Práctica de Búsqueda Local: Red de sensores y centros de datos

Inteligencia Artificial

Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

Práctica realizada por:
Marc Benedí San Millán
Martí Homs Soler
Hermes Valenciano Farré

ÍNDICE

1. Análisis del problema

1. Descripción del problema

El problema que se plantea consiste en determinar cómo se deben interconectar ciertos elementos capaces de recibir/transmitir información que están distribuidos a lo largo y ancho de una geografía con el propósito de cumplir ciertos criterios que se expondrán seguidamente.

2. Características y elementos relevantes

Los elementos que deberemos interconectar para formar la red anteriormente mencionada se pueden dividir principalmente en dos tipos: sensores y centros de datos.

- Los **sensores** son elementos capaces de tres cosas:
    - registrar información del mundo real
    - enviar información mediante una línea de comunicación cableada
    - recibir información de otros sensores mediante una línea de comunicación cableada por cada conexión

  Los sensores estarán colocados sobre un punto **fijo** perteneciente a la geografía sobre la que se desarrolla el problema.
    - Los sensores se pueden subdividir en 3 grupos:
        - los que son capaces de capturar 1Mb/s
        - los que son capaces de capturar 2Mb/s
        - los que son capaces de capturar 5Mb/s
    - Un sensor deberá tener obligatoriamente una conexión de salida por la que enviar la información que recoge más, opcionalmente, la que recibe de otros sensores.
    - Un sensor deberá tener un mínimo de 0 y un máximo de 3 conexiones de entrada.

  La información que un sensor (llamémoslo s) puede recibir de entrada de otros sensores está limitada por la clase a la que *s* pertenece: en concreto, podrá almacenar hasta dos veces la cantidad de datos que registra.
  Si un sensor recibe más información de la que puede almacenar, esta información se pierde.
  El envío de datos se produce por medio de un sistema que utiliza entrelazado cuántico, por lo que no es necesario tener en cuenta el tiempo de envío.

- Los **centros de datos** son elementos capaces de recibir información proporcionada por sensores a través de una línea de comunicación cableada por cada conexión.
  Los centros de datos estarán colocados sobre un punto **fijo** perteneciente a la geografía sobre la que se desarrolla el problema.
  La información que un centro de datos puede recibir está limitada por:
    - cantidad: no más de 150 Mb/s
    - número de conexiones entrantes simultáneas: 25

Típicamente, el número de centros de datos será menor que el de sensores.

Para hacer un análisis del objetivo del problema, antes deberemos saber que todas las conexiones cableadas que se realicen tendrán un coste que será el producto de la longitud de la línea al cuadrado por el volumen de datos transmitidos.

Se quiere resolver el problema para una instancia concreta, para ello se dispondrá de las coordenadas de los centros de datos y sensores y el volumen de datos que estos capturan.
Una solución ha de determinar con quién tiene que conectarse cada sensor (otro sensor o un centro de datos).

El objetivo es obtener la red de conexión que tenga el menor coste y recoger el máximo de datos, todos si es posible.

3. Análisis del enfoque: Búsqueda local

Pensamos que el enfoque correcto para este problema es el de búsqueda local porque cumple las siguientes condiciones:

- Se puede generar fácilmente una solución que no sea óptima e ir mejorándola iterativamente
- Podemos obtener la información que nos dice cuán buena o mala es una solución y así guiar a un heurístico
- Lo que nos interesa es la solución final a la que se llega tras haber aplicado las mejoras, y no el camino que se ha seguido hasta llegar a ella.
- Los elementos del problema son fácilmente cuantificables (conexiones, sensores, centros, volumen de datos, distancias), por lo tanto la granularidad de los operadores estará bien definida

2. Descripción/Justificación de la implementación del estado

1. Constantes

En el estado inicial tenemos las siguientes **constantes**:

```
private static int numCenters;
private static int numSensors;
private static int seed_c;
private static int seed_s;
private static final int maxFlowCenter = 125;
private static final int inputMaxCenter = 25;
private static final int inputMaxSensor = 3;
private static int notConnected;
```

Estas constantes nos permiten que el código sea más legible y también que sea más rápido cambiar los parámetros del problema. Las variables ***numCenters, numSensors, seed_c, seed_s*** se pasan como parámetros al ejecutable. La variable ***notConnected = numCenters - 1*** nos sirve para inicializar un array de conexiones a un estado de no conectado a nada.

