UNIVERSITAT POLITÈCNICA DE CATALUNYA

FACULTAT D'INFORMÀTICA DE BARCELONA

Intel·ligència Artificial
Práctica de búsqueda local - IA Desastres

Dídac Jaquet
Roger Sánchez
Xavier Alaman
Abril del 2022
Cuatrimestre de Primavera 2021 - 2022

Índice

1. Descripción del problema

En esta primera práctica de búsqueda local, se plantea una simplificación sobre la planificación de operaciones de rescate en una catástrofe natural. Estas operaciones de rescate consisten básicamente en la recogida de grupos de personas con distintas necesidades sobre una extensión de terreno predeterminada.

Este problema intenta buscar una secuencia óptima, o por lo menos eficiente, para recoger grupos de personas usando distintos helicópteros desde múltiples centros de rescate a lo largo de un conjunto de salidas.

En un caso real intervienen decenas de agentes, restricciones y condiciones climáticas adversas con comportamientos casi aleatorios que elevan la complejidad de encontrar una solución óptima muchísimo. Por tanto en este caso se reduce el conjunto de elementos que intervienen en el problema a los siguientes:

1.1 Elementos del problema

A continuación se listan todos los elementos que conforman el problema junto con una breve descripción.

1.1.1 Área de rescate

El área de rescate es el espacio físico sobre el que se define el espacio donde sucede la acción del problema. En este caso un terreno de una extensión de 50km x 50km. La esquina inferior es el centro de coordenadas.

1.1.2 Grupos

Un conjunto de personas se encuentran en una coordenada concreta del área de rescate. La agrupación de estas personas conjuntamente con las coordenadas definen un Grupo. Se entiende que en la vida real en número de personas podría ser relativamente grande por grupo, pero para este caso y para que el problema pueda ser resoluble se entiende que no habrà más de 15 personas por grupo para que puedan ser cargadas en un helicóptero.

Los grupos pueden contener personas heridas o no, definiendo así una prioridad en el orden de rescate. Los grupos con personas heridas serán considerados de prioridad 1 y aquellos que no las contienen de prioridad 2. Al igual que en una situación real, subir personas heridas a un helicóptero en caso de rescate, cuesta más si las persona están heridas. Por tanto las personas de un grupo de prioridad 1 tardaran en subir 2 minutos y las de un grupo de prioridad 2 1 minuto.

1.1.3 Helicóptero

Puede parecer obvio dada la información proporcionada hasta el momento del problema, pero los helicópteros son los encargados de recoger los diferentes grupos sobre el área de rescate.

Cada helicóptero tiene capacidad para recoger un máximo de 15 personas. Los helicópteros no pueden recoger grupos parcialmente ni más de tres grupos por salida.

Cada helicóptero al llegar al centro de rescate debe permanecer 10 minutos quieto antes de volver a realizar otro rescate, en caso de que así lo desee. Todo helicóptero debe volver a la base de la que ha partido sin excepción. Los helicópteros pueden realizar tantos viajes como quieran, hasta alcanzar el número de grupos entre todos ellos. Es decir, un helicóptero en caso extremo, podría realizar todos los rescates, con un grupo por rescate, dejando a los otros helicópteros sin partir (se entiende como un caso probablemente ineficiente).

Estos se mueven a una velocidad de 100 km/h por el área de rescate. Esta velocidad se entiende igual en todo momento, desde que despega hasta que aterriza sin reducir ni acelerar su marcha. Se entiende que los helicópteros se mueven en línea recta por el área de rescate.

No se contabiliza el tiempo de bajar a los tripulantes del helicóptero. En este caso sería lo mismo que aumentar el tiempo de cargarlos en el helicóptero por dos. Por tanto, aparte de aumentar el tiempo total de rescate por helicóptero no influiría en nada más.

1.1.4 Centros de rescate

Igual que las personas los helicópteros se pueden agrupar en centros de rescate. O lo que es lo mismo, cada grupo de rescate puede contener distintos helicópteros.

Los grupos de rescate se encuentran posicionados dentro del área de rescate en los bordes de la misma. Entendiendo como unidad mínima de geolocalización dentro del área de rescate el km, no tiene sentido pensar en que pueda haber más de 200 centros de rescate.

Cada centro de rescate puede contener un número ilimitado de helicópteros.

1.2 Estado del problema

Un estado del problema se entiende como una fotografía exacta de los diferentes elementos del problema en un momento concreto del espacio y tiempo.

Concretamente en nuestro caso entendemos, desde el ámbito más general, un estado del problema como una solución final.

1.2.1 Elementos del estado del problema

Un estado del problema entendido como solución final y salvaguardando las distancias con la implementación que se explica más adelante, está compuesto por la relación de salidas por helicóptero juntamente con los grupos que se recogen por salida.

Para poder hacer esta asociación de elementos también necesitaremos un conjunto de grupos distribuidos a lo largo y ancho del terreno, además de un conjunto de centros de rescate distribuidos alrededor de los bordes de la área de rescate y un conjunto de helicópteros inherente en cada centro de rescate.

1.2.2 Restricciones de un estado solución del problema

Un estado solución no es cualquier agrupación de helicópteros, salidas y grupos. Una solución solo es válida si y sólo si cumple ciertos requisitos introducidos anteriormente:

- Un grupo solo puede ser recogido exactamente por un helicóptero (implícito).
- Consecuencia de la anterior una agrupación de grupos en una salida de rescate sólo puede ser realizada por un helicóptero (implícito).
- Como consecuencia de las dos anteriores todos los helicópteros deben realizar salidas de rescate distintas (implícito).
- Todos los grupos deben ser rescatados.
- Toda salida de rescate debe contener un máximo de 3 grupos.
- Toda salida de rescate debe contener un máximo de 15 personas entre los diferentes grupos que la componen.
- Toda salida de rescate debe empezar y acabar en el mismo grupo de rescate.

1.3 Justificación problema de búsqueda local

Analizado el problema se puede llegar a la conclusión de que es un problema de búsqueda local.

Posteriormente se explica en más profundidad los distintos criterios que hacen mejor o peor una solución, pero bàsicamente se usan dos:

- La suma de todos los tiempos empleados por todos los helicópteros para rescatar a todos los grupos.
- La suma de todos los tiempos empleados por todos los helicópteros para rescatar todos los grupos buscando minimizar el tiempo que se tarda hasta rescatar el todos los grupos de prioridad 1.

Estos criterios llevan a pensar que son una buena estimación para encontrar una solución con el menor de estos tiempos (heurístico). Por tanto, si se comprende todo el espacio de posibles soluciones del problema como un espacio de búsqueda, esta estimación puede ser la guía para encontrar la solución.

Definido el problema como la búsqueda de una solución “óptima” en un espació de búsqueda determinado y teniendo en cuenta que no se quiere encontrar el mejor camino para generar una solución sino la mejor solución. Se puede entender el problema como un problema de búsqueda local que quiere encontrar la solución con el menor de los tiempos (según el criterio escogido) asociado.

2. Implementación

En este apartado se procede a definir en profundidad varios componentes que integran la resolución del problema así como el proyecto. Haciendo cierto énfasis, donde proceda, en la implementación.

2.1 Implementación del estado del problema

Para la implementación de un estado del problema se ha procedido a usar diferentes estructuras de datos para almacenar principalmente la relación entre helicópteros, salidas y grupos. También se han usado distintos valores auxiliares para hacer más fácil el cómputo de las operaciones, pero estos se mantienen igual para todas las operaciones, por lo tanto en java han sido declarados como static.
Un estado contiene los siguientes elementos:

- **static tempGrupos:** Es una matriz de tipo int que contiene los tiempos para moverse entre cada grupo. También incluye el tiempo de recogida de ese grupo concreto. Es decir para moverse de un punto A a un punto B en tempGrupos[A][B] se incluye el tiempo para desplazarse de A a B y recojer las personas del grupo B. Análogamente en tempGrupos[B][A] se incluye el mismo tiempo de desplazamiento más el tiempo que se tarda en recoger la gente de A. Es de tamaño centros x centros. Tiempo representado en ms.
- **static tempCentrosGrupos:** Es una matriz de tipo int que contiene los tiempos para moverse de un centro a un grupo. Concretamente el tiempo para desplazarse de un centro a un grupo más el tiempo de recogida de ese grupo. Es de tamaño centros x grupos. Tiempo representado en ms.
- **static tempGruposCentros:** Es una matriz de tipo int que contiene los tiempos para moverse entre un grupo y un centro. Contiene el tiempo de desplazamiento más los 10 minutos que va a permanecer parado el helicóptero en el centro antes de despegar. Posteriormente en el cálculo del heurístico se descontará este tiempo extra para el último grupo -> centro de cada trayecto. Es de tamaño grupos x centros. Tiempo representado en ms.
- **static gruposPrioridad:** Es un array de ints de tamaño grupos que contiene información sobre el tipo de prioridad de cada grupo, 1 o 2.
- **static nGrupos:** Es un int que recoge el número de grupos del problema.
- **static nHelicopteros:** Es un int que recoge el número de helicópteros del problema.
- **static getnHelicopterosCentro:** Es un int que recoge el número de helicópteros por centro del problema.
- **static personasGrupos:** Es un array de int que contiene el número de personas por grupo, siendo la posición de cada elemento del array el identificador del grupo. Es de tamaño grupos.

