INTELIGENCIA ARTIFICIAL
PRÁCTICA 1: BÚSQUEDA LOCAL

Miguel Ángel Merino
Pol Renau
Carla Lara

14-4-2018
2017/2018 - Q2

Índice general

Capítulo 1

Parte descriptiva

1.1. Identificación del problema

Para resolver esta práctica tenemos que analizar el problema del que partimos y definir ciertos criterios que nos ayudarán a resolverlo con más facilidad y eficacia.

Para empezar, nuestro principal problema es rescatar a todas las personas que se encuentran en apuros en una zona en concreto debido a un desastre natural. Estas personas están divididas en varios grupos situados en diferentes posiciones de la zona afectada por el desastre. Pero hay un inconveniente, las posiciones en las que se encuentran los grupos son de difícil acceso por varios motivos, es entonces cuando tenemos que actuar con helicópteros de capacidad y velocidad limitadas. Sabemos de antemano la situación, localización de todos los grupos (posiciones en valores enteros) y además el tamaño del espacio afectado por el desastre: **50 x 50 km**. Con lo que nuestro objetivo será encontrar una solución razonable en el mínimo tiempo posible. En la ecuación 1.1 vemos como se calcularán las distancias entre centro y grupo.

$$d(C, G) = \sqrt{(gx - cx)^2 + (gy - cy)^2)} \tag{1.1}$$

Dónde gx y cx son las coordenadas x del centro y grupo, gy y cy son las coordenadas y. Para la distancia entre dos grupos usaremos la misma fórmula pero usando los pares de coordenadas correspondientes a los dos grupos tratados.

Disponemos de un número fijo de centros de rescate con la misma cantidad de helicópteros cada uno. Estos pueden ser enviados al terreno para realizar un rescate con la única limitación de **esperar durante 10 minutos** al volver a realizar otro. Esto nos complica bastante la situación si disponemos de pocos helicópteros y muchos grupos a los que rescatar, por lo que el tiempo que nos supone rescatar a todos los grupos aumenta. Este es el motivo por el que hay que resolver el problema mediante la búsqueda local. Nosotros no deberíamos enviar los helicópteros al azar o allá donde creamos conveniente porque puede que los pasos que damos incrementen demasiado el tiempo de rescate. Por eso necesitamos de una inteligencia artificial

que nos proporcione la mejor solución posible en un tiempo razonable.

La *búsqueda local* nos permite dar este paso, partimos de una solución inicial que consideramos ni mala ni buena y a partir de allí, mediante las funciones heurísticas y algoritmos de búsqueda local, seremos capaces de obtener una buena solución en un tiempo razonable. Los problemas de tipo rescates, como el nuestro, se caracterizan por tener más de una solución. En el problema que nos ocupa no consideramos de lo más importante conseguir una solución muy óptima sino que preferimos obtener una buena solución en el mínimo tiempo posible y es por este motivo que usamos búsqueda local, puesto que hace este trabajo mucho mejor que otros algoritmos como la búsqueda ciega o heurística. Otra ventaja más es que podemos parar el algoritmo cuando queramos, ya sea porque tarda mucho o otro motivo, y aún así obtener una solución.

Evidentemente tenemos otro tipo de restricciones y condiciones a tener en cuenta para generar la mejor solución. Como se ha comentado anteriormente existen una cantidad de grupos a rescatar cada uno en un cierto tipo de condiciones: tenemos grupos con heridos, a los que les asignamos prioridad 1, y luego aquellos que no tienen heridos, asignados a prioridad 2. Los grupos que se han formado, sean de la prioridad que sean, pueden tener de uno hasta doce integrantes.

Nuestros helicópteros tienen que rescatar a todos los grupos a partir de dos criterios diferentes, uno está claro, que siempre que sea posible hay que procurar rescatar primero a aquellos de prioridad 1. El segundo es en el que los helicópteros no pueden hacer viajes en paralelo, sino que solo puede haber un helicóptero haciendo un rescate en el terreno a la vez. Nosotros nos hemos centrado en el segundo principalmente, aunque hay algún experimento en el que tenemos en cuenta el primer criterio. El inconveniente de los grupos de **prioridad 1** es que, al recoger a personas heridas, estas tardan una media de **dos minutos** en subir al helicóptero, a diferencia de los de **prioridad 2** que solo necesitan **1 minuto** para subir a él.

Otro problema añadido es la capacidad limitada de los helicópteros, esta se establece a un **máximo de 15 personas o 3 grupos**, lo que hace que se necesiten más helicópteros o más salidas de rescate de un mismo helicóptero. Aunque podamos rescatar como máximo a 15 personas, no podemos dejar un grupo a medio rescatar, es decir, que o se rescatan todos sus miembros o no se recoge a ese grupo en ese rescate. Además de que la velocidad que alcanzan los helicópteros también tiene un límite fijado a **100km/h**.

Con todas esas condiciones, nosotros intentaremos optimizar los rescates a partir del tiempo empleado en ellos. Queremos que la misión se realice en el mínimo espacio de tiempo posible de modo que la inteligencia artificial que programaremos se basará en calcular distancias en función de la velocidad de los helicópteros, tener en cuenta el tiempo que tardan en subir las personas heridas y las no heridas y por último el tiempo de reposo de estos helicópteros.

El *objetivo* final del programa será mostrar la mejor ruta posible: para cada helicóptero habrá una lista de los rescates que hace y los grupos recogidos en cada salida por orden de modo que se optimice el tiempo de rescate.

Las características del problema que tenemos basándonos en las condiciones explicadas anteriormente son las que siguen:

- **Número de Centros:** Sabiendo la cantidad de centros de los que disponemos, podremos tener en cuenta de cuántos helicópteros disponemos para realizar los rescates.
- **Coordenadas de cada Centro:** En cada centro tendremos un solo helicóptero, este es el que realiza los rescates por lo tanto nos interesa saber desde qué posición parte para calcular la distancia y tiempo con el grupo al que va a rescatar.
- **Número de grupos totales:** Hay que rescatar a todos los grupos que se encuentran en el espacio afectado por el desastre natural, de modo que es importante saber la cantidad total de grupos a los que hay que socorrer.
- **Posición de cada grupo:** Con el objetivo de conseguir rescatar a todos los grupos en poco tiempo, hay que conocer su situación y así priorizar qué helicóptero realizará su rescate para tener un tiempo esperado.
- **Número total de personas que puede abastecer un helicóptero:** Esto nos limita la cantidad de personas que se pueden salvar en un solo rescate de un helicóptero.
- **Número de grupos máximos por rescate:** Cada rescate no puede salvar más de una cantidad limitada de grupos. Esto determinará también el número de salidas de rescate que hace cada helicóptero.
- **Velocidad máxima del helicóptero:** Nos limita el tiempo en el que vamos a rescatar los diferentes grupos.
- **Tiempo de espera entre dos rescates de un mismo helicóptero:** Los helicópteros, al reposar después de un rescate, provocan que el tiempo de rescate total incremente.

Por lo que se puede observar, hay muchísimas características a tener en cuenta para generar la solución al problema, pero para representar un estado muchas de ellas no son necesarias puesto que son restricciones y no elementos que queremos que el estado solución muestre. Los siguientes elementos son los que queremos que se vean representados en los estados que generemos:

- **Helicóptero/s:** Los rescates se hacen mediante helicópteros. Estos tienen que estar presentes en el estado solución para poder indicar que helicóptero realiza que rescate y a que grupos salva. Vienen definidos por el centro de helicópteros del que inician su salida.
- **Rescate/s:** Por lo dicho anteriormente, se intuye que debemos saber que rescates se realizan y cuantos rescates realiza cada helicóptero. Cada rescate tiene que estar presente en un helicóptero y no puede estar vacío.
- **Grupo/s:** Nuestro objetivo es recatar a todos los grupos del área afectada por el desastre, por lo tanto es lógico adivinar que todos ellos tienen que estar presentes en los estados generados. Ningún grupo puede quedarse fuera puesto que entonces la solucion seria incorrecta. Cada grupo tiene que estar presente en un rescate.

1.2. Estado del problema y representación

Para poder trabajar en el problema de rescates de grupos mediante helicópteros, el primer paso a hacer es definir la estructura mediante la cual representaremos el estado en una o varias estructuras de datos que contengan todos los elementos del estado solución y además cumpla con las restricciones impuestas por el problema. Y así poder movernos por los diferentes estados que este puede tomar en un punto determinado de tiempo.

Tanto helicópteros, como rescates y grupos se pueden representar con un identificador de tipo entero. Sabiendo esto procedemos a pensar cual es la mejor estructura. Para dar con la estructura que consideraremos, posteriormente, la más adecuada para este ejercicio hemos hecho una lluvia de ideas de entre las cuales, las estructuras más populares (mejor vistas por los miembros del grupo) han sido las siguientes:

- **Array de arrays de arrays de Integers** donde cada posición del primer array son helicópteros, el segundo array son rescates que contiene un array de como máximo 3 posiciones donde se encuentran los grupos rescatados. Estas 3 posiciones están ocupadas o bien con el identificador del grupo a recatar o con un -1 si no hay grupo a rescatar. De modo que por cada helicóptero existente en nuestro espacio de estados tenemos un array con todos los rescates que este realizara y que grupos va a salvar en cada rescate. Como se puede comprobar, este representa la solución que nos interesa y en cada cambio de estado se seguirá dentro del estado solución, sea mejor o peor que el anterior, puesto que nos aseguraremos de que todos los grupos estén rescatados.

- **Array de arrays de Integers** donde el primer array son helicópteros y el segundo son rescates. Cada rescate del segundo array ocupa tres posiciones de este y en cada posición se indica el índice del grupo que se salva o bien un "-1" en caso que no haya ningún grupo. También cumple con la representación esperada del estado solución ademas de que nos ahorramos una estructura de datos más, un array. Es más eficiente temporalmente que el anterior y de fácil acceso. Como sabemos de antemano que cada centro solo tiene un helicóptero, podemos decir con seguridad que cada índice del primer array corresponderá al indice del centro al que pertenece el helicóptero.

- **Array** de grupos donde cada posición es un grupo que se le asigna un rescate y un **segundo array** de rescates donde en cada posición es un rescate a que se le asigna un helicóptero. Como en los otros casos se puede representar perfectamente todas las soluciones.

En todas las representaciones incluimos una copia estática de los centros y los grupos del problema.

La opción que consideramos mejor para el tipo de problema que nos ocupa es la segunda. Tomamos esta decisión porque la estructura es más simple de entender

que las demás, en cuanto a accesos nos resultan menos complicado que en los otros dos casos que contemplamos. Se pueden representar perfectamente los estados solución y al realizar cualquier operación sobre ese estado, comprobamos siempre que no violamos las restricciones indicadas al principio.

La estructura consta de un array de tamaño cantidad de helicópteros disponibles y por cada uno de ellos un array de tamaño cantidad de rescates por el helicóptero por 3 posibles grupos. El motivo por el cual creemos que el acceso es bastante sencillo es porque podemos acceder a cada helicóptero por índice y buscar sus rescates en menos pasos y e forma más intuitiva que en los otros casos puesto que: en el tercero habría que trabajar con variables auxiliares, y en el primero el algoritmo resultaría demasiado costoso en términos de tiempo. Además que con la representación que implementaremos se puede iterar sobre la misma estructura de datos para realizar todas las operaciones. Para acabar de concretar, la solución que nosotros hemos escogido tiene un coste por espacio de:

$$H * (G/H + 1) = G + H = grupos + helicópteros \tag{1.2}$$

Donde se suma uno debido a que en cada helicóptero tendremos una posición de más.

Este se podría considerar un coste un poco elevado respecto a lo que sería ideal(nos sobra el helicópteros), pero teniendo en cuenta todas las restricciones del enunciado y las expuestas anteriormente, nos resulta imposible pensar en una estructura mejor en cuanto a espacio y términos de acceso, ya pensando en la programación del algoritmo.

El *espacio de búsqueda* es en el cual se encuentran todas las posibles soluciones de nuestro problema, ya sean más eficientes o menos. Los estados que consideramos solución son aquellos que cumplen ciertas características comentadas con anterioridad: todos los grupos tienen que estar rescatados, los rescates tienen un máximo de 15 personas o 3 grupos diferentes.