2. Variables estáticas (de clase)

Tenemos las siguientes variables **estáticas de clase** (compartidas entre todos los estados):

```
//Total amount of data that sensors are collecting.
private static int dataEmitted;
//Index from 0 to numCenters-1
private static CentrosDatos centers;
//Index from 0 to numSensors-1
private static Sensores sensors;
//Distance matrix. Rows = numSensors, Columns = numCenter + numSensors. (The first columns reference centers)
private static double[ ][ ] distances;
//Size S + C
//This is equivalent to sensor.get(i).getCapacidad();
private static int[] collectedDataVolume;
```

Esta información es redundante, pero aún así hemos preferido tenerla en un array por la velocidad del acceso. (Al ser una variable estática no representa un coste relevante porque solo se tiene una vez).

3. Variables que representan y distinguen el estado

Finalmente tenemos las variables que **representan y distinguen el estado**. Estas son las variables que tendrán todos los estados.

// Size = S
private int[ ] **connectedTo**;
Este array nos indica que el sensor de la posición *i* está conectado a:

- Si el valor *v* de la posición *i* es no negativo significa que el sensor i-ésimo está conectado al sensor que ocupa la v-ésima posición en la estructura Sensors.
- Si el valor v es negativo, el sensor i-ésimo está conectado al centro *i + numCenters.*

// Size S + C
private int[ ] **inputConnections**;
Este array guarda el número de conexiones entrantes que tiene el elemento con índice *i.* Si el índice está entre 0 y numCenters representará un centro, y si el índice está entre numCenters y numCenters + numSensors representará un sensor.

// Size S + C
private int[ ] **inputFlow**;
Este array guarda el volumen de datos que recibe el elemento del índice *i.* Si el índice está entre 0 y numCenters representará un centro, y si el índice está entre numCenters y numCenters + numSensors representará un sensor.

Hay que remarcar que **inputFlow** se diferencia de **nonRestrictedInputFlow** en que el primero es el mínimo entre el volumen de datos que recibe y el máximo que puede re-enviar sin tener pérdidas.

// Size S + C
private int[ ] **nonRestrictedInputFlow**;
Este array guarda el volumen de datos que recibe el elemento del índice *i.* Si el índice está entre 0 y numCenters representará un centro, y si el índice está entre numCenters y numCenters + numSensors representará un sensor.

Como se ha comentado anteriormente, este array guardará todo el volumen de datos que se recibe, aunque no sea posible almacenarlo en su totalidad.

Justificación: hemos decidido mantener esta información que a priori no habíamos seleccionado como imprescindible porque nos representa un ahorro muy importante a la hora de recalcular el de volumen de datos entrantes cuando se produce una desconexión.

Usamos arrays en vez de objetos para minimizar el coste en memoria ya que el hecho de trabajar con objetos en Java implica un coste en memoria mucho mayor que el de guardar **int.**

3. Descripción/Justificación de los operadores elegidos

Los operadores que proponemos para abordar el problema son 2:

- Change connection
- Swap connections

1. Change connection

La idea del operador *change connection* es cambiar el destino de la conexión saliente de un sensor *x*. Supongamos que inicialmente el sensor x está conectado al elemento *y*. Al aplicar el operador change connection sobre el sensor *x*, este dejará de estar conectado al elemento *y*, y se conectará a otro con conexiones entrantes disponibles.