Todos los elementos descritos hasta el momento son estáticos, esto se ha implementado de esta forma para que puedan estar compartidos entre todos los estados siguientes. Esto se debe al hecho de que todos estos valores se obtienen con la creación del estado inicial o bien es información derivada de la generación de grupos y centros. Las matrices que contienen distàncias se han pre calculado de esta forma para evitar repetir cálculos innecesarios a lo largo

del proceso de generación de sucesores y cómputo del heurístico, que derivarían en un coste mayor que solo hace falta hacer una vez.

- **recogidas:** Esta matriz de matrices es el elemento principal de nuestro estado. Concretamente y con la finalidad de reducir el espacio en memoria se ha usado un ArrayList<ArrayList<ArrayList<Short>>>. Sigue esta estructura:

  [[[0,1], [6,7,8], ...],
  [[4,3], [2], ...],
  ...
  []]

  Cada helicóptero está representado en un ArrayList exterior mientras que cada salida de rescate queda representada por un vector interior con los id de los grupos dentro. Su tamaño es de como máximo grupos elementos repartidos con distintas configuraciones de vectores para cada estado. Se ha escogido esta representación ya que se cree que será más eficiente al tratar las inserciones, eliminaciones y swaps de grupos dentro de los propios trayectos de los helicópteros.

  Este elemento no es de tipo estático ya que varía para cada sucesor/estado dependiendo de las operaciones que se le apliquen para llegar hasta él y la creación del estado inicial como punto de partida.

La idea detrás de los criterios establecidos para crear nuevas estructuras y variables auxiliares dentro de la clase es intentar minimizar al máximo posible los tiempos de ejecución para encontrar soluciones. Por eso se cree que estas nuevas variables aportarán más rapidez en los cálculos al realizar solo accesos. Quedarà por ver, eso si el comportamiento espacial de esta decisión sobre problemas muy grandes donde si que se podría ver afectado el rendimiento por un mayor uso de memória.

2.1.1 Espacio de búsqueda

Nuestro espacio de búsqueda consta de todas las soluciones posibles que se pueden generar mediante la agrupación de grupos en distintas salidas de rescate.
Como el orden de los helicópteros no influye tendremos que nuestro espacio de búsqueda equivale a el número de grupos G*G como mínimo.

2.2 Operadores

En un primer instante se pensaron varios grupos de operadores, entre ellos distintas implementaciones para el swap y el move, además de operadores para eliminar salidas de rescate, añadirlas etc. Al empezar la implementación de la práctica apareció una simplificación terrible ya que nos dimos cuenta de que con dos operadores swap y move y sus funciones de aplicabilidad ya se podía generar soluciones suficientemente buenas como para que el algoritmo de búsqueda buscara y encontrara mínimos locales.

2.2.1 swapDestinations (int h1, int h2, int v1, int v2, int d1, int d2)

Este operador intercambia un grupo de rescate d1 con otro d2. Para localizar cada grupo usamos su posición dentro de la matriz recogidas, con h1 como la posición del helicóptero 1 que coincide

con el id de helicóptero, v1 como la salida de rescate dentro de ese helicóptero y d1 como la posición del grupo dentro de la salida (que no es lo mismo que el identificador de grupo). Análogamente para h2, v2 y d2.
Para poder aplicar esta operación hay que estar seguros de que el intercambio es posible, queriendo decir que al poner un grupo en otro viaje el número de personas de ese viaje es inferior a 15. Esto para los dos grupos que se intercambian con sus respectivas salidas de rescate.

El efecto de este operador tras su aplicación es que dos helicópteros intercambian grupos en una salida concreta. Es decir que un helicóptero h2, en la salida de rescate número v2 ahora pasa a recoger el grupo d1 en vez de él d2. Siendo igual para h1, v1 pero con d2.

El factor de ramificación es como mínimo el número de helicópteros ^ grupos.

2.2.2 moveDestinations (int h1, int h2, int v1, int v2, int g)

moveDestinations mueve un grupo de un cierto helicóptero y salida de rescate a un helicóptero y salida de rescate concreta. Es decir, coge un grupo concreto y lo sitúa dentro de otro. Como se comentará posteriormente la implementación interna se ha hecho teniendo en cuenta la solución inicial escogida que genera el número máximo de salidas de rescate para posteriormente reducirlas con el move o modificarlas con el swap. Por este motivo se cree que puede explorar suficiente espacio de soluciones y no hace falta además crear nuevas salidas de rescate, ya que en algún momento ya se podrá encontrar esa configuración. Eso sí, explorando muchos más nodos.

Para poder aplicar esta operación hay que tener en cuenta distintas restricciones, por ejemplo que en la salida de rescate donde se quiere mover el grupo haya menos de tres grupos y que quepa el grupo que se quiere mover conteniendo menos de 15 personas por salida de rescate.

El efecto de este operador tras su aplicación es que se mueve un grupo de una salida de rescate a otra. Quitando el grupo de la primera salida y/o helicóptero para añadirlo a la segunda salida.

El factor de ramificación es como mínimo helicópteros ^ grupos.

2.2.3 Elección de los operadores

Hemos escogido estos operadores ya que nos permiten explorar, a priori, todo el espacio de soluciones al usarlos juntos y teniendo en cuenta las reglas de aplicabilidad. También hay que tener en cuenta que estos operadores no generan soluciones no válidas ya que para usarlos se tienen en cuenta las restricciones que hacen un estado válido.

2.3 Análisis de la función heurística

Un estado solución se puede comparar con otro teniendo en cuenta diferentes factores, pero principalmente se tiene en cuenta el tiempo empleado en recoger a todos los grupos.
Los factores que intervienen principalmente en este hecho son:

- La distancia entre los centros y los grupos, convertida en tiempo.
- La distancia entre grupos convertida en tiempo.

- Tiempo de descanso entre salidas de rescate.
- Tiempo empleado para rescatar cada grupo una vez de se ha alcanzado.
- Prioridad de los grupos

2.3.1 Primer criterio

En el primer criterio se busca minimizar el tiempo empleado por los helicópteros para rescatar a todos los grupos (sin tener en cuenta que todos los helicópteros funcionan en paralelo). O lo que es equivalente, minimizar la suma de todos los tiempos empleados por cada helicóptero para recoger a todos los grupos asignados a ese helicóptero.

Esta función heurística principalmente va a iterar por todos los helicópteros sumando para cada salida de rescate el tiempo que tarda el helicóptero en ir desde del centro de rescate al primer grupo, recorrer los grupos restantes de esa salida cargando a las personas y volver. Sumando entre salidas de rescate de cada helicóptero 10 minutos.

La función devuelve el tiempo total en ms.

Como se calcula iterando por todos los helicópteros y grupos al final da igual el orden de los grupos dentro de una salida de rescate. Simplemente que las salidas de rescate sean las más óptimas para cada helicóptero haciendo la conjunción de todos los helicópteros la mejor solución en general. Un mejor tiempo respecto a otro estado indicará que los grupos están mejor repartidos en salidas y helicópteros. Causando el efecto de que el algoritmo de búsqueda siempre que pueda irá buscando una solución mejor con menos tiempo en general, sin tener en cuenta los grupos de prioridad.

2.3.2 Segundo criterio

Este segundo criterio busca además de minimizar la suma total del tiempo empleado por todos los helicópteros para recoger a todos los grupos, minimizar el tiempo empleado hasta recoger el último grupo de prioridad.

Esto además de repetir el criterio empleado en la anterior función heurística se va a sumar el mayor de los tiempos que tarde un helicóptero para recoger el último grupo de prioridad. Esto se consigue en la implementación buscando el máximo de los tiempos en recoger todos los grupos de prioridad 1 para cada helicóptero.

La función también va a devolver el tiempo en ms.

Se suma este último tiempo al tiempo total ya que un estado que tarda menos en recoger todos los grupos de prioridad 1 va a tener entonces un tiempo total menor. Ayudando así a que el algoritmo de búsqueda vaya escogiendo el mínimo.

El efecto que causa este heurístico en la búsqueda es que el algoritmo va a ir escogiendo soluciones que a priori minimicen el conjunto de los dos tiempos en general y en el caso de que se pondere el segundo tiempo (máximo en recoger el último grupo de prioridad) se van a ir priorizando siempre soluciones que pongan los prioritarios en las primeras salidas de rescate.

Solo se puede ponderar los elementos de la segunda heurística ya que es el único caso donde interviene más de un elemento en el cómputo del heurístico.