Cualquier estado que podamos imaginar o crear a partir de la estructura de datos escogida que no cumpla con esas condiciones no será considerado estado solución y por lo tanto no entrará en el espacio de búsqueda. Con lo cual podemos llegar a la conclusión que tenemos un espacio de búsqueda limitado a la cantidad de helicópteros y grupos con los que disponemos para realizar el rescate. Contemplaremos todas las opciones y descartamos aquellas que no sean estado solución.

1.3. Elección y generación del estado inicial

Tenemos que hacer una representación del estado inicial que sea, a la vez, un estado solución, con lo cual, todas las posibilidades que vamos a contemplar tiene que ser soluciones completas. Hay 3 variantes de estados iniciales:

- **Estado inicial Aleatorio:** Estado solucion generado a partir de una función random. El algoritmo itera por los grupos que hay que rescatar y para cada uno de ellos se le indica que helicóptero será el encargado de llevar a cabo su rescate. Una vez iniciado el helicóptero se añadirá el grupo al rescate de este al que quepa, teniendo en cuenta las restricciones, y si no hay sitio en ninguno se genera un nuevo rescate y se encargará de rescatar al grupo. En el caso peor, el coste de hallar la solución inicial es O(grupos*grupos).

- **Estado inicial Ordenado:** Solución generada con un grupo por rescate. El algoritmo itera por los helicópteros generando G/H+1 rescates por helicóptero de tres posiciones cada uno inicialmente con valor "-1" para asegurarnos que aunque haya más helicópteros que grupos a salvar, estos tendrán un rescate asignado como mínimo. Se generan los rescates vacíos del primer helicóptero y se le van asignando un grupo a cada rescate. Cuando se han completado todos los rescates de un helicóptero se va a al siguiente generando sus rescates vacíos y asignando grupos y así sucesivamente hasta que todos los grupos estén en un rescate. En el caso peor, el coste e hallar la solución inicial es O(helicópteros+grupos*3).

- **Estado inicial Greedy:** Solución que se genera en función de proximidades, es decir, sigue el criterio de asignar los grupos a aquellos helicópteros que pertenezcan al centro más próximo a ellos en cuanto a distancia se refiere. Con lo cual la generación del estado solución será realmente buena. Se itera sobre los grupos y mediante una función externa se decide cual es el helicóptero que pertenece al centro más cercano. En esta función externa se recorren todos los centros, por cada centro se recorren todos sus helicópteros y por cada uno de esos helicópteros se recorren sus rescates en búsqueda del mejor rescate y posición al que incluir el grupo. En el caso peor, el coste de hallar la solución inicial es O(grupos*(centros*(helicpotros-por-centro*grupos))).

Cualquier de las posibles generaciones del estado inicial tiene un coste O(centros+grupos) puesto que antes de la generarlos se tienen que inicializar los centros con sus helicópteros y los grupos a rescatar.

La elección de cuál de estos estados iniciales va a ser el usado por los experimentos posteriores se hará en uno de los primeros experimentos para comparar tiempos de ejecución y resultados obtenidos, con el objetivo de quedarnos con la mejor variante de generación de estado inicial y no equivocarnos ahora escogiéndolo a toda prisa.

1.4. Representación y análisis de los operadores

Una vez definida la estructura con la que trabajaremos a partir de ahora, tenemos que decidir cuáles van a ser los operadores que modifiquen nuestra estructura para pasar de un estado solución a otro, sea este mejor o peor que el que le precede.

Para ello hay que tener en cuenta varios factores para que al ejecutar nuestro algoritmo se encuentra la mejor solución posible en un tiempo de ejecución bastante razonable. Nuestros operadores indican todos los posibles caminos que se pueden tomar dado un estado solución y con el objetivo de que este se convierta en un estado solución con unas características bastante favorables. A esto se le llama **factor de ramificación** y es que es variable en función de si usamos Hill Climbing (generamos todos los sucesores y el heurístico se queda con el mejor) o Simulated Annelaing (generamos un estado sucesor de forma aleatoria y el heurístico decide si es lo suficientemente bueno como para quedarse con él).

Hay otra característica a tener en cuenta que puede llegar a convertirse en un problema es si **nuestro operador es capaz de recorrer todo el espacio de soluciones**. Decimos que puede llegar a convertirse en un problema puesto que si no somos capaces de recorrer todo el espacio de soluciones, puede que se pierdan ciertos estados solución con lo que podríamos impedir que se llegara a una solución óptima. O bien que se repitan estados solución con lo que el tiempo de ejecución se vería afectado.

Al principio de todo hemos hecho otra lluvia de ideas con una serie de operadores que creemos que pueden servir y aportar algo útil. Posteriormente los analizaremos y nos quedaremos con aquellos que creemos mejores. Estos operadores son:

- **Cambiar grupo:** Nos planteamos un operador con la misión de poner coger un grupo de un rescate cualquiera y ponerlo en otro rescate. Dado un determinado grupo g perteneciente a un helicóptero, y una determinada posición vacía pos perteneciente a otro o el mismo helicóptero, movemos "g" a la posición "pos". Este operador permite generar todos los estados solución posibles y sin repetidos. Para poder aplicar este operador hay que cumplir las siguientes condiciones de aplicabilidad:

  - No superar capacidad personas máxima rescate, 15.
  - No superar número máximo de grupos por rescate, 3.
  - "pos" tiene que ser posición vacía, su valor = -1.

  En caso de que podamos mover el grupo "g", lo movemos a pos, y ponemos en la posición donde se encontraba "g" ponemos un -1, ya que ese grupo no se encuentra allí después de mover el grupo.

- **Swap grupo:** La misión de este operador es, dado dos grupos diferentes, ya estén en el mismo rescate o diferente, los intercambiamos de posición siempre

que los dos grupos no estén vacíos. Con lo que ahora, donde estaba el primer grupo se encuentra el segundo y viceversa. Este operador nos permite generar todos los estados solución posibles. La condiciones de aplicabilidad de este operador:

  - El nuevo rescate al que van a pertenecer cada grupo no puede tener más de 15 personas.
  - Ambos grupos no pueden tener valor -1.
  - Los grupos a intercambiar no pueden ser iguales.

- **Swap rescate:** Se trata de un operador muy parecido al swap grupo pero en este caso se van a intercambiar rescates. Evidentemente se reduce bastante el espacio de estados solución que contempla este operador puesto que intercambia rescates completos, de ese modo se evita que grupos pertenecientes a un rescate puedan ir a otro que provoque generar un estado solución mejor. Aunque mejore el tiempo de ejecución debido a que tiene menos factor de ramificación, la solución se verá afectada negativamente. Las condiciones de aplicabilidad de este operador son:

  - Ambos rescates deben ser diferentes.
  - Puede haber un rescate vacío pero no ambos.

- **Cambiar rescate:** Se trata de un operador muy parecido al al cambiar grupo solo que en este caso cogemos un rescate de un helicóptero y lo añadimos a otro helicóptero. Volvemos a tener el mismo problema que con el operador anterior, no se pueden representar todos los posibles estados solución y el factor de ramificación se reduce bastante respecto al cambiar grupo. Las condiciones de aplicabilidad de este operador son:

  - El rescate no puede ser vacío.
  - El rescate destino tiene que estar vacío.
  - El rescate destino tiene que estar en un helicóptero diferente al actual para así obtener algún beneficio.

- **Eliminar rescate:** Operador que nos sirve para que cuando un rescate queda vacío, lo eliminamos. Evidentemente este operador por si solo no nos aporta ningún beneficio puesto que no puede explorar todos los posibles estados solución ademas que no mejora el tiempo de rescate, pero junto con otros operadores este podría ser algo útil. Las condiciones de aplicabilidad de este operador son:

  - Las tres posiciones del rescate tienen que estar vacías.

- **Añadir rescate:** Este operador nos crea un rescate vacío al helicóptero que le indiquemos. Estamos en las mismas condiciones que el anterior, no nos aporta beneficios si solo se usa este operador y tampoco permite explorar ningún

estado solución diferente del que tengamos originalmente, de modo que su uso debería ser solo conjunto con otros operadores que requieran de él. Las condiciones de aplicabilidad de este operador son:

  - El nuevo rescate debe crearse al final de la cola de rescates de un helicóptero para no cargarse otros rescates ya creados.

- **Swap helicóptero:** Operador que coge todos los rescates de un helicóptero y los intercambia con los de otro helicóptero. Este operador por sí solo no nos permite generar todos los estados solución posibles, y esto puede llegar a ser un problema. Se podría decir, que comparado con los anteriores, es el que menos estados diferentes visita y el que menos ramificación tiene. Las condiciones de aplicabilidad de este operador son:

  - Los dos helicópteros a intercambiar no pueden ser el mismo.
  - Puede pasar que uno de los dos helicópteros este vacío pero nunca ambos.

- **Cambiar helicóptero:** Operador algo parecido al swap helicóptero. Se cogen todos los rescates de un helicóptero para luego ponerlos en otro, tenga este ya algún rescate o no. Nos encontramos con el mismo problema que con el operador "swap helicóptero", poco factor de ramificación y no es capaz de generar todos los estados solución posibles. Las condiciones de aplicabilidad de este operador son:

  - El helicóptero del cual vamos a cambiar todos sus grupos no puede ser el mismo que el helicóptero destino.
  - Puede haber un helicóptero vacío pero nunca ambos.

Estos son los operadores que hemos pensado que podrían ser útiles para resolver el problema que nos ocupa, pero evidentemente usar los 8 operadores a la vez aumentaría el coste de ejecución una barbaridad y por lo tanto nuestro objetivo de encontrar una buena solución en un tiempo razonable no se cumpliría. Con los cual vamos a descartar desde un principio algunos de esos operadores a los que no les vemos sentido:

Los dos últimos operadores quedan descartados sin pensarlo demasiado, no nos permiten operar por todos los estados solución posibles y no son capaces de generar demasiados estados solución ni tampoco demasiado buenos. También descartamos los operadores de añadir y eliminar rescate, puesto que para las diferentes opciones de generar el estado inicial, estos no nos son muy necesarios si usamos otro tipo de operadores. Para finalizar decidimos dejar a fuera un último operador, el cambiar rescate, puesto que la implicación e que el rescate destino tiene que estar completamente vacío nos genera conflictos y nos hace pensar que, pudiendo usar el "swap rescate", este operador no nos aporta tantos beneficios y no puede generar todos los posibles estados solución, con lo que llegamos a la conclusión de descartar este operador.

Como conclusión, nos quedamos con los siguientes operadores, puesto que pueden llegar a generar todos los estados solución o al menos la gran mayoría de ellos:

- **Swap grupo**
- **Cambiar grupo**
- **Swap rescate**

Ahora mismo no somos capaces de decidir cuál de ellos es mejor de implementar par resolver con eficacia el problema que nos traemos entre manos, con lo que la elección la dejamos a manos del experimento encargado de ello.

Decir, que al haber descartado con tanta rapidez todos aquellos operadores. no hemos necesitado saber con exactitud su factor de ramificación puesto que sin saberlo ya sabíamos que no eran buenos operadores. Con lo cual, en el experimento encargado de decidir cuál de los tres operadores mencionados anteriormente será implementado nos encargaremos de analizar su factor de ramificación también.

1.5. Análisis de la función heurística

Una vez definidos la representación del estado solución la generación del estado solución inicial y los operadores sobre los que vamos a trabajar, analizamos la función heurística. Para poder resolver los dos criterios de solución planteados en el enunciado de la práctica, debemos realizar dos heurísticos distintos, porque el resultado final que debe devolver tiene distintas priorizaciones. La función heurística que resolverá el primer criterio, la llamaremos función heurística 1, y a la que resuelve el segundo, heurística 2.

Función Heurística 1:

El criterio que debe seguir esta función es bastante simple, el objetivo es minimizar la suma de todos los tiempos empleados por los helicópteros en rescatar a todos los grupos, sin considerar que los rescates se puedan realizar en paralelo.