Condiciones de aplicabilidad

Para poder aplicar este operador sobre un sensor *x* y una destinación *d*, se debe cumplir que:

- *x* no está conectado a *d* previamente, y
- o bien *d* es un centro y tiene conexiones entrantes disponibles,
- o bien *d* es un sensor, tiene conexiones entrantes disponibles y no está conectado directa o indirectamente a *x* (lo que nosotros hemos llamado que *d* no dependa de *x*)

Hemos impuesto estas condiciones de aplicabilidad para evitar y prevenir que se formen ciclos en la red y mantenernos en el espacio de soluciones.

La aplicación de este operador no nos sacará en ningún caso del espacio de soluciones ya que debido a las condiciones de aplicabilidad, no se violarán las restricciones de conexiones entrantes y además no se formarán ciclos, de forma que en todo momento, todos los sensores estarán conectados directa o indirectamente con algún centro.

Análisis del factor de ramificación

Viendo el funcionamiento del operador, se puede deducir rápidamente que una **cota superior** del factor de ramificación es:

$$número\ de\ sensores \cdot (número\ de\ centros + número\ de\ sensores\ - 2)$$

ya que no queremos que un sensor pueda conectarse a sí mismo, y además el sensor ya estará conectado a otro elemento.
No obstante, dependiendo de la configuración de la red (solución intermedia) en la que nos encontremos, el factor de ramificación será menor, ya sea porque haya sensores que dependan de otros o porque no todos los elementos tendrán conexiones entrantes disponibles.

1. Swap connections

La idea del operador *swap connections* es intercambiar los destinos de dos conexiones salientes de dos sensores *a* y *b*. Supongamos que el sensor a está inicialmente conectado al elemento *x,* y que el sensor *b* está inicialmente conectado al elemento *y*. Una vez aplicado el operador sobre los sensores *a* y *b,* el sensor *a* estará conectado a *y,* y el sensor *b* estará conectado a *x*.

Condiciones de aplicabilidad

Para poder aplicar este operador sobre dos sensores *a* y *b,* se debe cumplir que:

- *a* y *b* sean dos sensores diferentes, y
- las conexiones salientes de *a* y *b* no tengan el mismo destino, y
- *a* no esté conectado directa o indirectamente a *b,* y
- *b* no esté conectado directa o indirectamente a *a*.

Hemos impuesto estas condiciones de aplicabilidad para evitar y prevenir que se formen ciclos en la red y mantenernos en el espacio de soluciones.

La aplicación de este operador no nos sacará en ningún caso del espacio de soluciones ya que debido a las condiciones de aplicabilidad, no se violarán las restricciones de conexiones entrantes y además no se formarán ciclos, de forma que en todo momento, todos los sensores estarán conectados directa o indirectamente con algún centro.

Análisis del factor de ramificación

Viendo el funcionamiento del operador, se puede deducir rápidamente que una **cota superior** del factor de ramificación es:

$$(n\ sobre\ 2) = \frac{número\ de\ sensores!}{2! \cdot (número\ de\ sensores - 2)!} = \frac{n \cdot n-1}{2}$$

ya que como máximo podremos aplicar el operador sobre todas las parejas de sensores.

No obstante, dependiendo de la configuración de la red (solución intermedia) en la que nos encontremos, el factor de ramificación será menor, ya sea porque haya sensores que dependan de otros o porque haya sensores que estén conectados al mismo elemento.

3. Swap +

El Swap + es el conjunto de operadores resultante de la combinación de los dos operadores anteriores. La idea de combinarlos es evitar las limitaciones de los dos anteriores y al mismo tiempo adquirir sus ventajas.

El problema del *change connection* es que, si por algún casual, hay una buena solución a la que hay que llegar intercambiando dos conexiones, pero en los pasos intermedios la solución no es buena, no se explorará esa zona del espacio, debido a que el heurístico solo puede saber el valor del heurístico aplicando un cambio, y no una composición de ellos.

El problema del *swap connections* es que todos los cambios de conexión se hacen de dos en dos, y por lo tanto, si hay alguna buena solución que requiere un número impar de cambios, no se llegará a ella. Con lo que únicamente con este operador no podemos explorar todo el espacio de soluciones.