2.4 Elección y generación del estado inicial

Para este problema hemos considerado dos algoritmos para generar el estado inicial. Uno que reparte todos los grupos de la forma más homogénea posible entre todos los helicópteros haciendo una salida de rescate por grupo y otro donde se van rellenando también de forma homogénea pero intentando rellenar al máximo posible las salidas de rescate.

Se han escogido estas dos opciones ya que intentar buscar una solución mejor acercando grupos a helicópteros cercanos, por ejemplo, puede ser muy costosa de entrada y puede acabar derivando en una reducción del espacio de búsqueda.

Tampoco tenía sentido generar soluciones ordenando los grupos dentro de los helicópteros ya que al menos para el primer heurístico no influenciará en nada y aunque para el segundo si, puede ser también más costoso de generar.
Los dos algoritmos que hemos escogido basan sus propiedades en generar el máximo número de viajes posible de entrada, para después reducirlo o intentar agrupar grupos para después ordenarlos y expandirlos.

2.4.1 Solución inicial básica

Este es el pseudocódigo para generar la solución más básica:

```
recogidas = ArrayList(nHelicopteros)
desde id = 0 hasta nGrupos:
        viaje = ArrayList()
        viaje.add(id)
        recogidas.get(id % nHelicopteros).add(viaje)
```

Como se puede ver el coste de encontrar esta solución inicial es lineal en función de nHelicopteros.

2.4.2 Solución inicial más elaborada

Para este algoritmo el pseudocódigo es el siguiente:

```
desde id = 0 hasta nHelicopteros:
        viaje = ArrayList()
        viaje.add(id)
        recogidas.get(i).add(viaje)

desde id = nHelicopteros hasta nGrupos:
        helicoptero = recogidas.get(id % nHelicopteros)
        numPassajeros = 0
        para grupID en helicoptero.ultimoViaje()
                numPassajeros += personasGrupos[grupID]
```

```
si numPassageros + personasGrupos[id] <= 15 y helicoptero.ultimoViaje().size() < 3:
        helicoptero.ultimoViaje().add(id)
else:
        viaje = ArrayList()
        viaje.add(id)
        helicoptero.add(viaje)
```

Como se puede ver, el coste de encontrar esta solución inicial es lineal en función de nGrupos.

Dados los dos algoritmos y tras realizar varios experimentos como se puede ver más adelante la mejor solución es la más básica ya que es la que genera más viajes y le da a las operaciones más juego para poder eliminar salidas, mover grupos e intercambiarlos para generar la mejor solución.

3. Experimentos

En esta sección ejecutaremos el programa con diferentes operadores, soluciones iniciales, algoritmos y demás, para así comparar y encontrar la manera óptima de encontrar una solución al problema. Además, estudiaremos su comportamiento con diferentes valores de entrada.

3.1 Estudio de los operadores en Hill Climbing

Observación	Tenemos que encontrar la mejor combinación de operadores.
Planteamiento	Escogemos entre diferentes operadores y observamos sus resultados.
Hipótesis	Usar diferentes operadores mejorará la solución
Método	• Escogeremos 11 semillas aleatorias. • Probaremos cada semilla con diferentes operadores mediante **Hill Climbing**. • Cada ejecución será con 100 grupos, 5 centros y 1 helicóptero por centro. • Escogemos los mejores operadores para los siguientes experimentos.

Después de haber hecho los experimentos tal y como hemos dicho en la tabla anterior, los resultados han sido los siguientes:

	Heurística(ms)			Nodos expandidos		
Seed	Move&Swap	Move	Swap	Move&Swap	Move	Swap
1	164899919	180853472	267424878	115	76	54
2	172239612	183826128	258450394	106	75	53
3	184248498	195452605	291423156	107	73	50
4	168563933	178726212	259066556	119	83	49
5	155968659	170423862	262157906	106	82	48
6	146949199	155250513	231179050	110	84	49
7	156788650	174506628	265009452	109	73	50
8	162747824	181510024	279324704	114	69	50
9	141090110	150246356	242336258	119	83	52
10	142265517	152379173	214292938	103	90	47
1234	160802477	172795897	277386440	120	76	46

Figura 1: Tabla con los resultados del primer experimento

Para poder ver más claramente los resultados de los experimentos mostrados en la figura 1, vamos a mostrar un gráfico de cajas con los resultados, tanto de la heurística de la solución final (Figura 2), como del número de nodos expandidos para llegar a ella (Figura 3):

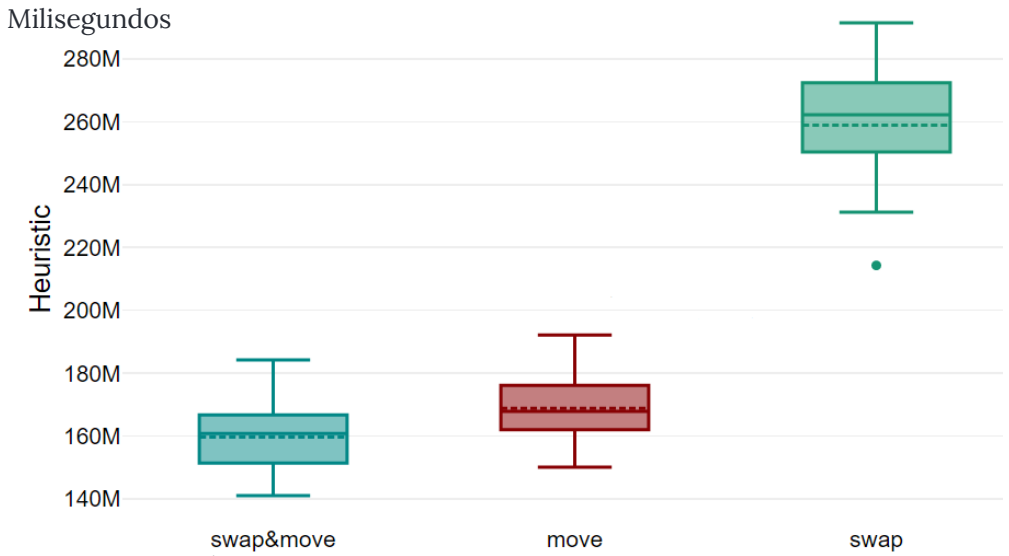

Figura 2: Gráfico de cajas de la heurística resultante para cada operador

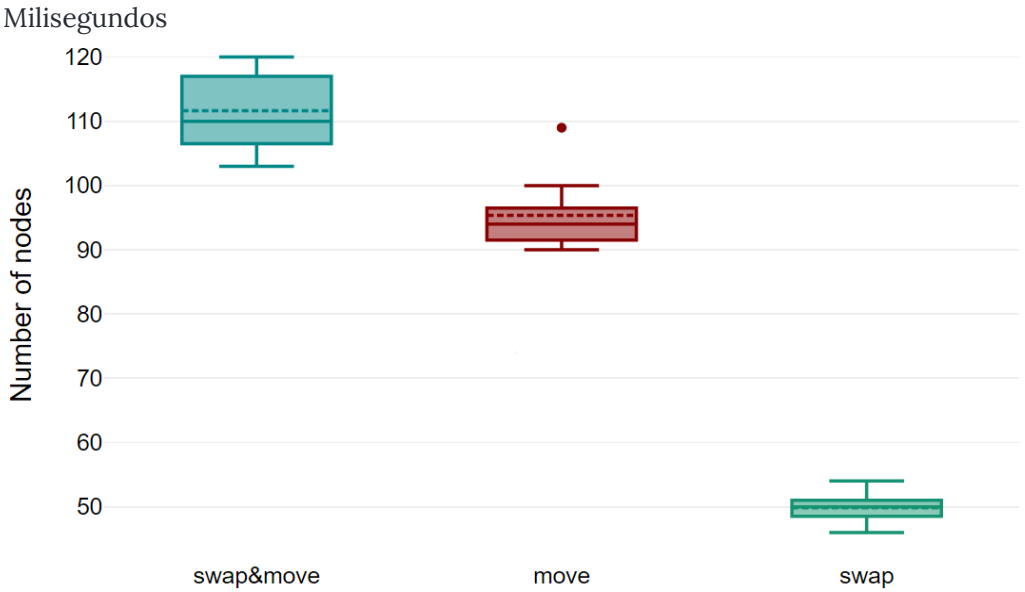

Figura 3: Gráfico de cajas del número de nodos expandidos para cada operador

Por último, mostraremos la media y la desviación estándar de los resultados en la siguiente tabla:

	Move&Swap	Move	Swap
Media heurística	159687(12472)	168855(11770)	258913(21256)
Media nodos	111.64(5.76)	95.36(5.19)	49.82(2.33)

Figura 4: Tabla con la media aritmética de la heurística, el número de nodos expandidos y sus desviaciones estándar (en segundos)

Podemos ver como el operador Swap por sí solo dan una heurística mucho peor comparada con las otras dos opciones, y casi no explora el espacio de soluciones debido a sus pocos nodos expandidos. Esto es esperado ya que este operador por sí solo no es capaz de agrupar varios grupos en un mismo viaje. Por esta razón, esta opción es descartada.

