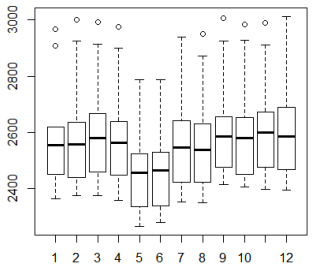
(b) Tiempo de rescate de heridos, HC



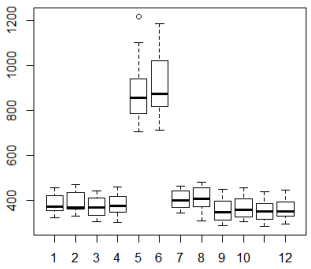
(c) Heurística total, HC



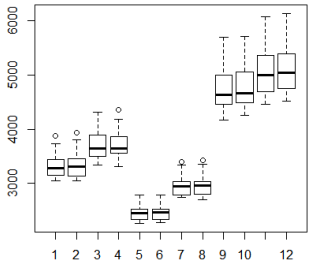
(d) Tiempo de ejecución, HC



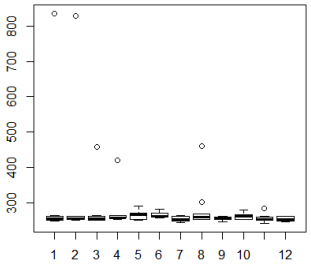
(a) Tiempo de rescate total, SA



(b) Tiempo de rescate de heridos, SA



(c) Heurística total, SA



(d) Tiempo de ejecución, SA

En las gráficas tenemos, por cada pareja de columnas (por ejemplo col 1 y col 2) a la izquierda el heurístico 2 (sin castigo) y a la derecha el heurístico 3.

Observamos que no hay diferencia significativa entre ambas excepto en el tiempo de ejecución de HC, en que el heurístico 3 toma mayor tiempo en terminar la ejecución.

4.9 Experimento 8

Experimento especial, tiempo de rescate y de ejecución en el siguiente escenario:

C=5, H=1, G=100, grupos y centros distribuidos según la semilla 1234 para el generador aleatorio.

Usando el algoritmo Hill Climbing, con 10 repeticiones para obtener el tiempo de ejecución medio. (El tiempo de rescate no cambia debido al determinismo de Hill Climbing).

Resultados:

Tiempo de rescate promedio: 2538.33; Tiempo de ejecución promedio: 2274 ms

5 Conclusiones

Tras observar los resultados, tanto experimentales como de desarrollo, podemos concluir que usar las técnicas de búsqueda local fue una decisión acertada.

Es más, mediante su uso nos han aportado conocimiento sobre el problema que no nos habíamos planteado inicialmente, y que luego empíricamente han demostrado saber más. Por ejemplo en experimentos como el 7.4, hemos visto que nuestro planteamiento sobre funciones heurísticas no era del todo acertado, y en los experimentos sobre operadores nuestra intuición sobre restringir hubiera resultado en mayor tiempo de rescate, y vidas potencialmente perdidas.

Tras el estudio tanto sobre las características generales del problema como sobre los detalles que se han ido descubriendo a lo largo de la implementación, hemos descubierto datos que aportan una mayor comprensión sobre la logística de desastres naturales, y cómo planificar los rescates de personas en una situación como la representada por el problema.

No sólo eso, pues aun trabajando sobre un problema muy simplificado, extender o modificar la definición del problema para añadir complejidad no debería ser demasiado difícil a no ser que los cambios sean extensos o afecten a la esencia del problema. Por ejemplo, cambiar el área de búsqueda, la velocidad de los helicópteros y las condiciones de carga no costaría demasiado. De hecho, para nuestro conjunto final de operadores, también podríamos cambiar el número de grupos por vuelo con relativa facilidad, o añadir una restricción sobre la cantidad de km que puede recorrer un helicóptero.

Cabe añadir que encontramos problemas a la hora de experimentar, ya que debido a la naturaleza de los algoritmos y el problema, muchos de los experimentos requerían de un tiempo de ejecución extenso, lo cuál limitó el número de repeticiones que pudimos realizar. Sin embargo, que con las repeticiones que realizamos tenemos bastante certeza de la fiabilidad de nuestros datos.

Encontramos además ciertos resultados sorprendentes en nuestra experimentación. Esperábamos un tiempo de ejecución muchísimo mayor para el algoritmo de Simulated Annealing. Sin embargo, descubrimos que empíricamente no tan solo es muchísimo más rápido, sino que escala mucho mejor que Hill Climbing, dando resultados ligeramente peores en determinados casos pero generalmente comparables.

Con relación a la comprensión de Simulated Annealing, sin tener en cuenta nuestro problema concreto, hemos dado con que la optimización de los parámetros de dicho algoritmo es ya de por sí un potencial problema de búsqueda, ya que requiere optimizar sobre un espacio de cerca grande sin demasiada información. No sólo eso, sino que con cada cambio en el tamaño espacio de búsqueda, se requiere de un recálculo de los parámetros del algoritmo para mantener su optimalidad.