Por lo tanto al usar el Swap +, evitamos las limitaciones de los dos operadores anteriores, y nos beneficiamos de sus ventajas, ya que el heurístico decidirá cuando sea más conveniente aplicar uno que otro.

Además, el uso de este conjunto de operadores nos garantiza que no nos vamos a salir del espacio de soluciones, ya que ninguno de los operadores que lo componen lo permite, como se ha dicho anteriormente en cada una de sus descripciones.

Análisis del factor de ramificación

Sabiendo el factor de ramificación de los dos operadores que componen este conjunto, podemos determinar que **una cota superior** del factor de ramificación del conjunto Swap + es:

$$ns \cdot (nc + ns - 2) + \frac{ns \cdot ns-1}{2}$$

siendo ns el número de sensores y nc el número de centros de la instancia del problema.

ya que es la suma de los factores de ramificación de los operadores que componen el conjunto.

No obstante, dependiendo de la configuración de la red (solución intermedia) en la que nos encontremos, el factor de ramificación será menor, debido a las condiciones de aplicabilidad de los dos operadores.

4. Descripción/Justificación de la estrategia para hallar la solución inicial

1. Estrategia Greedy

La intención de esta estrategia para la generación de una solución inicial es partir de una solución medianamente buena, para que el algoritmo se ahorre explorar las zonas del espacio de soluciones donde las soluciones son poco óptimas.

El algoritmo de implementa esta estrategia se resume en los siguientes pasos:

1. Primero, se crea una cola de prioridad para cada centro y en ella se introducen todos los sensores, ordenados primero por capacidad (1, 2 o 5, primero los más capaces) y después por distancia (primero los más cercanos) a ese centro.
2. Después se comienzan a llenan las conexiones de cada centro siguiendo esa cola de prioridad y respetando las restricciones sobre el número de conexiones entrantes.
3. Si cuando ha terminado el paso 2 aún quedan sensores por conectar, éstos se conectarán por niveles (o en capas) a los sensores que ya estén conectados, siempre respetando las restricciones sobre el número de conexiones entrantes.

Análisis del coste de generar esta solución inicial

El coste de generar esta solución inicial es básicamente el coste de recorrer cada centro y para cada uno, consultar la distancia a cada sensor e insertarlo en su cola de prioridad.

La inserción en la cola de prioridad si se hace con un Heap tiene coste logarítmico en la altura del árbol. Por lo tanto la primera parte tendría coste: nc * ns * log(ns)

Por último, se produce una fase que dura hasta que no quedan sensores por conectar, por lo tanto es proporcional al número de sensores, pero como este número es menor al coste de la primera fase, el coste temporal en notación asintótica que tiene el algoritmo para generar la solución inicial greedy es:

$$O(nc \cdot ns \cdot log(ns))$$

Justificación de que es solución

La red que genera el algoritmo que implementa esta estrategia está en el espacio de soluciones ya que todos los sensores están conectados a un centro directa o indirectamente, y no se violan las restricciones de conexiones entrantes de ningún centro ni sensor.

Limitaciones

Una de las limitaciones obvias que le hemos encontrado a esta estrategia es que es posible que al aplicar el algoritmo, la solución que genere se encuentre en un mínimo local en el que el heurístico tenga poca libertad para explorar soluciones mejores. Dicho de otra forma, es posible que el algoritmo haga demasiado trabajo, en concreto el trabajo que le corresponde al heurístico.

2. Conectarlos todos en línea

La intención de esta solución inicial es que el estado inicial no sea bueno para dejar al algoritmo Hill Climbing espacio para trabajar y no hacer que empiece condicionado por un potencial mínimo local.

El algoritmo de implementa esta estrategia se resume en los siguientes pasos:

1. Recorrer todos los sensores y conectar el i-ésimo al (i+1)-ésimo
2. Conectar el último sensor a un centro de datos

Es importante señalar que a la hora de realizar las conexiones no tenemos en cuenta ni distancias entre los sensores ni el volumen de datos que cada sensor es capaz de transmitir.