En cuanto a los otros dos, también podemos ver como Move da peores resultados que Move y Swap, tanto en la heurística final como en los nodos expandidos . A pesar de que el tiempo de ejecución de Move&Swap es mucho mayor que el de Move solo, consideramos que la heurística mejora lo suficiente como para ser escogido como la mejor opción.

En conclusión, elegiremos Move&Swap ante las otras dos opciones debido a sus mejores resultados y a su mayor exploración del espacio de soluciones, por lo que en este caso nuestra hipótesis era correcta.

3.2 Estudio de las soluciones iniciales en Hill Climbing

Observación	Tenemos que encontrar la mejor solución inicial.
Planteamiento	Escogemos entre diferentes soluciones iniciales y observamos sus resultados.
Hipótesis	Usar una solución que nos acerque más a la solución final nos dará mejores resultados.
Método	• Escogeremos 10 semillas aleatorias. • Probaremos cada semilla con las 2 soluciones iniciales implementadas mediante **Hill Climbing**. • Cada ejecución será con 100 grupos, 5 centros y 1 helicóptero por centro. • Escogemos la mejor solución inicial para los siguientes experimentos.

Los resultados de las ejecuciones han sido los siguientes:

	Heurística(ms)		Nodos expandidos	
Seed	BadSolution	BetterSolution	BadSolution	BetterSolution
1	164899919	171412969	115	89
2	172239612	174549921	106	107
3	184248498	188733250	107	105
4	168563933	170968433	119	87
5	155968659	166497392	106	106
6	146949199	150711882	110	92
7	156788650	160447332	109	108
8	162747824	168366077	114	97
9	141090110	146042649	119	114
10	142265517	140872122	103	109

Figura 5: Tabla con los resultados del segundo experimento

Igual que antes, mostraremos dos gráficos de cajas y una tabla con las medianas y las desviaciones típicas para poder ver los resultados con más claridad:

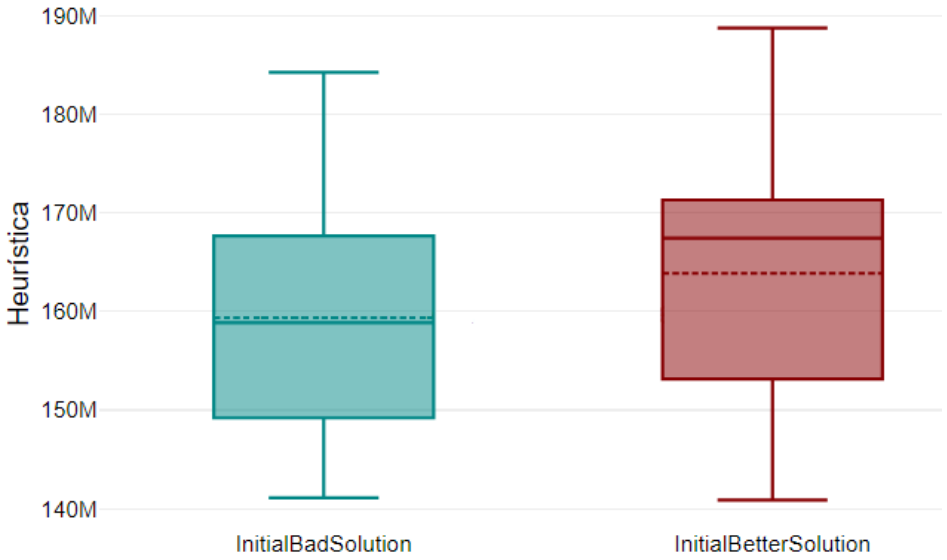

Figura 6: Gráfico de cajas de la heurística para cada solución inicial

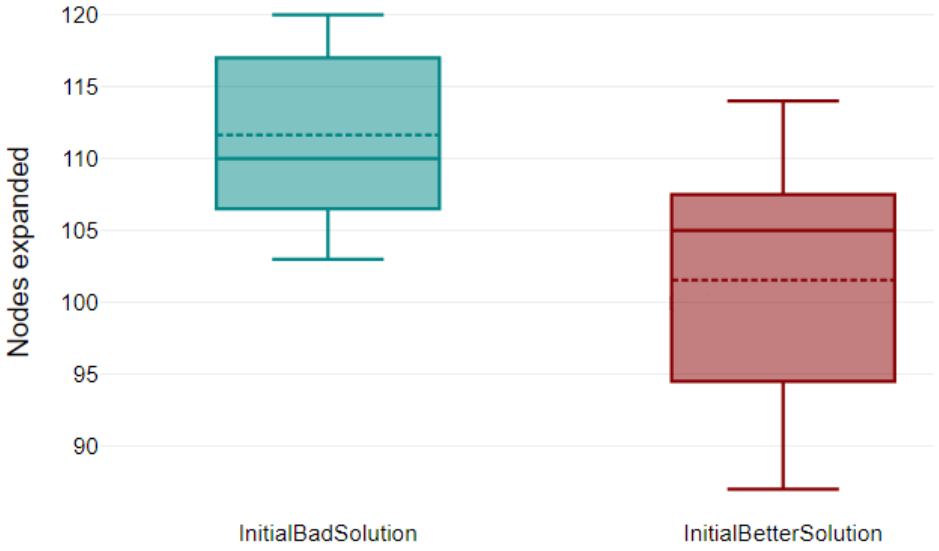

Figura 6: Gráfico de cajas de la heurística para cada solución inicial

	InitialBadSolution	InitialBetterSolution
Media heurística	159687(12472)	163860(13778)
Media nodos	111.64(5.76)	101.55(8.51)

Figura 7: Tabla con la media aritmética de la heurística, el número de nodos expandidos y sus desviaciones estándar (en segundos)

Nuestra hipótesis en un principio fue que la InitialBetterSolution, la cual intenta acercarse más a la solución real, daría mejores resultados, o los mismos pero más rápidos. No obstante, podemos ver como la solución inicial menos cercana a la real es la que da mejores resultados (aunque no más rápidos).

Creemos que esto es debido a que, como reparte todos los grupos a recoger en los helicópteros uno a uno y en diferentes viajes, la solución es creada desde 0, y toma las mejores decisiones desde el principio. Puesto que este es un problema que la mayoría de veces la opción óptima es la de mejor heurística, partir desde 0 y dejar que las tome lo antes posible es lo mejor para el resultado final.

No obstante, al igual que en el apartado anterior, estamos sacrificando tiempo de ejecución para conseguir una mejor solución, puesto que al partir de 0 la media de nodos a expandir es mayor.

En conclusión, usaremos a partir de ahora en todos los experimentos los operadores Move y swap, y como solución inicial InitialBadSolution, puesto que es la combinación que nos aporta mejores resultados finales.

3.3 Estudio de los parámetros K y λ en el Simulated Annealing

Observación	Tenemos que encontrar la mejor combinación de los parámetros K y λ que nos permita conseguir el mejor estado final.
Planteamiento	Elegimos distintos valores para K y λ, y observamos qué combinación da el mejor coste del estado final encontrado.
Hipótesis	Para valores más altos de λ y más pequeños de K obtendremos peores resultados.
Método	• Escogeremos 10 semillas aleatorias. • Ejecutaremos 12 experimentos por cada semilla, para toda combinación posible de K = 1, 5, 25, 125 y λ = 1, 0.1, 0.01 . • Cada ejecución será con 100 grupos, 5 centros y 1 helicóptero por centro. • Haremos las medias de los resultados obtenidos en las 10 ejecuciones de cada combinación. • Compararemos los resultados obtenidos.

Después de haber realizado el experimento utilizando el método explicado en la tabla anterior, hemos conseguido los resultados que se muestran en la siguiente tabla:

	K = 1	K = 5	K = 25	K = 125
λ = 1.0	185058174	185993458	186650333	186131686
λ = 0.1	163004674	160623937	161406232	162938649
λ = 0.01	161537905	163438591	161189040	162732522

Figura 8: Tabla con los resultados del tercer experimento

Para poder ver con más claridad cómo se comporta la variación de dichas variables, a continuación, podemos ver un gráfico tridimensional que nos facilitará ver dicha variación.