Para seguir este criterio, lo que hemos hecho, es a partir de la estructura definida anteriormente, recorrer todos los helicópteros de manera secuencial, y para cada helicóptero todos sus rescates. Y entonces para cada rescate, calculamos el tiempo de desplazamiento entre el centro que estábamos, hasta el primer grupo de dicho rescate, este lo calculamos mediante la distancia euclidiana entre los dos puntos, y la dividimos entre la velocidad, 1.666666, valor sacado a partir del factor de conversión de Km/H a mM/min, donde la velocidad del helicóptero es 100 Km/h. Una vez calculado el tiempo de desplazamiento, calculamos el tiempo que tarda el grupo a rescatar en subir al helicóptero. Como especificaba el enunciado, los grupos de prioridad 1 tardaban 2 minutos por persona, mientras que los de prioridad 2 tardaban tan solo 1. Con este tiempo calculado, calculamos la distancia euclidiana entre el grupo que estamos, y el siguiente grupo a rescatar, y en caso de que no haya ninguno más con el centro del helicóptero, ya que debe volver al centro para terminar el rescate. Cuando el helicóptero ha terminado un rescate, sumamos diez minutos al tiempo total de rescate. Y seguimos este procedimiento para todos los helicópteros y sus respectivos rescates.

En este heurístico interviene un **único factor**, que es el **tiempo**, esto se debe a que el único objetivo era minimizar el tiempo de rescate, y por eso hemos escogido este heurístico, ya que el algoritmo se encarga de minimizar el valor del heurístico, y el valor de nuestro heurístico es la suma del tiempo total de rescate. El factor tiempo no necesita ponderación, puesto que hay un único factor, esta ponderación afectaría de igual modo a todos los estados solución, y por tanto no encontraríamos ningún cambio respecto a no tenerla.

Este heurístico combina muy bien con cualquier de los 3 operadores que tenemos en este momento, puesto que el heurístico potenciará cualquier cambio realizado por los operadores siempre que implique una mejora del tiempo.

Función Heurística 2:

El criterio de este heurístico es más complicado que el anterior, ya que debe minimizar la suma de todos los tiempos empleados por los helicópteros en rescatar a todos los grupos (ídem que antes) pero minimizando además el tiempo que se tarda en rescatar a todos los grupos de prioridad 1, es decir, minimizar también el tiempo desde el inicio del rescate hasta que el último grupo de prioridad 1 llega a un centro de rescate.

Esta función heurística parte de la heurística 1, con pequeñas modificaciones. Puesto que debemos hacer que el tiempo de los rescates de prioridad 1 se minimice, añadimos un nuevo factor al heurístico, que es la **penalización**. Para valorar la penalización, lo que hemos realizado es que para cada vez que el grupo que estamos rescatando un grupo de prioridad 1, el valor de penalización es igual a penalización + tiempo actual, de este modo los grupos de prioridad 1 deben ir antes en la línea del tiempo, y cuando finalizamos un rescate con algún grupo de prioridad 1, es decir cuando el helicóptero llega al centro, hacemos penalización igual a **penalización+tiempo rescate**, de manera que tenemos en cuenta el tiempo al que llega un grupo de prioridad. Como tansolo la suma del tiempo más la penalización no es un valor heurístico suficiente, debemos añadir una ponderación a este valor, para que tenga más peso que el tiempo de rescate. Este valor no puede ser un valor cualquiera ya que con valores muy pequeños el heurístico no es capaz de assegurar que se cumpla la prioridad principal, rescatar antes a los grupos de prioridad 1. De modo que el valor que hemos asignado antes de realizar la suma del valor heurístico, es penaliza = penaliza*x, donde x toma por valor x ¿= 16, este valor ha sido calculado mediante pruebas en el experimento 7. Decidimos que 16 era el valor límite, ya que a partir de ese valor, los grupos están ordenados, y el valor que da de tiempo de rescate como solución se mantiene estable, a diferència de cuando 'x' tiene valores inferiores.

Este heurístico combina del mismo modo que el heurístico uno, puesto que la penalización es función del tiempo, y la ponderación escogida, hará que primero se ordenen los grupos por prioridad 1 hasta que no se pueda más y después reducirá el tiempo de rescate, manteniendo esa estructura inicial, a no ser que eso haga mejorar el heurístico, que es difícil, ya que la ponderación está hecha expresamente para que no haya vuelta atrás en este paso.

Capítulo 2

Experimentación

2.1. Experimento 1

En este experimento vamos a decidir cuál es el mejor operador de los diferentes que hemos probado. Para poder realizar-lo, hemos fijado el generador de estado inicial para todas las muestras realizadas. El estado inicial nos genera un grupo determinado de rescates(G+H), tamaño fijo, este valor està pensado para que todo grupo tenga un rescate. Cada rescate consta de tres posiciones, y en el momento de generar el estado inicial toda posición de rescate que no esté ocupada por un grupo, serà una posición vacía, con valor -1.

A continuación, calcularemos el factor de ramificación de cada operador, dato que consideramos relevante para llegara una buena conclusión del experimento:

- Operador 1 (mover grupo):
  total-de-pos = 3*(G+H)
  Fact-ram-per-grup = total-de-pasos - G = 2G +3H
  Fact-ram = G* Fact-ram-per-grups = $2G^2 + 3H * G$

- Operador 2 (swap grupos):
  total-de-pos = 3*(G+H)
  Fact-ram-per-grup = total-de-pasos - 1= 3G +3H -1
  Fact-ram = G* Fact-ram-per-grups = G*(3G +3H -1)

- Operador 3 (swap rescates):
  Fact-ram-per-rescat = G+H -1
  Fact-ram = (G+H) * Fact-ram-per-rescat = $G^2 + 2GH + H^2 - G - H$

Hipótesis: el operador 2 será mejor que el resto.

Para tener una muestra completa de los datos, hemos realizado cada ejecución del programa con una seed Random tanto de grupos, como de centros, pero manteniendo el tamaño de estos en 100 y 5, respectivamente. Para cada operador hemos hecho un total de diez recogida de datos. Después realizamos un boxplot para cada

variable a analizar. En este experimento, las variables que hemos analizadas son: el número de pasos, el tiempo del rescate y el tiempo de ejecución del programa.

Boxplots de los resultados obtenidos:

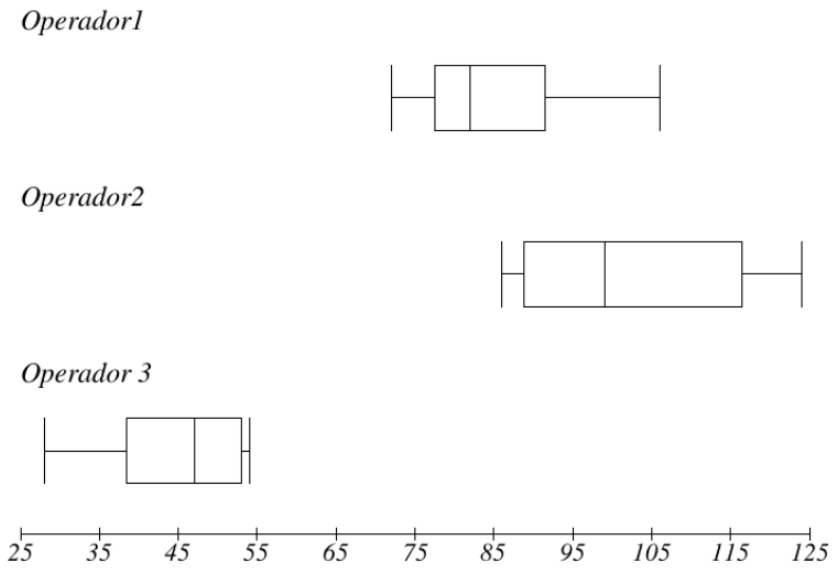


Figura 2.1: Distribución del número de pasos para cada operador.

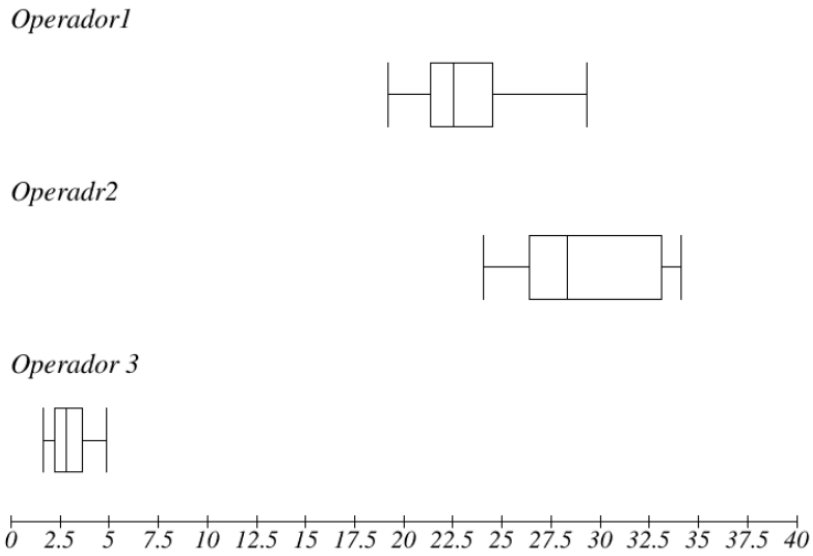


Figura 2.2: Distribución de los tiempos de ejecución en segundos, obtenidos con los diferentes operadores.

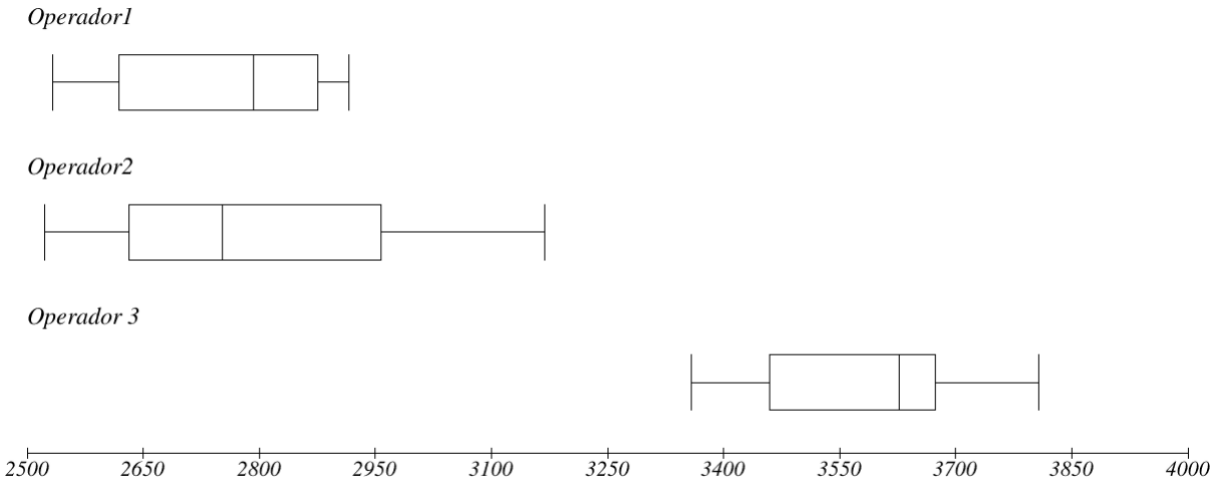


Figura 2.3: Distribución del los tiempos de rescate en minutos, obtenidos con los diferentes operadores.

Como vemos en todos los boxplots, hay uno de los tres operadores que lo podemos descartar sin hacer un análisis más exhaustivo. Ya que en todos es malo, excepto en tiempo de ejecución, pero este no tiene importancia si los otros no se ajustan a parámetros más cercanos a una solución óptima. Así que de ahora en adelante solo analizaremos el operador 1 y el operador 2, para decidir cual seria el mejor para usar.

En el primer boxplot, el de número de pasos, nos interesa que el algoritmo haga bastantes pasos, por eso debemos ver si los operadores hacen un número aceptable de pasos. El operador 1 oscila entre los valores de 65 y 105, con una mediana de 80 pasos. El operador 2 tiene valores entre el 85 y el 125, con una mediana con valor cercana a los 100 pasos. En este primero el operador 2 tiene mejores resultados, es por eso que en la muestra de número de pasos elegimos el operador de Swap grupos.

Evaluamos el tiempo de ejecución en segundos, con el segundo boxplot. El cambio de grupo oscila entre los valores 18 y 29 segundos, con una mediana cerca 22 segundos. Mientras que el operador 2 oscila entre los valores 24 y 33 segundos, con una mediana cercana a los 28 segundos. En esta muestra el operador 1 es mucho mejor, ya que el tiempo en mediana es bastante inferior al tiempo en mediana en el operador 2.