Análisis del coste de generar esta solución inicial

El coste que tiene generar esta solución inicial es básicamente el coste que tiene recorrer los sensores, así que, en notación asintótica, el coste temporal será lineal en el número de sensores:

$$\Theta(\textit{número de sensores})$$

Justificación de que es solución

Al igual que con la estrategia anterior, la red que genera el algoritmo que implementa esta estrategia está en el espacio de soluciones ya que todos los sensores están conectados a un centro directa o indirectamente, y no se violan las restricciones de conexiones entrantes de ningún centro ni sensor.

Limitaciones

Las limitaciones de esta estrategia son claras: si bien la red que proporciona está en el espacio de soluciones, la solución que genera es de una calidad ínfima, y esto afecta a que el algoritmo que se ejecute tendrá que expandir muchos nodos antes de llegar a una solución de calidad media.

5. Descripción/Justificación de las funciones heurísticas

Las funciones heurísticas que hemos pensado (o usado) son las siguientes:

1. Función heurística coste

No es una función heurística viable ya que no tiene en cuenta el volumen de datos que se recibe en los centros de datos. Recordar que entre los objetivos del problema estaban minimizar el coste y maximizar el volumen de datos recibido. Este algoritmo cumple la primera parte de minimizar el coste, sin embargo no tiene en cuenta la segunda. Esta función heurística ha sido usada en el experimento especial 8 ya entregado.

2. Función heurística coste/datos

Esta función heurística cumple uno de los objetivos del problema ya que intentará minimizar el coste de la red y aumentar los datos. El problema es que al ser un cuociente estamos trabajando con dos variables de valor muy diferente y no será significativo.

3. Función heurística resta

Finalmente se decidió que la mejor forma de trabajar con los dos valores era encontrar una ponderación para cada uno y despues restarlos de la siguiente forma:

$$pond_c \cdot coste - pond_d \cdot datos$$

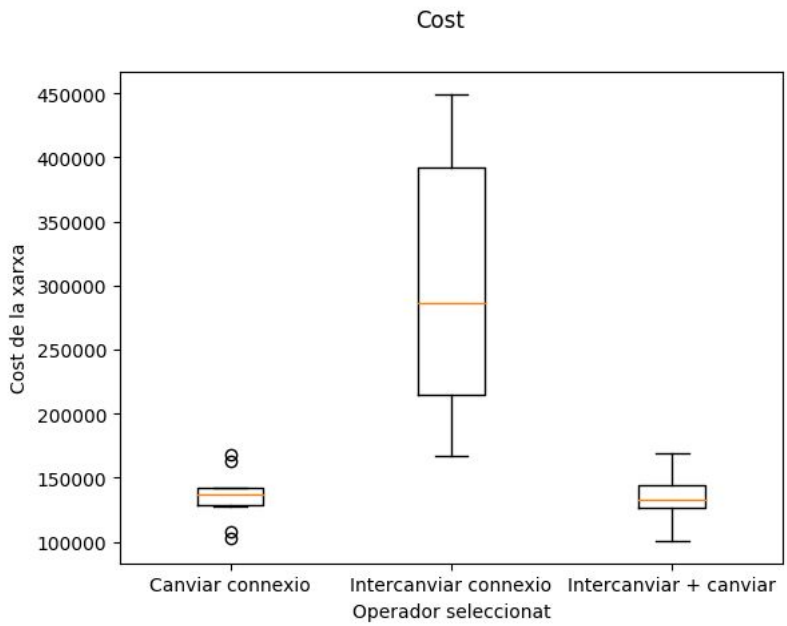
Después de experimentar con los valores de las ponderaciones se les ha asignado este valor:

$$pon_c = 0.1$$
$$pon_d = 35000$$

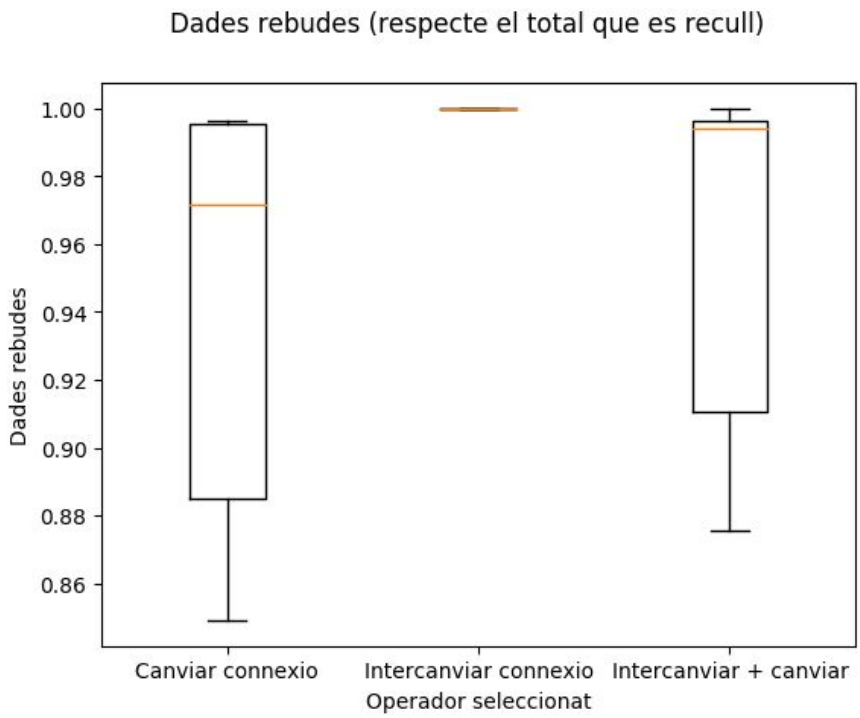
6. Experimentos

1. Comparación de conjuntos de operadores

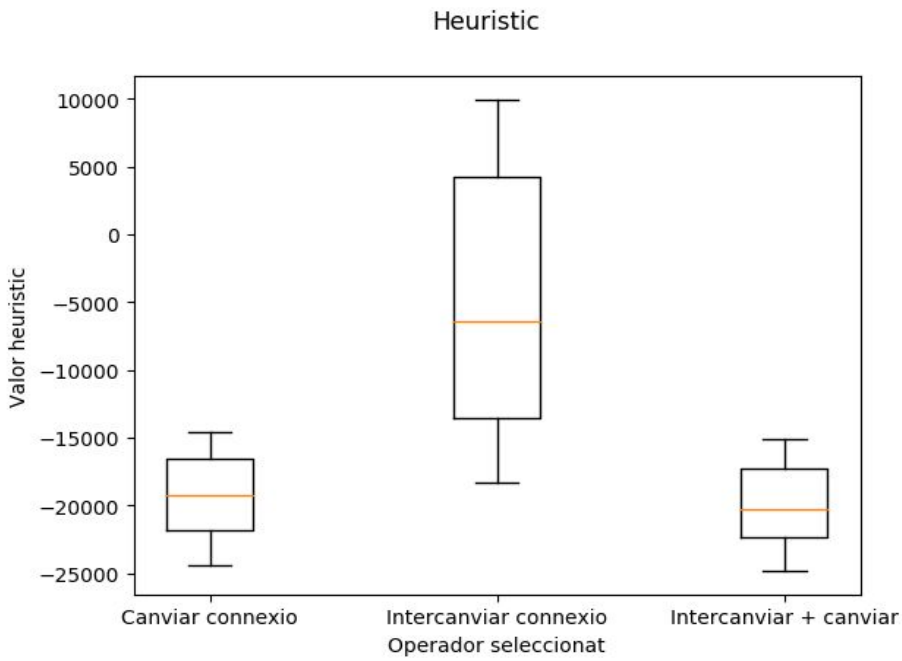
Observación	Pueden haber conjunto de operadores que sean mejores que otros.
Planteamiento	Escogemos diferentes grupos de operadores y analizamos las diferentes soluciones que nos dan.
Hipótesis	Nula (H0): todos los conjuntos de operadores dan los mismos resultados. Alternativa (H1): hay conjuntos de operadores mejores que otros.
Método	• Elegiremos 10 semillas arbitrarias para generar centros y otras 10 diferentes para generar sensores, una para cada réplica. • Ejecutaremos 1 experimento para cada par de semillas y cada conjunto de operadores. • Experimentamos con problemas de 4 centros y 100 sensores. • Usaremos el algoritmo de Hill Climbing. • Usaremos el generador de solución inicial greedy. • Mediremos diferentes parámetros para realizar la comparación a posteriori.