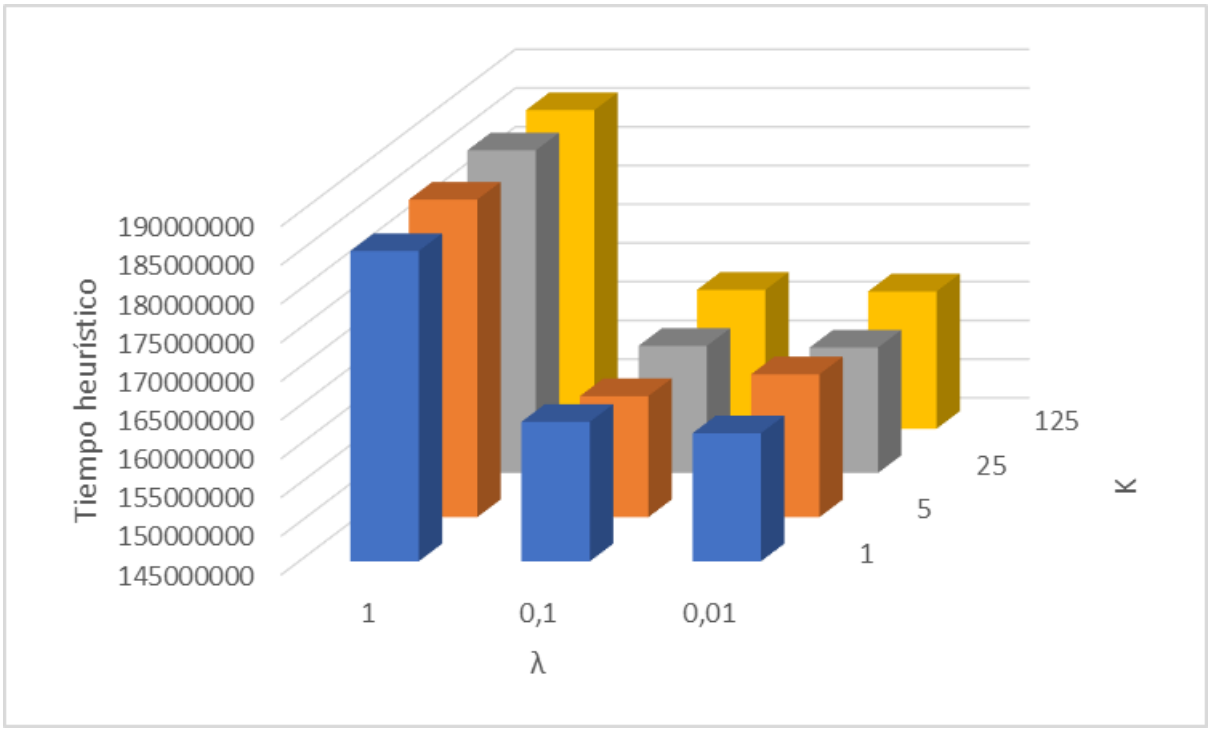

Figura 9: Gráfica con los resultados de los diferentes valores de K y λ

Como podemos ver observando el gráfico,la principal diferencia está al bajar la λ de 1 a 0.1. También podemos ver que con λ bajas, conseguimos mejores resultados si la K no es muy grande. Si nos fijamos en los resultados de la tabla, podemos ver que la mejor solución la conseguimos cuando K = 5 y λ = 0.1.

3.4 Estudio del tiempo de ejecución respecto al tamaño del problema

Observación	Queremos estudiar el crecimiento del tiempo de ejecución respecto al tamaño del problema.
Planteamiento	Aumentamos progresivamente el tamaño del problema para ver la evolución del tiempo de ejecución.
Hipótesis	El tiempo de ejecución del programa crecerá exponencialmente respecto al tamaño del problema.
Método	• Escogeremos una semilla (semilla 1 en este caso). • Ejecutaremos cada caso 10 veces y haremos el promedio. • Cada caso irá aumentando el tamaño del problema (centros y grupos) en una proporción de 5:100, y habrá 1 helicóptero por centro. • Estudiaremos los resultados y veremos los límites de nuestro programa.

Después de haber realizado el experimento utilizando el método explicado en la tabla anterior, hemos conseguido los resultados que se muestran en la siguiente tabla:

Tamaño(centros:grupos)	Hill Climbing (ms)	Simulated Annealing (ms)
5:100	3663	139
10:200	83145	154
15:300	485404	185
20:400	—	221
25:500	—	260.5

Figura 10: Tabla con los resultados del cuarto experimento

No hemos podido seguir con las ejecuciones del Hill Climbing debido a problemas con el heap de memoria de java. No obstante, se pueden ver las tendencias igualmente en las siguientes gráficas (Figura 11 y Figura 12):

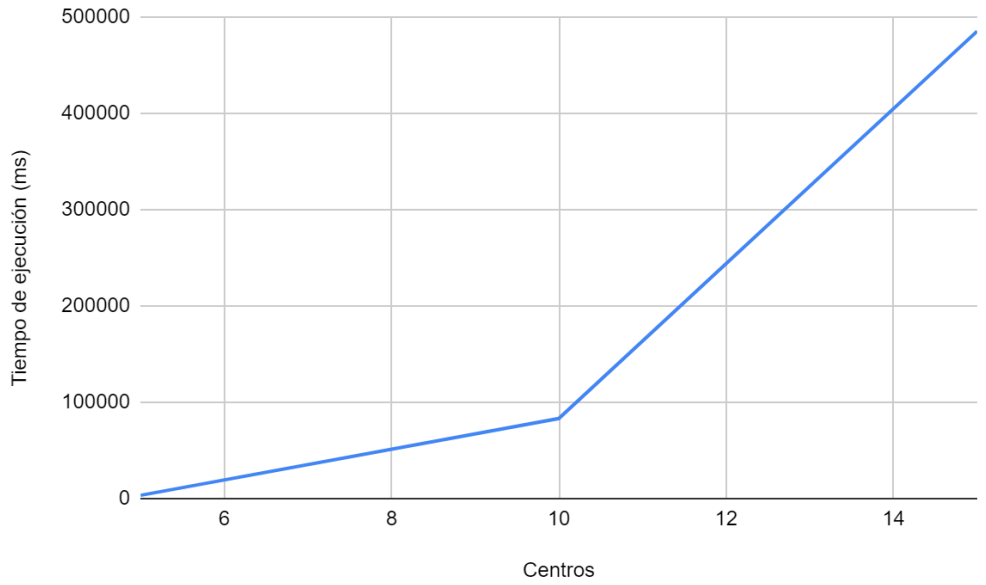

Figura 11: Gráfico con los resultados del Hill Climbing

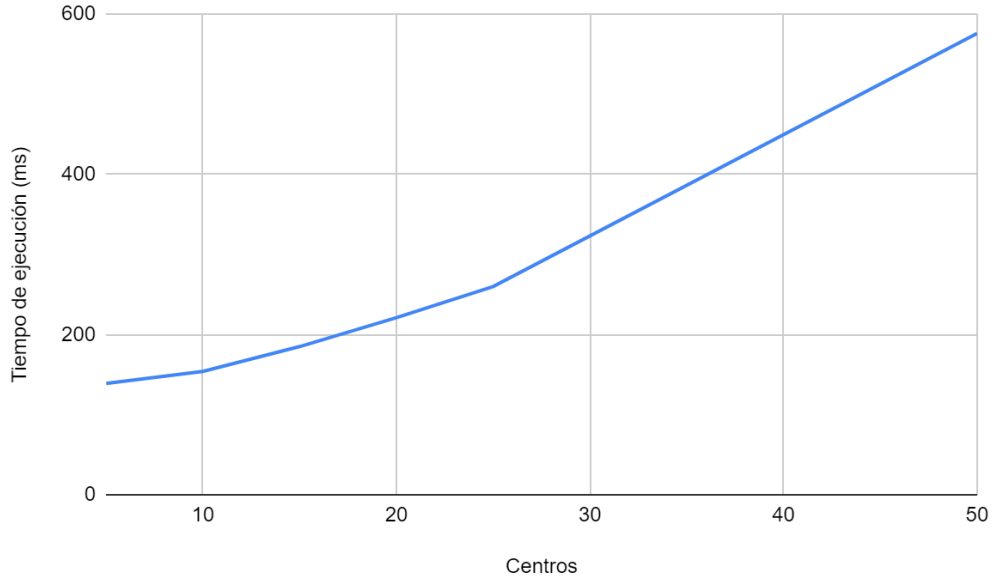

Figura 12: Gráfico con los resultados del Simulated Annealing

Los gráficos nos muestran que el Simulated Annealing es mucho más rápido que Hill Climbing, y permite usar tamaños mucho más grandes debido a que solo se genera un sucesor por generación. Además, el tiempo de ejecución del Hill Climbing aumenta exponencialmente, mientras que el de Simulated Annealing crece a un ritmo mucho más lento, casi linealmente.

No obstante, esto no quiere decir que los resultados finales heurísticos sean mejores, tema del cual hablaremos en experimentos posteriores.

No obstante, el problema de memoria del Hill Climbing no nos permite tener tamaños muy grandes, por lo que para poder utilizarlo se tendría que optimizar más.