En el tercer boxplot lo que analizaremos es el tiempo de rescate de todos los grupos en forma secuencial, de cada operador. El operador 1 oscila entre 2550 y 2900 minutos, con una mediana de 2800 minutos. Mientras que el operador 2 oscila en un rango de valores más grande, de 2500 a 3150 minutos, pero con una mediana ligeramente inferior 2720 minutos. Aunque cambio de grupo tenga un rango de resultados más pequeño, la mediana es peor que la de swap de grupo. És difícil decidir qué operador es mejor en esta muestra, por eso los dejamos como soluciones equivalentes en este aspecto.

Con ninguno de los boxplots hemos podido demostrar que ninguno de los operadores numero 1 y 3 sean mejor que el operador 2. Por eso vamos a tener en cuenta el factor de ramificación de cada operador. Si los recordamos, veremos que el factor de ramificación del operador 2 es mayor que el del operador 1, además, analizando más a fondo la ramificación de ambos, vemos que el operador cambio de grupo en esencia, genera una ramificación, que es un subconjunto de la del swap de grupo. Ya que un cambio de grupo es lo mismo que hacer swaps con posiciones vacías.

Conclusión:

Se cumple la hipótesis nula del experimento, ya que el operador de swap engloba el espacio de soluciones posibles del operador cambio de grupo, y a más a más tiene un espacio mayor, y a mayor espacio de soluciones, existen más posibilidades de que el algoritmo de Hill Climbing sea más efectivo. A parte de esto, en la mediana de tiempo de rescates total, es mejor la del operador 2, y aunque se mueva en un rango de valores más grande, probablemente debido a que se han ejecutado con diversas seeds, podemos concluir que operador 2 podrá dar mejores resultados la mayoría de veces.

2.2. Experimento 2

Realizamos este experimento para saber qué generador de estado solución será el más adecuado para este programa. En este experimento usaremos el operador de swap, ya que en el experimento anterior hemos demostrado que era el mejor de todos. Como en el experimento anterior fijamos un tamaño de grupos y centros, y las seeds de cada uno serán Random de igual manera.

Hipótesis: el generador de estado inicial Ordered, será el mejor de los tres propuestos.

Para la realización de este experimento, debemos tener en cuenta 5 variables distintas: el número de pasos, el tiempo de ejecución, el tiempo de rescate, el beneficio y el beneficio respecto al tiempo de ejecución. Hemos decidido que sean estas variables las relevantes para este experimento, ya que todas ellas nos ayudan a decidir si una determinada generación del estado inicial, hace que el algoritmo de Hill Climbing sea más productivo y al mismo tiempo nos genere una solución dentro de lo aceptable.

Para generar los datos de la función que crea un estado inicial de forma random, debemos hacer diversas ejecuciones con la misma seed, cinco en nuestro caso, y hacer la mediana de todos los valores. Con esta mediana definimos el valor par esa seed, para este experimento hemos hecho una muestra de 10 datos por variable y creador estado inicial, con lo que hemos recogido una muestra para el random de 50 valores para cada variable.

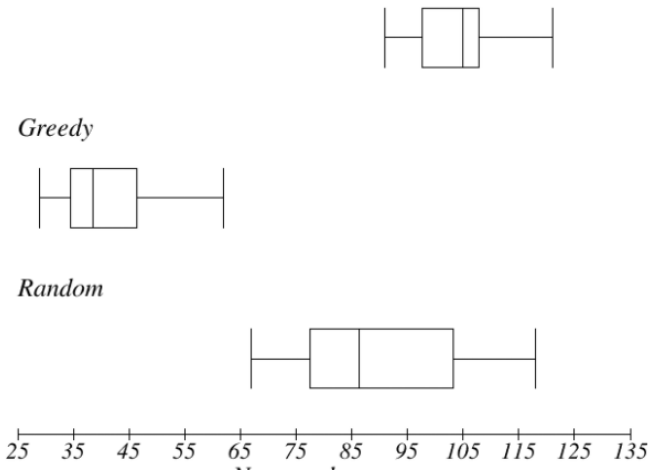


Figura 2.4: Distribución del número de pasos para cada estrategia de generación de estado inicial.

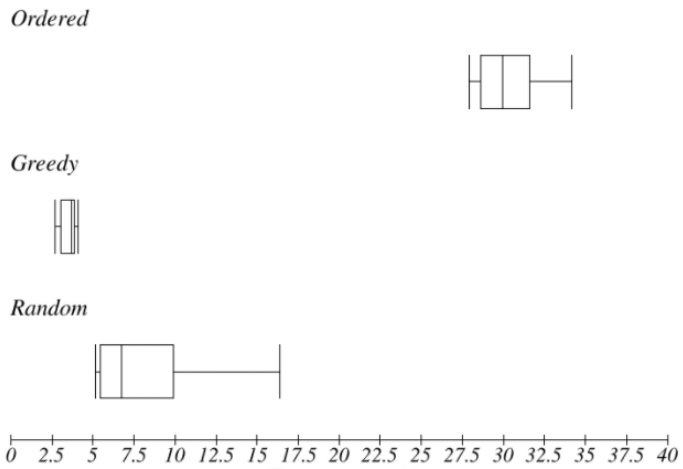


Figura 2.5: Distribución de los tiempos de ejecución en segundos, obtenidos con las diferentes estrategias de generación de estado inicial.

En el primer conjunto de boxplots, vemos una muestra del número de pasos hechos por cada algoritmo. El ordered tiene una oscilación de cantidad de pasos desde 87 hasta 116, con una mediana de 104. El greedy de 26 a 65 y con una mediana de 36 pasos, esta cantidad de pasos es bastante pequeña, teniendo en cuenta que estamos trabajando con 100 grupos y 5 centros. Y con el algoritmo random tenemos una variación de resultados de 66 a 118 pasos con una mediana aproximada de 86 pasos. Para esta primera muestra podemos concluir que el mejor ha sido ordered, ya que tiene mayor probabilidad de expandir más nodos, con lo que más posibilidades de que el Hill Climbing sea más efectivo.

En el segundo, evaluamos la variable tiempo de ejecución. El ordered tiene un tiempo que oscila entre los valores 26.5 y 35 segundos, con una mediana de 30 segundos, este tiempo, aún ser aceptable, no es el más deseable para este programa. El greedy, el que tiene el tiempo más bajo de todos que oscila entre 2.2 y 4 segundos, con una mediana de 3.5 segundos, este tiempo es muy bueno, pero nos hace pensar que el algoritmo no está haciendo gran trabajo con esta inicialización. Finalmente tenemos el Random con tiempos de 5 a 15.75 segundos, y de mediana valor cercano a los 7 segundos, este tiempo es bastante bueno, un tiempo con el que el algoritmo puede hacer una evolución razonable. Para esta muestra podemos decir que el Greedy es el mejor, pero falta ver si es realmente efectivo.

A continuación estudiaremos los boxplots de las variables que serán más decisivas en el momento de demostrar cuál de las inicializaciones es mejor. Ya que sobre estas se refleja el rendimiento del algoritmo con cada una de las inicializaciones.

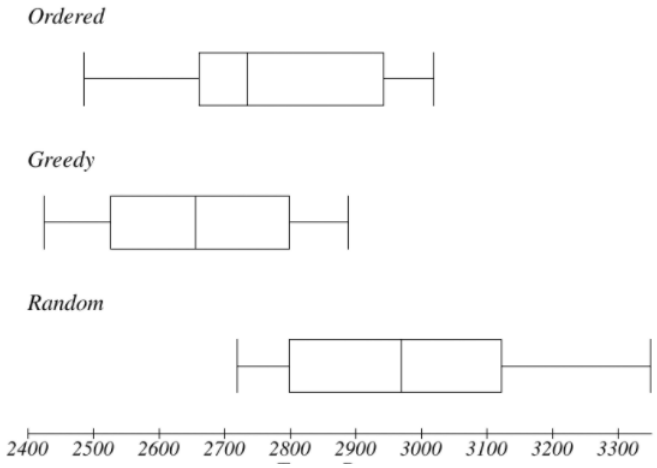


Figura 2.6: Distribución de los tiempos de rescate en minutos, obtenidos con los diferentes operadores.

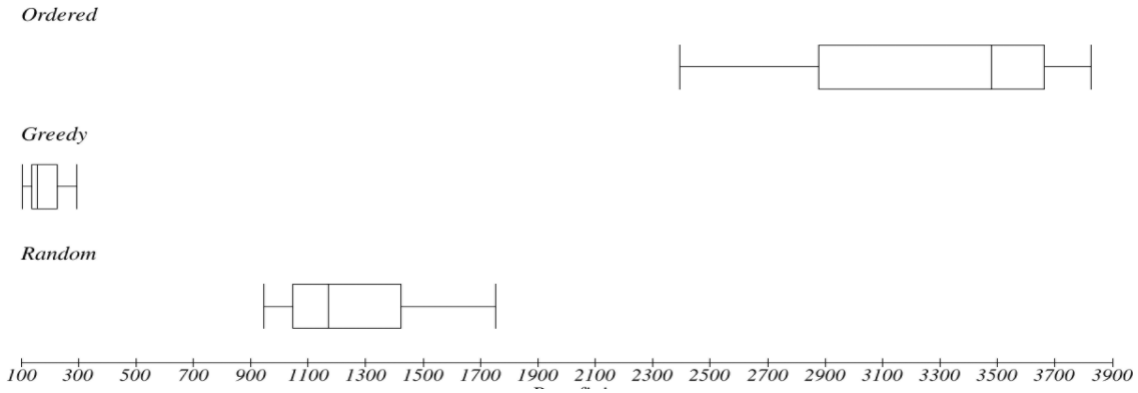


Figura 2.7: Distribución del los beneficios obtenidos con los diferentes operadores.

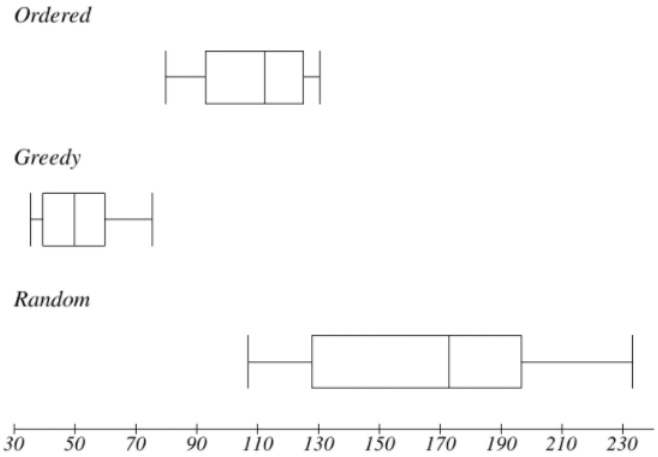


Figura 2.8: Distribución de las relaciones Beneficio/Tiempo de ejecución obtenidas con los diferentes operadores.

El primero de todos la variable que tenemos en cuenta es el tiempo total de rescatar a todos los grupos del estado solución final. El algoritmo de inicialización de ordered, obtenemos valores entre 2450 minutos y 3020 minutos, y la mediana de este, es cercana a los 2700 minutos. En segundo lugar tenemos el greedy con una mediana de 2650, ligeramente inferior a la mediana del ordered, y oscila entre valores de 2400 y 2870 minutos.

Finalmente el algoritmo que analizamos es el random, la mediana de este es bastante superior a la del resto, con un valor de 2980 y variando de los 2710 a 3340 minutos. Los valores de este no son muy aceptables.

El siguiente boxplot a analizar, será el que nos muestra el beneficio para cada algoritmo. El cálculo de este lo hemos realizado mediante la diferencia del estado inicial y el final del programa. De este modo podemos determinar el trabajo que realiza el algoritmo de Hill Climbing sobre el programa. El primero a analizar es el ordered, su beneficio toma valores desde 2350 a 3850 de beneficio, y obtenemos un valor de mediana de 3500. Este beneficio es bastante bueno y nos muestra que el algoritmo hace una gran parte de trabajo de la mejora de la solución inicial. El siguiente es el algoritmo de greedy, este tiene un beneficio realmente pequeño, ya que la solución inicial que genera es relativamente buena. Toma valores de 100 a 300 con una media de 160. Por último tenemos el algoritmo de random, con un beneficio pequeño, pero no tan escaso como el de greedy, con el que toma valores de entre 920 y 1750 con una mediana de 1150. Para esta parte de la experimentación, el algoritmo que sale ganador es el Ordered ya que es con el que obtenemos mayor beneficio.