3.5 Estudio del tiempo de ejecución respecto la cantidad centros y grupos

Observación	Queremos estudiar el crecimiento del tiempo de ejecución respecto el número de grupos y centros que hay
Planteamiento	Aumentamos progresivamente el número de grupos o centros para ver la evolución del tiempo de ejecución.
Hipótesis	Veremos un crecimiento del tiempo de ejecución exponencial cuando aumentemos el número de grupos pero en el caso de los centros el crecimiento será lineal.
Método	• Escogeremos 10 semillas aleatorias. • Ejecutaremos 12 experimentos en cada semilla, para los diferentes valores que comprobaremos • Cada ejecución será con 1 helicóptero por centro y se fijará el número de centros o grupos según el caso. • Haremos las medias de los resultados obtenidos en las 10 ejecuciones de cada combinación • Compararemos los resultados obtenido

Para realizar este experimento,aumentamos de 50 en 50 el número de grupos y de 5 en 5 el número de centros. A continuación podemos ver la tabla que nos muestra el resultado del experimento.

Aumentando el número de centros		Aumentando el número grupos	
Número de centros	Tiempo ejecución (ms)	Número de grupos	Tiempo ejecución (ms)
5	3352	100	3302
10	3509	150	12601
15	4051	200	51582
20	4075	250	136214
25	4175	300	280805
30	4435	350	549591

Figura 13: Tabla con los resultados del experimento 5

Observando la tabla podemos ver que nuestras hipótesis parecen ser bastante correctas, para poder ver mejor la tendencia de crecimiento del tiempo de ejecución, a continuación veremos unas gráficas de líneas obtenidas a partir de la tabla anterior.

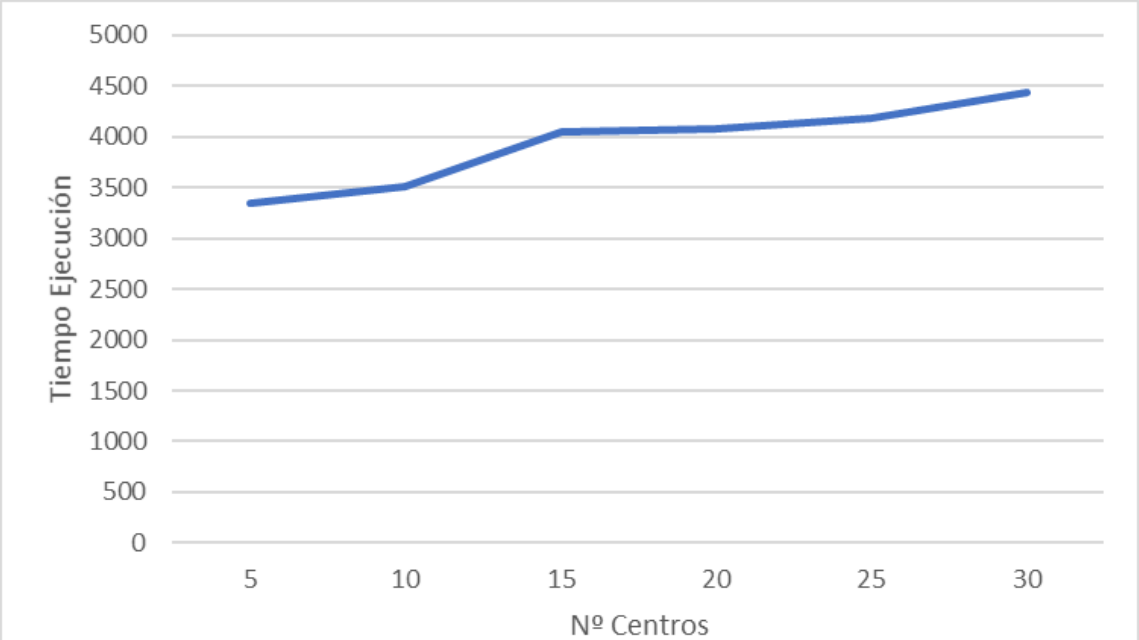

Figura 14: Gráfica con el crecimiento del tiempo de ejecución por nº centros

En esta primera gráfica, podemos ver que el crecimiento es bastante lineal, tal y como habíamos supuesto nosotros antes de hacer el experimento. También podemos ver que el crecimiento no es muy importante, pues a pesar de multiplicar por 6 el número de centros, solo se tarda 1 segundo más en ejecutarse el programa, que eso representa un crecimiento aproximado del 30%.

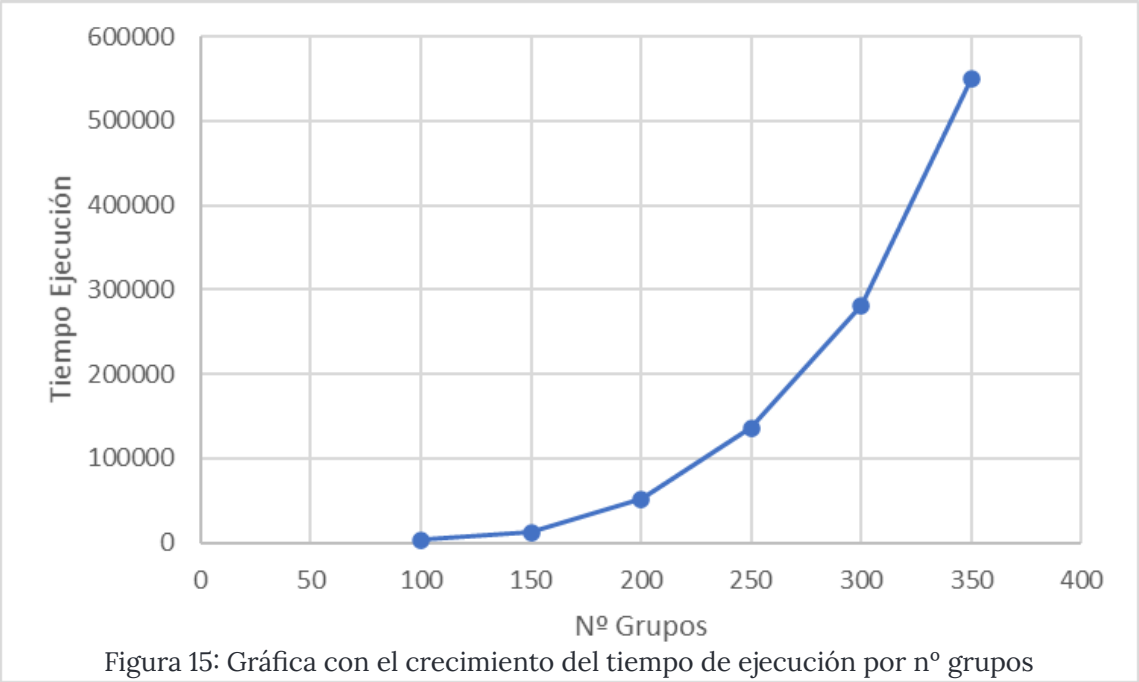

Figura 15: Gráfica con el crecimiento del tiempo de ejecución por nº grupos

En esta otra gráfica podemos ver como el tiempo de ejecución crece exponencialmente, como suponíamos en la hipótesis del experimento,

3.6 Estudio de la cantidad de centros y helicópteros cada 100 grupos

Observación	Tenemos que ver si es mejor aumentar el número de centros o el número de helicópteros por centro.
Planteamiento	Elegimos distintas cantidades de helicópteros, que estarán repartidas entre los mismos 5 centros o con un centro para cada helicóptero y observamos cómo afecta al tiempo de ejecución del programa y al coste del estado final encontrado.
Hipótesis	Conseguiremos mejores resultados aumentando el número de centros que aumentando el número de helicópteros por centro. Por lo que hace al tiempo de ejecución, no creemos que se aprecie un gran empeoramiento.
Método	• Escogeremos 10 semillas aleatorias. • Ejecutaremos 12 experimentos para los diferentes valores que comprobaremos • Cada ejecución será con 100 grupos y se fijará el número de centros o de helicópteros por centro según el caso. • Haremos las medias de los resultados obtenidos en las 10 ejecuciones de cada combinación (tiempos de ejecución). • Compararemos los resultados obtenidos.

Para realizar este experimento, cogimos los valores de 5, 10, 15, 20, 25 y 30 helicópteros, que en un caso implica dicho número de centros, y en el otro que tenemos 5 centros con 1, 2, 3, 4, 5 y 6 helicópteros en cada uno. A continuación podemos ver la tabla que nos muestra el resultado del experimento.

Aumentando el número de centros			Aumentando el número de helicópteros por centros		
Número de centros con 1 helicóptero	Tiempo ejecución (ms)	Heurística	Número de helicópteros por centros (5 centros)	Tiempo ejecución (ms)	Heurística
5	3352	164899919	1	3352	164899919
10	3509	136015071	2	3033	161269427
15	4051	130574116	3	3378	157641693
20	4075	124768974	4	3437	154601599
25	4175	117593799	5	3639	154247241
30	4435	118683468	6	3602	150053005

Figura 16: Tabla con los resultados del experimento 6

Para poder interpretar mejor los resultados, a continuación veremos unas gráficas de líneas que nos permitirán comparar más fácilmente los dos casos.