Por último, analizamos el Beneficio respecto al tiempo de ejecución, nos mostrará cuál de todos es el que tiene mayor eficiencia. El primero de todos, ordered, tiene una eficiencia de 110 en mediana, y valores comprendidos entre 70 y 130. El algoritmo de greedy obtiene valores entre 30 y 70, con una mediana de 50. Finalmente tenemos el algoritmo de random con oscilaciones entre los 105 y los 230, y de mediana 170. El tercer algoritmo es el que tiene mayor beneficio, respecto al tiempo de ejecución.

El algoritmo de ordered nos permite obtener soluciones bastante buenas, similares a las del greedy, pero con un beneficio mucho mayor, es decir que el algoritmo de Hill Climbing es el que hace la gran parte de trabajo para obtener una solución aceptable. Por esta razón podemos concluir que el algoritmo de ordered es mejor que el de greedy. De este modo solo debemos comparar el algoritmo ordered con el random para demostrar la hipótesis planteada al principio del experimento. Aunque el ordered tenga un mayor beneficio, el random tiene mayor beneficio respecto al tiempo de ejecución, así que debemos tener en cuenta la solución obtenida por cada inicialización del estado. En este aspecto el ordered es bastante mejor que el random.

Conclusión:

Con este experimento concluimos que soluciones iniciales demasiado buenas, el

caso del greedy, no son efectivas para el Hill Climbing, ya que el estado solución apenas mejora. Y que el hecho de tener una solución media, el caso del random, tampoco es bueno, ya que encuentra máximos locales demasiado pronto.Podemos afirmar que el algoritmo de ordered es el mejor que podemos usar para este programa, ya que partimos de una solución inicial bastante mala, pero conseguimos un tiempo final bastante bueno, y con ello un gran beneficio.

2.3. Experimento 3

Una parte bastante importante para esta práctica es determinar los parámetros que vamos a usar en el algoritmo de Simulated Annealing para que sus resultados sean lo mejor posible, es decir, el menor tiempo de rescate de todos los grupos en un tiempo de ejecución razonable.

El algoritmo trabaja a partir de 4 parámetros sobre los cuales tenemos que estudiar su valor:

- **Número total de iteraciones** (steps) que realizará el algoritmo.
- **Número de iteraciones por cada cambio de temperatura:** Cuando haya un cambio de temperatura se harán un número constante de iteraciones en las cuales la probabilidad de elegir un sucesor peor se mantiene. Cada vez que se cumple este número de iteraciones decrece la probabilidad de escoger a un estado peor.
- **k:** Parámetro de la función de aceptación de estados que afecta a la probabilidad de elegir un estado sucesor peor. Sabemos que cuanto mayor sea el valor del parámetro, la probabilidad de quedarse con un estado sucesor peor tarda más en decrecer.
- **Lambda:** Otro parámetro de la función de aceptación de estados que afecta a la probabilidad de elegir un sucesor peor. Cuanto mayor sea el valor de lambda menos tardará en decrecer la probabilidad de aceptar un estado sucesor peor.

El objetivo es que el resultado del tiempo del Simulated Annealing sea igual o mejor que el del Hill Climbing pero en un principio nos resulta complicado saber cuáles serán los valores de los parámetros dichos anteriormente que nos beneficiaran más en la búsqueda. Por lo tanto no vamos a dar una hipótesis inicial con que valores de los parámetros creemos que funcionará mejor el algoritmo.

Vamos a proceder con el análisis. Este consistirá en descubrir cuales son el mejor conjunto de valores a asignar a los parámetros a partir de experimentaciones sobre ellos, teniendo fijado con anterioridad la cantidad de grupos a rescatar y los centros de los que vamos a disponer para el rescate. En un principio fijamos el número de iteraciones a un valor lo suficientemente grande como para que la probabilidad de aceptar un estado sucesor peor acaba decayendo a 0 antes de que termine la ejecución del algoritmo. Una vez definido haremos varias experimentaciones sobre los parámetros **k** y **lambda** manteniendo uno constante y modificando el valor del otro y viceversa. Al ser detectados los mejores valores para ambos parámetros iremos decreciendo el número de iteraciones hasta que sea posible, es decir, ejecutar en tan pocas iteraciones como se pueda obteniendo el mismo resultado que con la cantidad fijada al principio.

Usaremos la misma función heurística, operadores, sucesores, y estado inicial indicados en los experimentos anteriores.

Hay que tener en cuenta la aleatoriedad del algoritmo que estamos analizando, debido a ello consideramos que hay que realizar las búsquedas varias veces con los mismos parámetros y después hacer la media para conseguir un resultado más exacto.

Número de Centros de helicópteros	5
Número de Grupos	100
Número de repeticiones	15

Cuadro 2.1: Valores fijados a los elementos del problema.

Hecha toda esta introducción es momento de introducir el rango de valores, bastante amplio al principio, con el cual vamos a realizar los experimentos y así decidir aquellos que nos proporcionan la mejor solución. El número de iteraciones lo fijamos a 200.000 y los rangos son los siguientes:

- **k:** = [1, 200]
- **Lambda:** = [0.0001, 1]

Para este rango de valores conseguimos estos resultados:

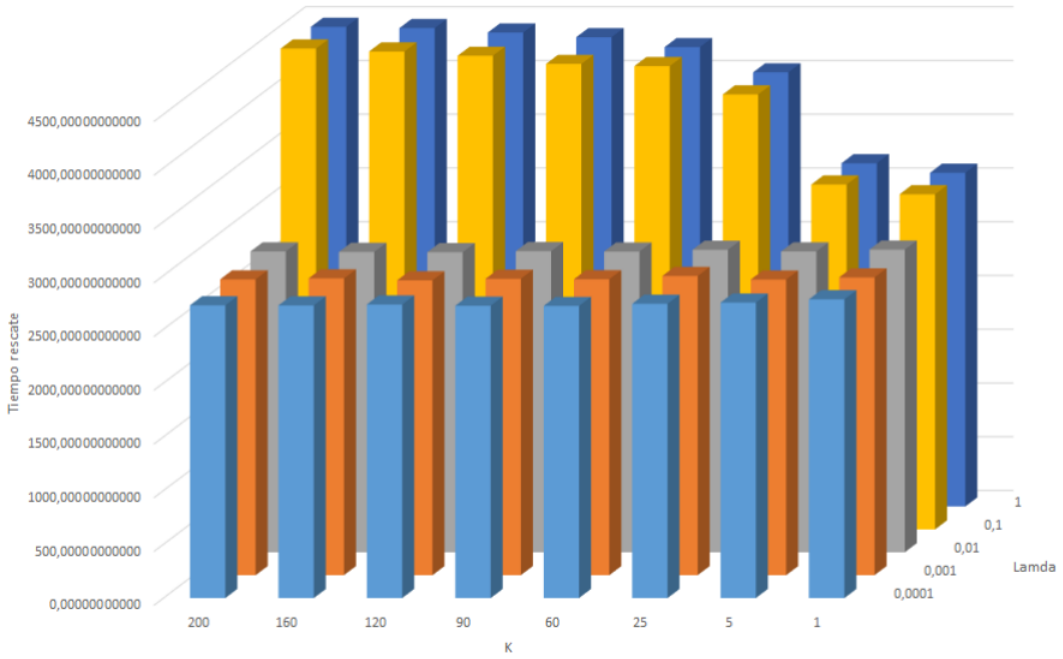


Figura 2.9: Tiempos de rescate obtenidos en función de los parámetros lambda y k.

De estos resultados llegamos a la conclusión que los mejores tiempos de rescate de todos los grupos se obtienen cuando estamos en un rango de valores de **k** = [20, 90] y **lambda** = [0.0001, 0.001].

Aun teniendo el rango en el cual surgen los valores mínimos, no podemos determinar con exactitud un valor, con lo cual repetiremos el experimento, pero esta vez trabajando en el rango más reducido que hemos obtenido:

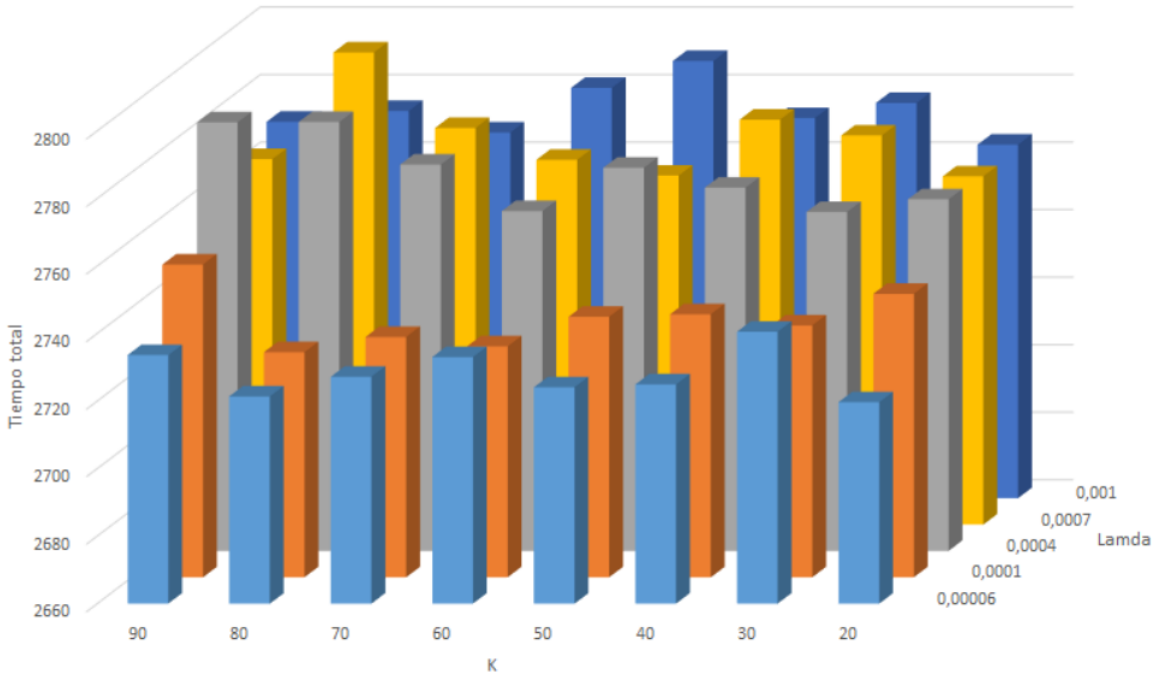


Figura 2.10: Tiempos de rescate obtenidos en función de los parámetros lambda y k con rangos inferiores a los visualizados anteriormente.

Gracias a estos resultados ya podemos ver claramente cuales son los valores de los parámetros más adecuados para obtener el tiempo mínimo: k = 80 y lambda = 0'00006. Con estos valores obtenemos unos resultados un poco mejores que con el Hill Climbing, sin embargo el tiempo que se tarda en ejecutar el algoritmo es un poco mayor en comparación aunque no lo hemos tenido en cuenta para decidir el valor de los parámetros.

Teniendo ya claros estos valores, es el momento de decidir el número de iteraciones para que funcione correctamente disminuyendo así el tiempo de ejecución respecto al obtenido. Para ello vamos a hacer una gráfica que vaya en función del número de iteraciones. Evaluaremos el tiempo de rescate en función a los pasos que se hayan ejecutado en el algoritmo buscando el punto, cantidad de iteraciones, a partir del cual el tiempo no sufre mejoras. Gracias a esto reduciremos el tiempo de ejecución del programa sin afectar al resultado de tiempo de rescates.

La siguiente gráfica va de 1 iteración a 200.000, la cantidad de iteraciones usadas para la experimentación con k y lambda:

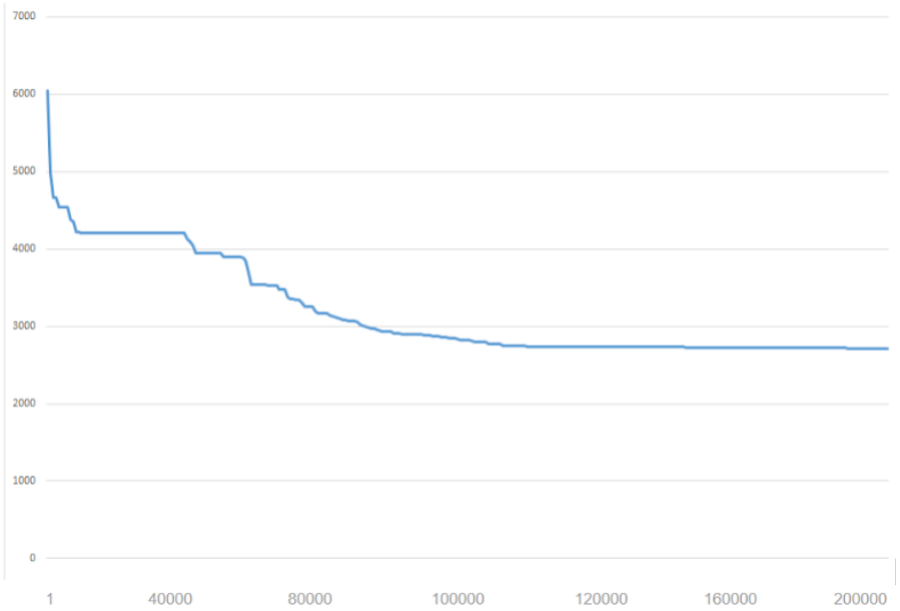


Figura 2.11: Relación entre el tiempo de rescate obtenido variando el número de iteraciones empleadas.

Podemos observar que al llegar a la mitad de iteraciones no se notan demasiado las mejoras, sino que estas que pueden apreciar en la primera mitad de iteraciones. Sin embargo, el tiempo de rescate sigue sufriendo mejoras hasta estabilizarse prácticamente por completo alrededor de la iteración 150.000. A partir de este punto decidimos que cortaremos la ejecución puesto que las siguientes iteraciones lo único que harán será aumentar el tiempo de ejecución sin llegar a obtener un resultado notablemente mejor.

Conclusión

Los parámetros que usaremos en el Simulated Annealing que nos ofrecen el mejor conjunto de soluciones son:

Iteraciones	150000
Iteraciones por cambio de temperatura	1000
k	80
lambda	0.00006

Cuadro 2.2: Valores que ofrecen mejor resultado para el Simulated Annealing

2.4. Experimento 4

Llegados a este punto ya tenemos definido el mejor estado inicial, sus operadores, el heurístico y los algoritmos Hill Climbing y Simulated Annealing con los parámetros más adecuados, ya podemos proceder a comparar algoritmos.

Los compararemos evaluando la tendencia del tiempo en ambos algoritmos usando diferentes seeds por cada grupo de variaciones de los elementos del estado inicial. Variamos los elementos del estado inicial con la siguiente proporción, 5:100, es decir, inicialmente haremos los análisis con 5 centros de helicópteros y 100 grupos a rescatar. Haremos 10 ejecuciones de cada algoritmo con seeds randoms manteniendo la misma proporción para luego hacer la media y así conseguir un resultado más real y fiable. Una vez terminado el análisis aumentaremos la cantidad de centros de helicópteros en 5, y así sucesivamente hasta que hallemos una tendencia en el tiempo de ejecución.

Como hipótesis inicial sobre el resultado de estos análisis, es de esperar que el tiempo de ejecución del Simulated Annealing vaya aumentando linealmente puesto que sólo se genera un estado sucesor por iteración y el número de iteraciones está fijado de antemano. Por lo tanto el único factor que influirá en el tiempo de ejecución es lo que se invierte en generar el estado inicial y los operadores. Por otra parte, el Hill climbing no tiene un número fijo de generación de estados sucesores, así que suponemos que el tiempo de ejecución aumentará exponencialmente puesto que cuantos más centros tengamos, si hay que seguir rescatando la misma cantidad de grupos, más aumentará el factor de ramificación.

Tras ejecutar los algoritmos obtenemos los siguientes resultados en el Simulated Annealing:

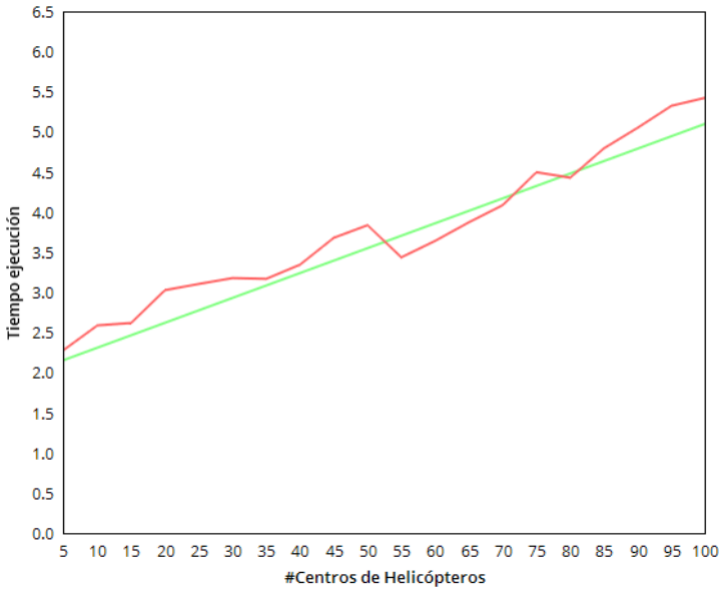


Figura 2.12: Tiempos de ejecución obtenidos para diferentes números de centros para Simulated Annealing.

La línea de color rojo representa el tiempo de ejecución(s) respecto la cantidad de centros. La línea de color verde representa la forma en la que aumenta el tiempo de ejecución(s). Tal y como hemos previsto en la hipótesis inicial, el tiempo de ejecución aumenta linealmente en función de la cantidad de centros de helicópteros que tengamos, además, se ve como los puntos de la línea de color rojo se asemejan a la recta de regresión lineal de color verde. Pero aunque el tiempo de ejecución tenga ese tipo de crecimiento, eso no quiere decir que el tiempo de rescate tenga el mismo comportamiento. Este lo estudiaremos posteriormente cuando compararemos los diferentes tiempos de rescate entre los algoritmos.

Tras ejecutar los algoritmos obtenemos los siguientes resultados en el Hill Climbing:

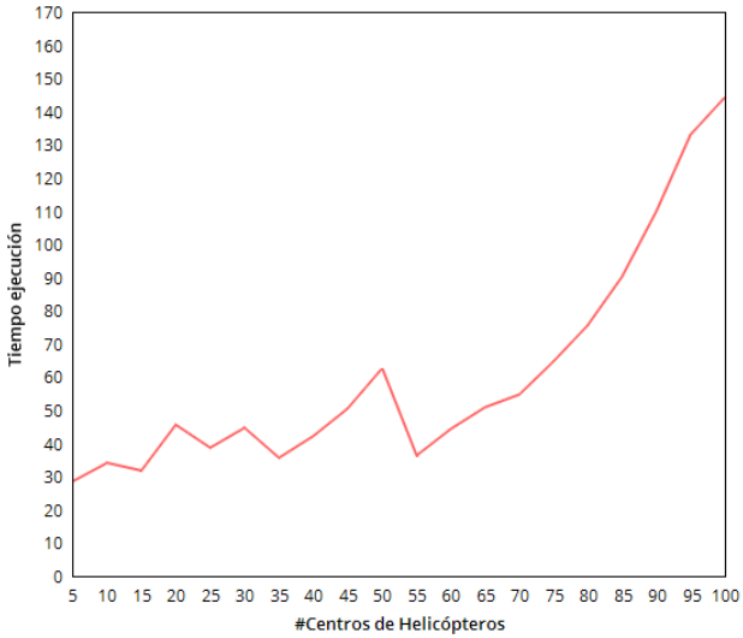


Figura 2.13: Tiempos de ejecución obtenidos para diferentes números de centros para Hill Climbing.

Al observar la gráfica por encima podríamos llegar a pensar que el tiempo de ejecución el el Hill climbing no sigue ningún patrón y por lo tanto nuestra hipótesis inicial quedaría descartada, pero hay que pararse un poco y analizarla teniendo en cuenta cómo se genera el estado inicial:

- Por cada helicóptero se generan (G/H)+1 rescates de tres posiciones vacías y luego se van rellenando con un grupo en cada rescate.

Eso implica que si se mantiene constante la cantidad de grupos y se van aumentando los centros de helicópteros, la cantidad de rescates por helicóptero va a ir variando. Entonces, desde la primera configuración en la que se tiene una cantidad de rescates por helicóptero hasta la última que se tiene la misma cantidad (siempre con diferentes centros), el tiempo aumenta exponencialmente. Esto queda bastante claro si observamos la gráfica a partir de 55 centros. Desde este punto hasta el final de la gráfica se tiene un rescate por helicóptero y entonces vemos claramente como aumenta el tiempo de ejecución de forma exponencial.

Finalmente queremos acabar comprobando si los parámetros hallados en el experimento 3 para el algoritmo Hill Climbing siguen dando buenos resultados. Esto lo haremos comparando los resultados obtenidos, en cuanto a tiempo de rescate, de los dos algoritmos con los mismos datos para generar la solución inicial que los usados anteriormente y con seeds randoms. El resultado es el siguiente:

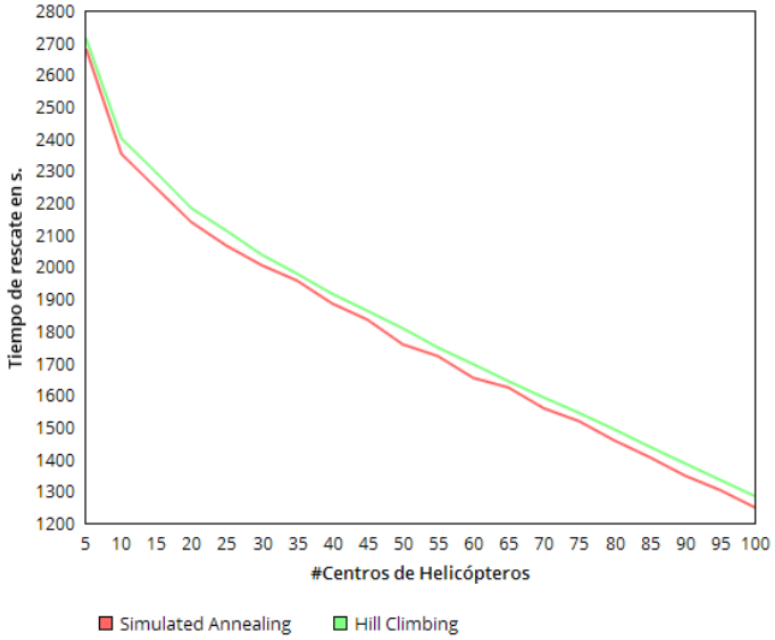


Figura 2.14: Tiempos de rescate obtenidos para diferentes números de centros para Hill Climbing y Simulated Annealing.

La línea roja que representa la ejecución con el Simulated Annealing está ligeramente por debajo que la que representa la ejecución con el Hill Climbing, por lo tanto deducimos que el primero da realmente buenos resultados, incluso mejores, además que tiene la ventaja de que el tiempo de ejecución es bastante poco en comparación con el segundo algoritmo.

Como observación, vemos que si se van aumentando la cantidad de centros manteniendo constante la cantidad de grupos, al principio el tiempo de rescate baja enormemente pero a medida que se va aumentando, de 5 en 5, la cantidad de centros, el tiempo va disminuyendo de forma constante y regular.

Conclusión

A pesar de que la diferencia de tiempos de rescate entre ambos algoritmos no sea lo bastante grande, y aun siendo random el algoritmo de Simulated Annealing, este nos proporciona más ventajas si lo usamos puesto que el tiempo de ejecución es realmente poco y además las soluciones que nos proporciona para nuestro problema son suficientemente buenas.

2.5. Experimento 5

El objetivo principal de éste experimento va a ser estudiar cómo influye el tamaño del problema en el coste temporal de la búsqueda. Para realizar éste análisis asumiremos que el coste depende del factor de ramificación y del tamaño del espacio de búsqueda.

Dado que el factor de ramificación es algo complejo (como hemos podido ver en el Experimento 1, en el que concluimos que utilizaríamos el operador swap), lo que haremos será dividir el experimento en dos con el objetivo de entender mejor como afecta el tamaño del problema al coste temporal de búsqueda. En ambos experimentos lo que haremos será escoger distintos tamaños de búsqueda y obtener así sus respectivos tiempos de ejecución.