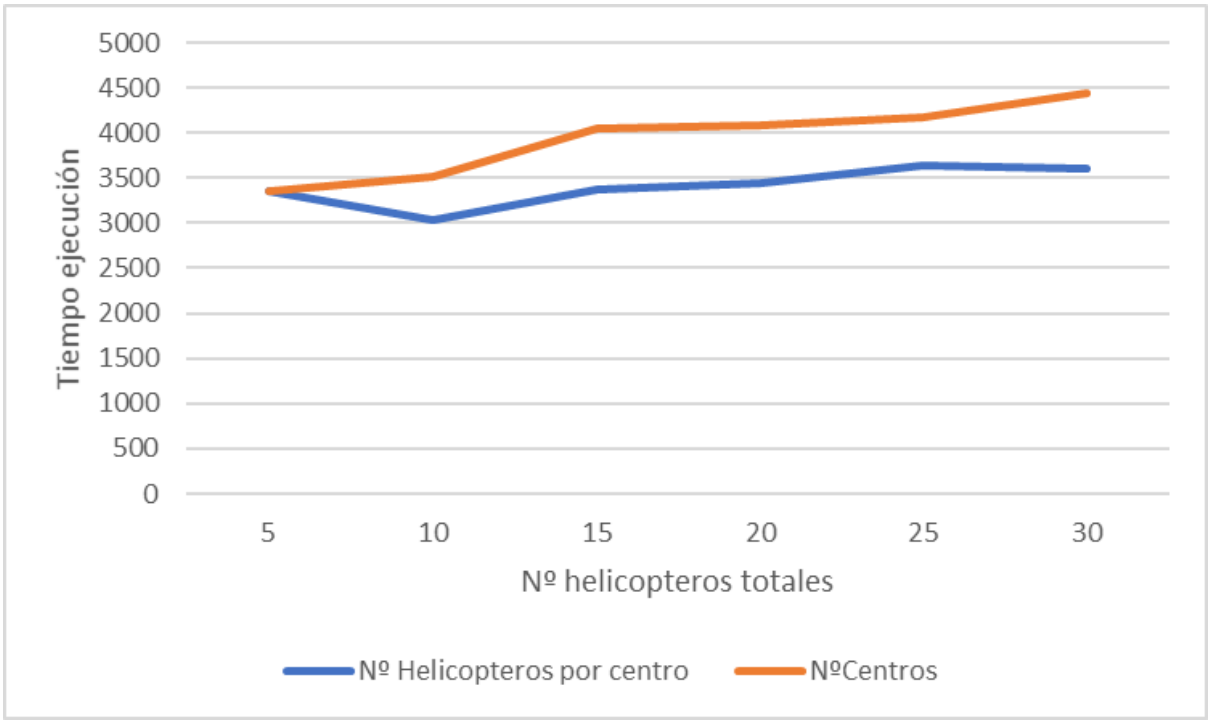

Figura 17: Gráfica que compara el tiempo de ejecución entre nº centros y nº helicópteros por centro

En esta primera gráfica, podemos ver que si mantenemos el número de centros, el tiempo de ejecución no se ve muy afectado, en cambio, al contrario de lo que pensábamos, este si que se ve afectado en el caso que aumentamos el número de centros pero mantenemos 1 unico helicoptero por centro.

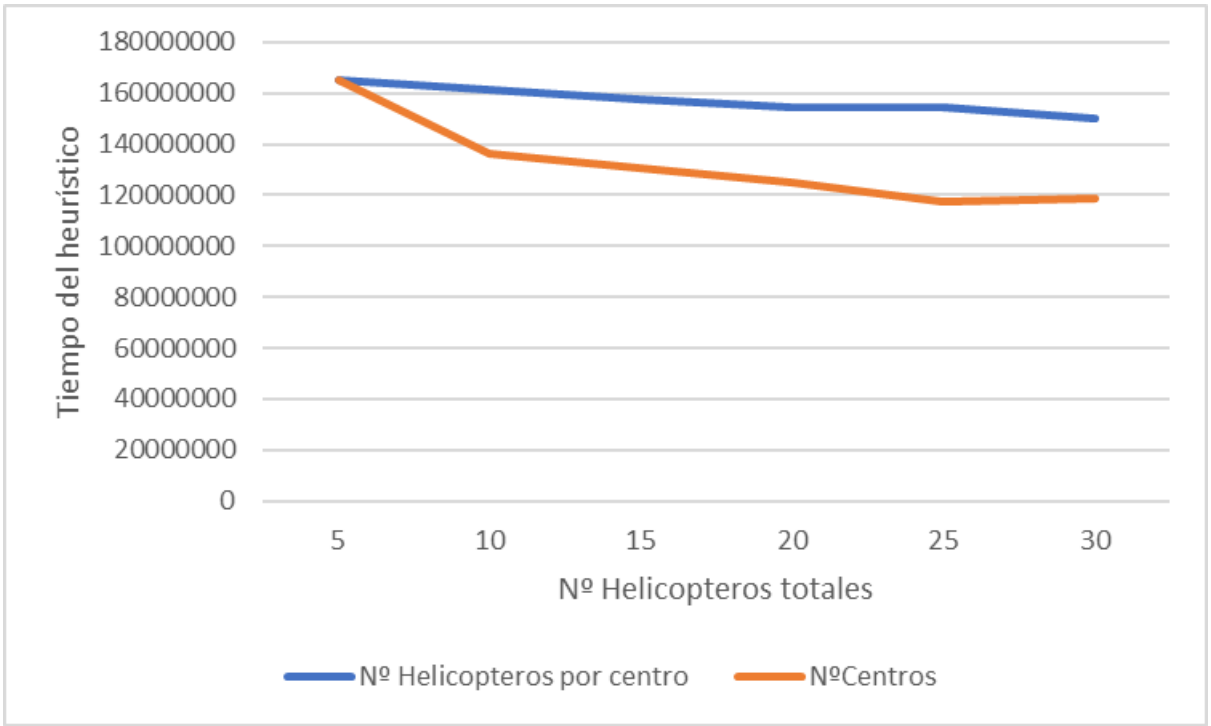

Figura 18: Gráfica que compara la heurística entre el nº centros y nº helicópteros por centro

En este segundo gráfico podemos ver que, tal y como esperábamos, el hecho de aumentar el número de centros era una mejor opción que aumentar el número de helicópteros por centro. Esto se debe al hecho que al haber más centros, puedes llegar desde más sitios a los grupos, y por lo tanto, es muy probable que los nuevos centros estén más cerca de algún grupo que el centro que antes estaba más cerca, o era la mejor opción.

3.7 Estudio de la segunda función heurística

La segunda función heurística usa 2 criterios distintos para dar su solución:

- El primer criterio es el mismo que la primera función heurística, la suma de todos los tiempos de los viajes de todos los helicópteros.
- El segundo criterio es el tiempo de vuelta del último grupo de prioridad 1 (con heridos).

La segunda función heurística devuelve la suma de ambos criterios, pero se puede poner una ponderación a cada uno para ver cómo cambian los resultados a partir de esta, que es lo que haremos en este último experimento.

Observación	Queremos estudiar la evolución del tiempo de recogida de todos los grupos al aumentar la ponderación del segundo criterio
Planteamiento	Aumentamos progresivamente la ponderación del segundo criterio para ver la evolución del tiempo total de recogida.
Hipótesis	Al aumentar más el segundo criterio, se tendrá menos en cuenta el tiempo total (primer criterio), por lo se tardará más en recoger todos los grupos.
Método	• Escogeremos 10 semillas aleatorias. • Ejecutaremos 2 experimentos por cada semilla para las diferentes ponderaciones (Uno con Hill Climbing y otro con Simulated Annealing). • En el caso de Simulated Annealing cada semilla será ejecutada 10 veces y se hará el promedio de ellas. • Haremos las medias de los resultados obtenidos en las semillas para calcular la heurística promedio. • Compararemos los resultados obtenidos entre la primera heurística, el Hill Climbing y el Simulated Annealing.