En primer lugar, aislaremos el crecimiento del tamaño del problema incrementando solamente el número de grupos a rescatar. De esta manera esperamos ver una tendencia al menos cuadrática, dado que el factor de ramificación es cuadrático en el número de grupos.

La metodología empleada en éste experimento será la siguiente:

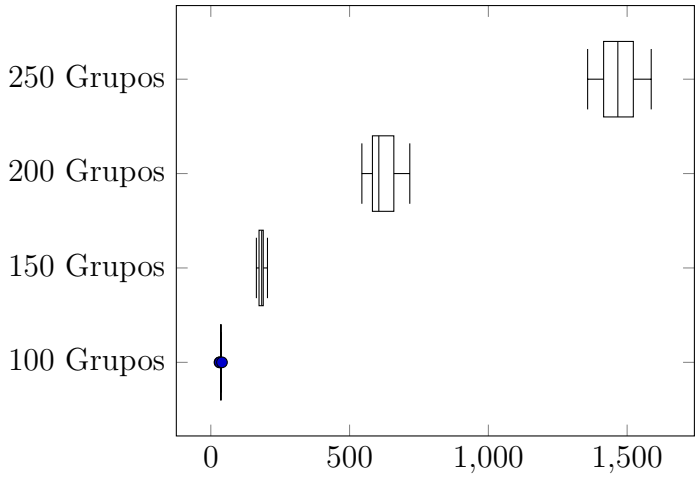
- Escogeremos de manera aleatoria 10 semillas distintas para las ejecuciones sobre un mismo tamaño de problema (dentro de una misma ejecución, se usará la misma semilla tanto para el generador de grupos como el de centros).
- Realizaremos 10 ejecuciones para cada tamaño para obtener así la media de los resultados.
- Aumentaremos el tamaño incrementando para cada experimentación en 50 el número de grupos, partiendo de 100 grupos y llegando a los 250.
- Usaremos el algoritmo Hill Climbing con la configuración establecida en los experimentos 1 y 2.
- La medición del tiempo de computación se realizará con la diferencia de los tiempos obtenidos pre y post ejecución con la llamada `System.currentTimeMillis()` de Java.
- Las ejecuciones se realizarán sobre un ordenador con procesador Intel® Core™ i5-3570K CPU @ 3.40Hgz.

En la siguiente tabla podemos ver un sumario de los resultados obtenidos, con ella, nos podemos hacer una idea del rango de valores y del crecimiento del coste temporal de búsqueda a medida que aumenta el número de grupos.

A continuación, con el objetivo de visualizar mejor los datos obtenidos, tenemos el boxplot con la distribución de todos los tiempos de ejecución.

Tamaño	100	150	200	250
Media	35,982	182,494	617,451	1469,481
Desviación	2,531	11,867	57,233	81,472

Cuadro 2.3: Descripción de los tiempos de ejecución (en segundos) obtenidos variando el número de grupos.



Es cierto que claramente podemos observar como a medida que el número de grupos aumenta, también lo hace el tiempo de ejecución. Sin embargo, pese a que a primera vista podemos ver que el aumento no es proporcional, queremos saber que tendencia sigue. Para ello hemos elaborado la siguiente gráfica:

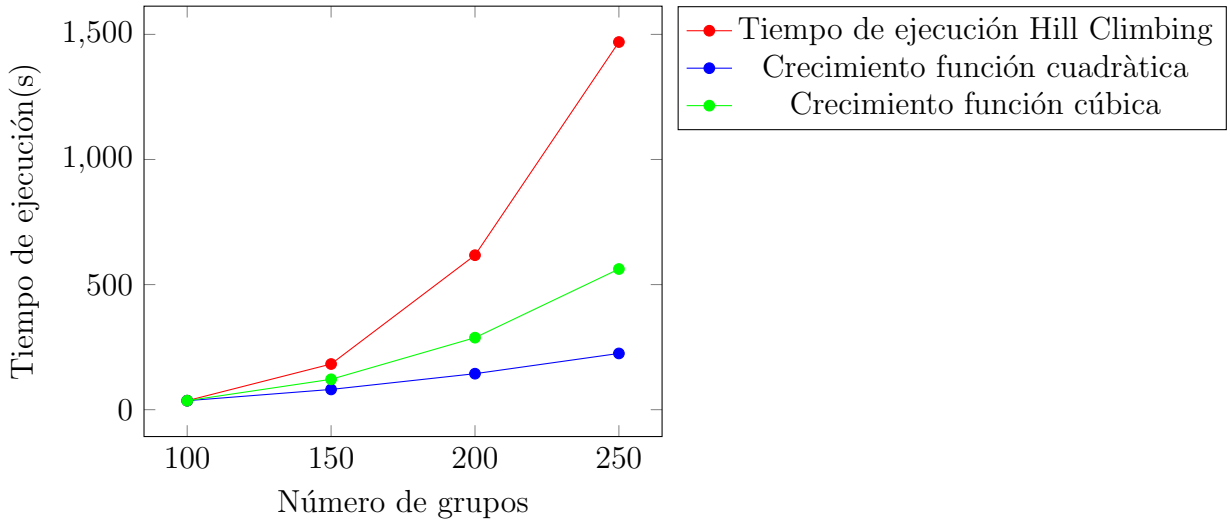


Figura 2.15: Comparativa crecimiento funciones.

Como podemos observar y pese a que necesitaríamos mas muestras para con-

firmar que la tendencia que cumple, el crecimiento de la función es bastante más elevado que cuadrático o cúbico, en contra de lo que pensábamos inicialmente. Ésto se puede deber principalmente a la influencia del tamaño del espacio de búsqueda sobre el factor de ramificación incrementando notablemente el coste temporal de la búsqueda.

Una vez analizado el coste temporal en función al número de grupos, lo vamos a hacer en función al número de centros. En este caso la hipótesis es distinta, ya que como el factor de ramificación crece linealmente en el número de centros (véase Experimento 1), esperamos ver un crecimiento lineal en el tiempo de ejecución también.

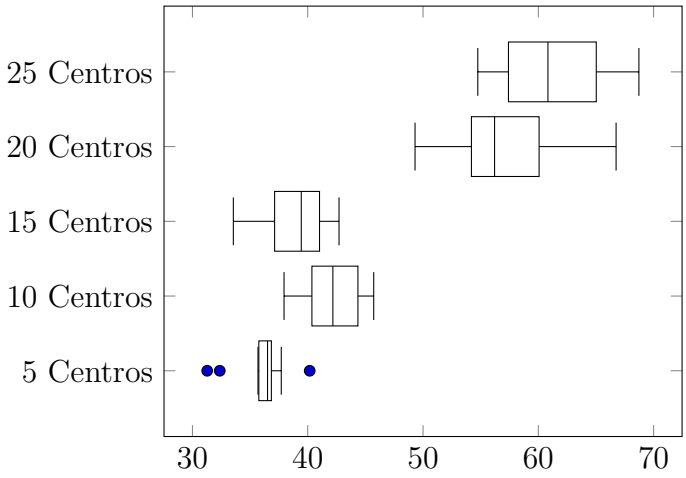
La metodología aplicada será idéntica, la única diferencia será que estudiaremos un intervalo mas de número de centros ya que en éste caso el tiempo de ejecución no nos será limitante y así podremos observar la tendencia mejor.

En primer lugar desplegamos a continuación una tabla análoga a la vista anteriormente para hacernos una idea del rango de valores que tendremos en el experimento.

Tamaño	5	10	15	20	25
Media	35,982	42,147	38,911	57,186	61,534
Desviación	2,531	2,647	2,990	5,034	4,844

Cuadro 2.4: Descripción de los tiempos de ejecución (en segundos) obtenidos variando el número de centros.

A continuación, para visualizar la distribución de los datos obtenidos, tenemos el siguiente boxplot:



Simplemente observando el boxplot, podemos ver que pese a que hay una variación en la mayoría de los valores para número de centros igual a 15 (que se puede

deber a factores de la implementación), en general vemos que el tiempo de ejecución aumenta pero de manera controlada, sin ir aumentando la más cada vez en relación al anterior, por lo que podemos asumir que se cumple la hipótesis y que el coste temporal aumenta de manera lineal cuando aumentamos el número de helicópteros.

2.6. Experimento 6

Suponemos que el tiempo aumentara de manera lineal ya que el numero de helicopteros aumenta en n el factor de ramificacion. En cuanto a la calidad de la solución dado que el heuristico es el tiempo en si, no aumentara mucho ya que cada helicoptero busca a los mas cercanos y se los queda el (pese a que este helicoptero haga mucho mas trabajo) porque no buscamos mejorar el tiempo en paralelo sino en secuencial y añadir helicopteros en los mismos sitios en los que ya hay no mejorará la solucion porque siguen estando igual de cerca a los otros grupos. Sin embargo si añadiesemos estos helicopteros en centros con nuevas localizaciones, el tiempo aumentaria de la misma manera (linealmente) sin embargo la calidad de la solución aumentaria mas que dejandolos en el mismo punto que ya hay.

Comprobaremos que el tiempo aumenta de manera lineal y que la calidad de la solución aumenta de manera muy lenta.

En éste experimento nos vamos a centrar en analizar como afectaría al programa el aumento del número de helicópteros (que había sido 1 hasta ahora), estudiando la afectación de la calidad de la solución y el tiempo de ejecución entre otros factores.

Inicialmente, vamos a centrarnos en el coste temporal de la búsqueda cuando aumenta el número de helicópteros. Si volvemos al Experimento 1, veremos que el número de helicópteros afecta linealment a factor de ramificación, con lo que nuestra **hipótesis** será que el tiempo de ejecución aumentará linealmente también.

La metodología empleada para éste experimento es la siguiente:

- Escogeremos de manera aleatoria 10 semillas distintas para las ejecuciones sobre un mismo número de helicópteros. Con estas 10 ejecuciones obtendremos la media de los resultados.
- Los otros elementos del problema quedarán prefijados siendo el número de centros 5 y el número de grupos 100.
- Incrementamos el número de helicópteros en intervalos de 1, iniciando en 1 y acabando en 4.
- Usaremos el algoritmo Hill Climbing con la configuración establecida en los experimentos 1 y 2.
- Para la medición del tiempo de ejecución emplearemos los mismos métodos y recursos usados en el Experimento 5.

Una vez obtenidos los datos, empleamos el boxplot para observar la distribución que siguen:

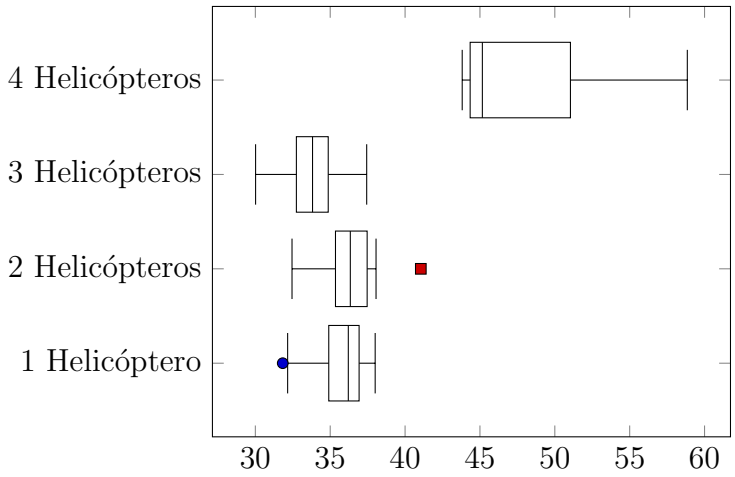


Figura 2.16: Distribución de los tiempos de ejecución (en segundos) obtenidos para los diferentes números de helicópteros.

Pese a que el número de muestras no es suficiente para observar una tendencia, observamos que si que se produce un aumento en los tiempos de ejecución, destacando la anomalía cuando el número de helicópteros es 3, que puede ser consecuencia de factores de implementación.

A continuación, vamos a mostrar una gráfica con las medias de los tiempos de rescate obtenidos con las ejecuciones anteriores, con el objetivo de ver si es cierto que se produce una mejora muy lenta o si observamos algún comportamiento distinto. Creemos que pese a que se produzca una mejora, esta será muy lenta ya que dado que el heurístico empleado calcula el tiempo total de todos los helicópteros (sin ser en paralelo) esto implicará que cada helicóptero buscará a sus grupos mas cercanos, sin importar que ya tenga mucho trabajo por hacer o no ya que como hemos dicho el heurístico calcula el tiempo suponiendo que todo sucede en secuencial.