A continuación enseñamos la tabla con los resultados del experimento:

	Promedio heurísticas de 10 semillas (s)				
Algoritmos	Primer Heur.	Ponderado *1	Ponderado *2	Ponderado *3	Ponderado *4
Hill Climbing	159687	162341	164486	163463	165623
Simulated Annealing	160310	163560	163652	165790	166125

Figura 19: Tabla con los resultados del experimento 7

Para ver más claramente los resultados, los mostraremos en el gráfico de la figura 20:

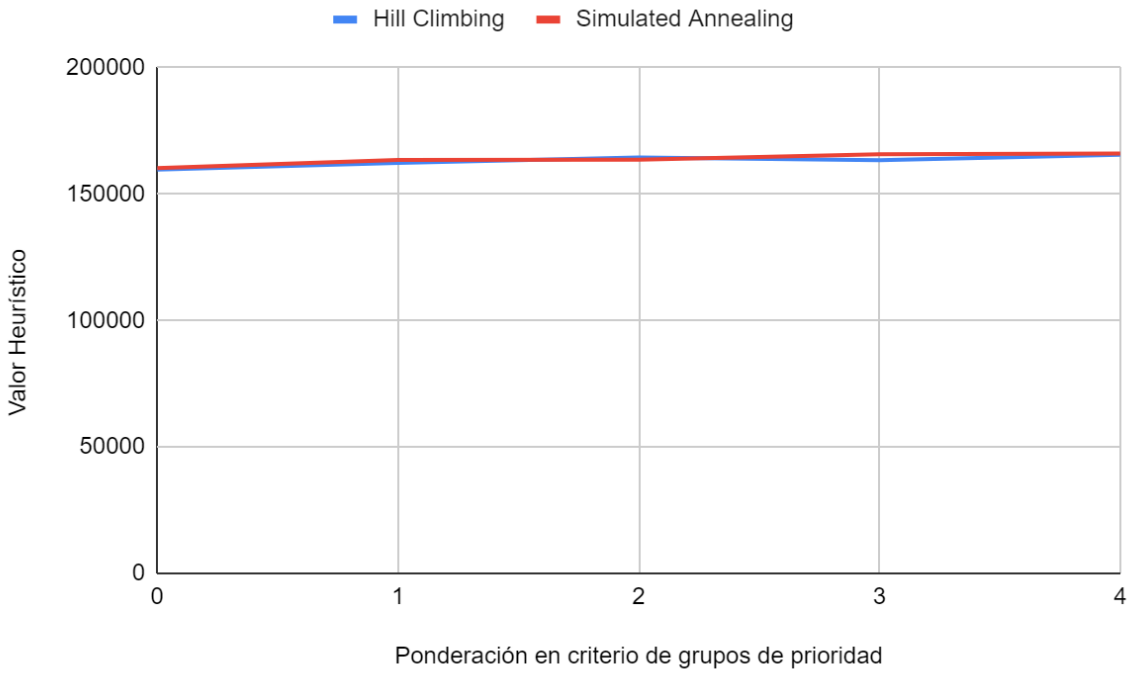

Figura 20: Gráfico con los resultados del experimento 7

En este gráfico y la tabla podemos ver como los resultados de la heurística del Hill Climbing y del Simulated Annealing son muy parecidos. No obstante, el valor devuelto por el Simulated Annealing varía mucho, por lo que a veces es peor o mejor que el Hill Climbing.

Además, se ve una clara tendencia a que cuanto mayor sea la ponderación del segundo criterio, mayor es el tiempo de recogida de todos los grupos. Esto es debido a que algunos sucesores que darían menor tiempo total son descartados por otros que hacen que el último grupo de prioridad llegue antes. Cuanto mayor es el segundo criterio, menos peso tiene el primero, por lo que se favorece más a los grupos de prioridad que al tiempo total, y más sucesores que favorecen al primer criterio son descartados.

En conclusión, cambiando las ponderaciones de cada criterio, podemos hacer que la solución final tenga en cuenta más uno u otro, creando una solución intermedia dependiendo de lo que necesitemos. Si queremos favorecer a los heridos, aumentamos la ponderación del segundo criterio, y si queremos favorecer que se recojan todos los grupos lo antes posible, la disminuimos.

4. Comparación entre los resultados de Hill Climbing y Simulated Annealing

Para finalizar los experimentos, vamos a comparar la eficiencia y resultados finales de ambos algoritmos y así ver cual es el óptimo. La gran parte de este estudio ya lo hemos visto en apartados anteriores: en el apartado 5 vimos que el tiempo de ejecución del simulated annealing es mucho menor al del Hill Climbing, tanto para casos pequeños como para casos grandes.

Además, en este último apartado (experimento 7) hemos visto también como la heurística promedio del Simulated Annealing es muy parecida a la del Hill Climbing, aunque un poco peor.

No obstante, el Simulated Annealing da un resultado aleatorio en cada ejecución, mientras que en el Hill Climbing siempre da lo mismo. Esto hace que a veces el Simulated Annealing consiga mejores soluciones que el Hill Climbing, como se puede ver, por ejemplo, en las 10 primeras semillas (Figura 21). Para crear esta tabla hemos encontrado las mejores soluciones del Simulated Annealing escogiendo la mejor de 10 ejecuciones, pero podríamos ejecutarlo más veces para encontrar un mejor resultado:

	Heurística	
Seed	Hill Climbing	Simulated Annealing
1	164899919	163529919
2	172239612	157967796
3	184248498	178314652
4	168563933	161075297
5	155968659	149263461
6	146949199	141350029
7	156788650	150791399
8	162747824	164271085
9	141090110	140977745
10	142265517	138217904

Figura 21: Tabla con resultados del Hill Climbing y los mejores resultados de Simulated Annealing

Para ver los resultados más claramente, haremos la tabla con los promedios:

	Media heurística(s)
Hill Climbing	159687
Simulated Annealing	154576

Figura 22: Tabla con promedios Hill Climbing y de Simulated Annealing

Con estos resultados podemos decir que, además de ser mucho más rápido en su ejecución que Hill Climbing, admitir entradas de datos mucho más grandes y crecer en tiempo de ejecución casi linealmente, si ejecutas Simulated Annealing varias veces y coges el mejor resultado también devuelve una mejor solución. Por lo tanto podemos concluir que en este problema Simulated Annealing es muy superior a Hill Climbing.

5. Trabajo de innovación

5.1 Descripción del tema

Grammarly es una compañía de software con distintas herramientas. Quizás la más conocida por todos los estudiantes es el corrector en inglés que ofrecen en su web y extensión. Está basado principalmente en técnicas de IA, concretamente una combinación de ML, DL y NLP.

5.2 Reparto del trabajo entre los miembros del grupo

Durante esta primera fase del trabajo de innovación principalmente se ha intentado aportar ideas entre todos sobre empresas y productos. Una vez valorados y escogido grammarly se ha procedido a buscar información en profundidad sobre los distintos puntos que requiere tratar el trabajo. Nos repartimos los puntos y buscamos la información entre todos.

5.3 Lista de referencias

A continuación se listan algunas de las referencias que se han usado como punto de partida para definir el proyecto de innovación:

Dambrauskaite, Elena. “How Was Grammarly Developed?” *Wiredelta*, 1 June 2021, https://wiredelta.com/how-was-grammarly-developed/

“What Types of Machine Learning Techniques Does Grammarly Use for Natural Language Processing?” *Quora*, https://www.quora.com/What-types-of-machine-learning-techniques-does-Grammarly-use-for-natural-language-processing

Ben Dickson, et al. “Grammarly Ai: The Sweet Spot of Deep Learning and Natural Language Processing.” *TechTalks*, 17 Oct. 2019, https://bdtechtalks.com/2019/10/17/grammarly-ai-assistant-grammar-checker/

Jamdade, Ankita. “ML/AI Case Study: Grammarly.” *LinkedIn*, LinkedIn, 20 Oct. 2020, https://www.linkedin.com/pulse/mlai-case-study-grammarly-ankita-jamdade/

Marr, Bernard. “The Amazing Ways Google and Grammarly Use Artificial Intelligence to Improve Your Writing.” *Forbes*, Forbes Magazine, 15 Nov. 2018, https://www.forbes.com/sites/bernardmarr/2018/11/12/the-amazing-ways-google-and-grammarly-use-artificial-intelligence-to-improve-our-writing/?sh=442f41cb3bb0

“How Grammarly Uses Natural Language Processing and Machine Learning to Identify the Main Points in a Message.” *How Grammarly Uses NLP & ML to Identify Main Points | Grammarly Engineering Blog*, 29 Sept. 2021, https://www.grammarly.com/blog/engineering/nlp-ml-identify-main-points/

"Adversarial Grammatical Error Correction." *Adversarial Grammatical Error Correction (GEC) | Grammarly Engineering Blog*, 30 Aug. 2021, https://www.grammarly.com/blog/engineering/adversarial-grammatical-error-correction/

"How We Use AI to Enhance Your Writing: Grammarly Spotlight." *Grammarly Spotlight: How We Use AI to Enhance Your Writing*, 17 May 2019, https://www.grammarly.com/blog/how-grammarly-uses-ai/

5.4 Dificultades

Encontrar información concreta sobre qué modelos o técnicas específicas usan para productos es la parte más complicada del proyecto. Son una empresa muy conocida con una gran cantidad de artículos sobre AI en su blog, pero mueren entre la divulgación y la concreción.