A continuación observamos la gráfica mencionada anteriormente:

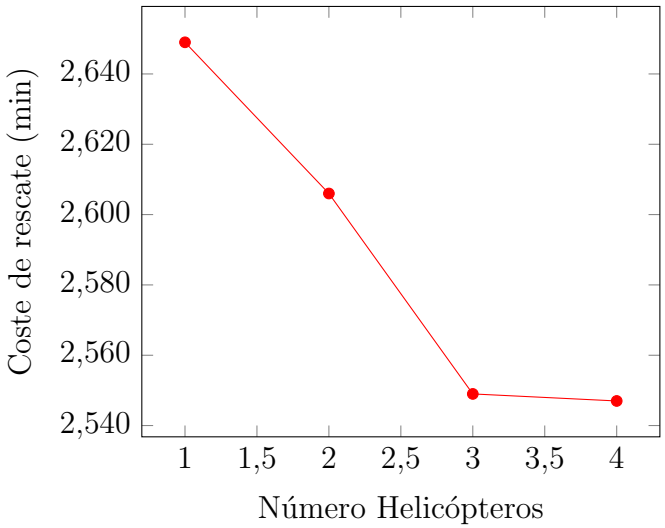


Figura 2.17: Coste de rescate en función al número de helicópteros,

Podemos observar que existe una tendencia a la baja en los tiempos de rescate que parece estancarse al final. No obstante vemos que en general se producen mejoras 40 min en cada paso pese a haber duplicado el número de helicópteros, lo que nos confirma que las mejoras se producen de manera muy lenta.

Esto no sucedería si en lugar de duplicar el número de helicópteros añadiendo uno mas a cada centro, duplicásemos el número de centros pero manteniendo el de helicópteros. En este caso las nuevas localidades de los centros harían que nuevos helicópteros estuviesen mas cerca de grupos a que anteriormente se rescataban con helicópteros de centros mas lejanos, mejorando así en mayor medida la calidad de la solución.

2.7. Experimento 7

En este experimento debemos analizar el comportamiento de los algoritmos Hill Climbing i Simulated Annealing, con el segundo heurístico, que nos prioriza que los grupos de prioridad 1 sean rescatados antes que el resto. Hemos definido una variable que se llama penalización, y durante este experimento cambiamos el valor de x en Heurístico = penaliza*x +tiempo. Esta variable nos define el peso que tiene la penalización. Para tener unos datos más concretos, hemos realizado 10 ejecuciones, con distintas seeds, para cada variable con Hill Climbing, 30 en el caso del Simulated Annealing, puesto que al ser random, es mejor tener más datos para aumentar la precisión de estos. Al final de este experimento dejaremos como datos adicionales las tablas donde hemos recogido estos datos. Para cada variable, hemos hecho la mediana de cada uno de los datos obtenidos, y así con ellas podemos analizar la evolución de cada algoritmo respecto al aumento de la penalización, este con la notación de penalización*x.

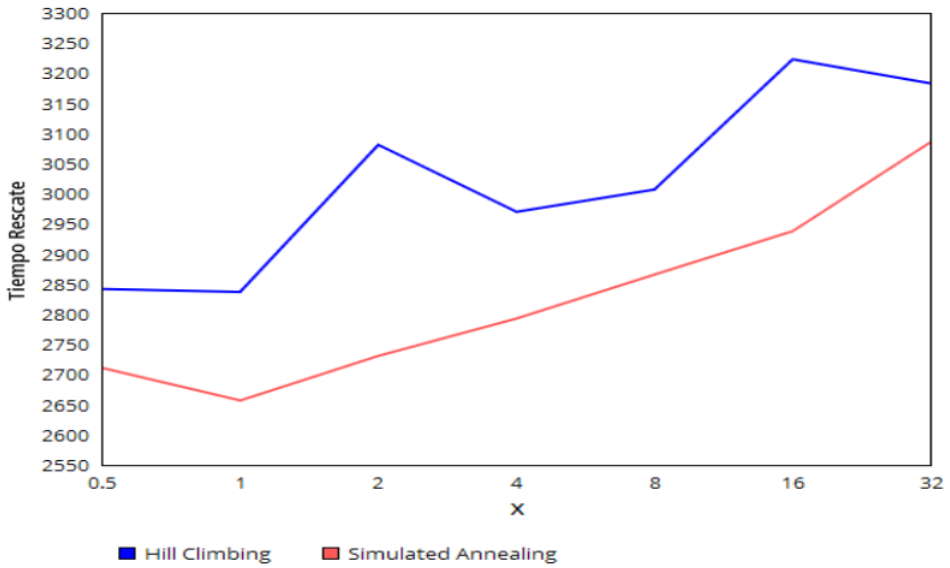


Figura 2.18: Tiempos de rescate obtenidos para diferentes penalizaciones para Hill Climbing y Simulated Annealing.

Analizamos Hill Climbing:

Esta función al principio, vemos que se mantienen en un valor bajo y constante. Con el valor de penalización tan bajo, el heurístico no es capaz de rescatar primero a los grupos de prioridad numero 1, ya que pràcticamente tienen el mismo heurístico que un estado que no los rescata al principio, por ese motivo el tiempo de rescate es bajo, ya que el heurístico da más importancia al tiempo que a la penalización en sí. Conforme aumentamos el tamaño de la variable x, el valor del tiempo va subiendo, ya que empezamos a dar más importancia a que los grupos de prioridad 1 estén delante, y de este modo conseguimos la solución deseada. Esta gráfica se mantiene

irregular, pero mantiene una tendencia de subir a medida que “x” lo va haciendo.

Analizamos Simulated Annealing:

En esta gráfica, al principio, para x = 0.5 y x =1, nos encontramos en la misma situación que para el Hill Climbing, incluso más exagerada, ya que SA nos disminuye el tiempo de rescate para x =1, a pesar de ello, esto no nos ordena los grupos por prioridad, por el mismo motivo explicado anteriormente. Para el resto de variables vemos que esta vez el Simulated Annealing aumenta de manera prácticamente constante, hasta que toma el valor de x = 16, y aunque en la gráfica no se aprecie, a partir de x =16, el valor del tiempo de rescate se mantiene constante, pero el tiempo se ve afectado por el random de las seeds y el del propio algoritmo.

Conclusión:

Si escogemos una x demasiado pequeña el algoritmo no será capaz de ordenar los grupos por prioridad, puesto que el incremento de la penalización respecto al heurístico no es relevante. Entonces debemos escoger una “x” tal que el valor del heurístico nos permita que realmente penalice. Como vemos en la gráfica, a partir del valor x ¿= 16 el algoritmo ya ordena los grupos por prioridad, y es por eso que a partir de este punto, el valor del tiempo de rescate se mantiene más o menos estable, o incluso el tiempo disminuye. Esto se debe a que una vez superado el punto donde ordenas todos los grupos la suma de la penalización es constante para todos, excepto en los casos donde quitas un grupo de prioridad 1 del principio, y entonces el heurístico es mucho peor y descartamos el estado, por tanto, aunque se incremente el valor de su ponderación, este no afectará al resultado del tiempo.

Capítulo 3

Conclusión

A lo largo de este proyecto de introducción a la Inteligencia Artificial (IA) hemos logrado comprender las diferencias entre afrontar un problema con algoritmos y técnicas convencionales versus hacerlo con algoritmos del campo de la IA. La experimentación nos ha permitido entender con más claridad lo explicado en las clases de teoría y transformar aquellos conocimientos teóricos en prácticos.

Hemos podido ampliar y perfeccionar nuestros conocimientos sobre dos de los más importantes algoritmos de la búsqueda local, Hill Climbing y Simulated Annealing. Gracias a los específicos experimetnos hemos sido capazes de entender a la perfección todos y cada uno de los distintos aspectos característicos del funcionamiento de ambos algoritmos y sus limitaciones basándonos en un ejemplo de la vida real.

Por todos estos motivos creemos que la realización de esta práctica nos ha sido de gran utilidad, especialmente para ampliar y enriquecer nuestros conocimientos.

Capítulo 4

Datos adicionales

Dado que en el experimento 7 no se muestra ninguna representación directa de los datos obtenidos (e.g boxplot), a continuación de expone la tabla de valores para Hill Climbing y Simulated Annealing obtenidas y a partir de las cuales se han calculado las distintas medianas empleadas en dicho experimento.

Tabla de datos Hill Climbing:

	Penal.*0.5	Penal.*1	Penal.*2	Penal.*4	Penal.*8	Penal.*16	Penal.*32
1	2840.117	2486.369	3456.274	3153.448	3145.768	2937.936	3217.448
2	2597.560	2713.740	3024.168	2740.735	2838.144	3363.393	3047.213
3	2773.881	2898.157	3431.459	3392.697	3235.113	3465.740	3308.722
4	2726.832	2806.552	3110.971	2688.379	3240.259	3071.967	3384.607
5	2847.132	2680.171	3108.000	2817.061	2984.018	2803.152	3133.124
6	2973.879	2878.501	2932.274	2854.950	2917.111	3296.336	2992.561
7	2851.967	2567.925	3405.962	3166.115	2909.768	3745.644	3474.574
8	2968.995	3568.212	2674.890	3093.253	3051.171	3183.408	3222.577
9	2994.576	2780.108	2863.035	2730.942	2796.768	3052.329	2881.991
10	2845.618	2991.679	2805.175	3068.801	2960.652	3319.240	3175.821

Tabla de datos Simulated Annealing:

	Penal.*0.5	Penal.*1	Penal.*2	Penal.*4	Penal.*8	Penal.*16	Penal.*32
1	2635.226	2788.810	2541.621	2921.212	2966.937	2831.922	2951.384
2	2784.921	3025.580	2771.120	2809.446	2712.808	2986.369	3135.133
3	2702.504	2630.010	2544.754	2614.134	2592.030	2958.808	2852.661
4	2818.089	2440.513	2535.259	2510.732	2984.679	2788.425	3050.746
5	2600.620	2439.204	2519.576	3034.737	2712.739	2963.455	3295.500
6	2811.902	2717.801	2652.552	2534.280	3266.794	2938.829	3290.014
7	2364.825	3300.180	3004.718	2729.772	2832.850	2738.560	3295.496
8	2972.079	2536.505	2980.631	3066.341	2744.711	2737.077	2934.498
9	2698.144	2609.487	2809.872	2563.649	3325.594	2933.174	3237.440
10	2324.838	2825.093	2695.157	2819.238	2888.747	2939.067	3023.708
11	2906.199	2750.425	2625.074	2574.776	2797.052	2839.480	3013.646
12	2829.289	2486.886	2853.271	2700.675	2854.954	2868.997	3037.685
13	2873.976	2207.931	2481.922	2838.358	2984.160	3241.095	3191.400
14	2391.838	3365.954	3277.193	2923.637	2675.896	3147.087	3187.059
15	2707.943	2471.238	2676.764	2971.136	2458.288	2661.370	2887.680
16	2460.169	2481.207	2965.096	3019.398	2813.890	2567.905	3076.643
17	2785.840	2545.414	2881.358	2829.962	3049.672	2795.804	3009.717
18	2546.074	2847.725	2506.024	2970.200	2959.876	2995.101	3377.937
19	2881.592	2950.395	2519.928	2435.933	3208.064	2855.185	2961.196
20	2549.776	2769.898	2518.175	2819.671	2896.578	3417.399	3068.227
21	2436.886	2649.348	2953.182	2958.814	2701.104	3149.384	3134.059
22	2811.442	2544.657	2725.836	2731.776	2696.181	3250.767	3335.545
23	2936.405	2979.808	2755.649	2906.349	3296.687	3073.477	3149.589
24	2879.401	2371.425	2687.653	2561.940	2768.751	2928.607	2982.349
25	2327.277	2331.179	2829.297	2690.419	2763.578	2630.242	3154.150
26	3067.354	2422.387	2593.045	2800.551	2945.148	3233.902	3103.165
27	2648.267	2560.557	3373.551	2622.139	2835.702	3027.257	3263.218
28	2628.464	2694.868	2617.080	2714.126	2729.398	3022.843	2941.957
29	2909.463	2166.003	2315.240	3215.000	2857.825	2885.847	2950.228
30	3049.903	2815.500	2745.480	2915.528	2668.046	2757.542	2710.894