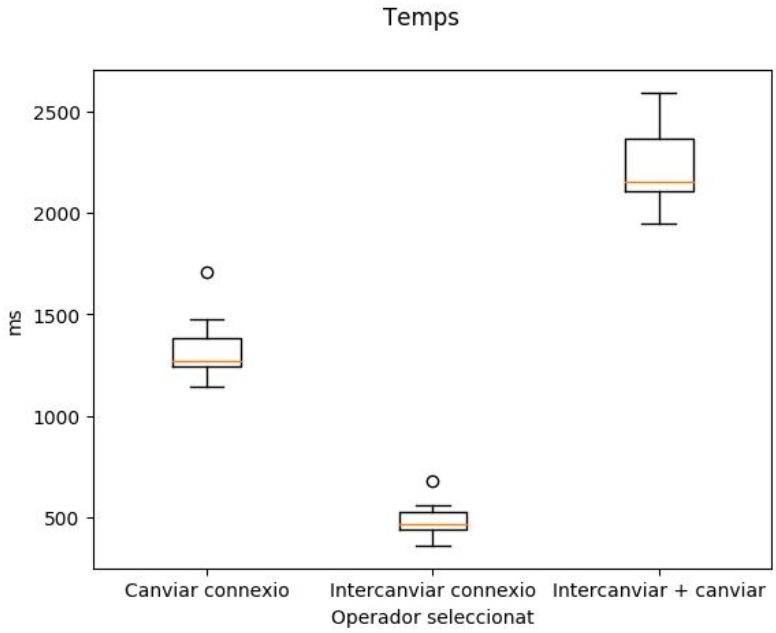
En la gráfica se observa que el operador de *intercambio de conexión* no es una buena opción en comparación con los otros dos si nos preocupa el coste final de la red. En cambio, tanto el *cambiar conexión* como el *cambiar + intercambiar* dan resultados muy similares.



Se observa que el mejor operador en cuanto a datos recibidos es el de *intercambiar conexión*, que en todos los casos recibe la totalidad de los datos emitidos. Los otros dos operadores proporcionan resultados muy similares, siendo el tercero ligeramente mejor.



Si tenemos en cuenta el valor del heurístico, el segundo operador se ve muy penalizado en relación a los otros dos, que obtienen resultados muy similares, siendo el tercero ligeramente mejor.



En la gráfica se observa que el operador que hace que la ejecución sea más rápida es el segundo. Lo sigue el primer operador, y por último, el que hace que la ejecución sea más costosa en tiempo es el tercero, con una diferencia notable, de aproximadamente el doble que el primero en media.

Los resultados obtenidos no son exactamente los que esperábamos, ya que al final el operador de intercambio de dos conexiones ha resultado ser bastante peor de lo que se esperaba en cuanto al coste de la red de la solución a la que se llega.

Una vez obtenidos estos resultados hemos reflexionado y creemos que tiene sentido que el coste sea tan elevado con el segundo operador ya que con la solución inicial greedy, ya se transmiten el 100% de los datos, y como la función heurística le da una ponderación muy alta a los datos, el algoritmo solo avanza en la dirección de disminuir el coste, pero como solo puede hacer los cambios de dos en dos, tiene limitaciones en este aspecto.

Por otro lado, esperábamos que el conjunto de operadores formado por ambos (cambiar 1 conexión + intercambiar dos conexiones) fuera notablemente mejor que ambos por separado, y no ha sido así. La calidad de las soluciones ha resultado ser muy parecida entre el operador cambiar conexión y la unión de los dos.

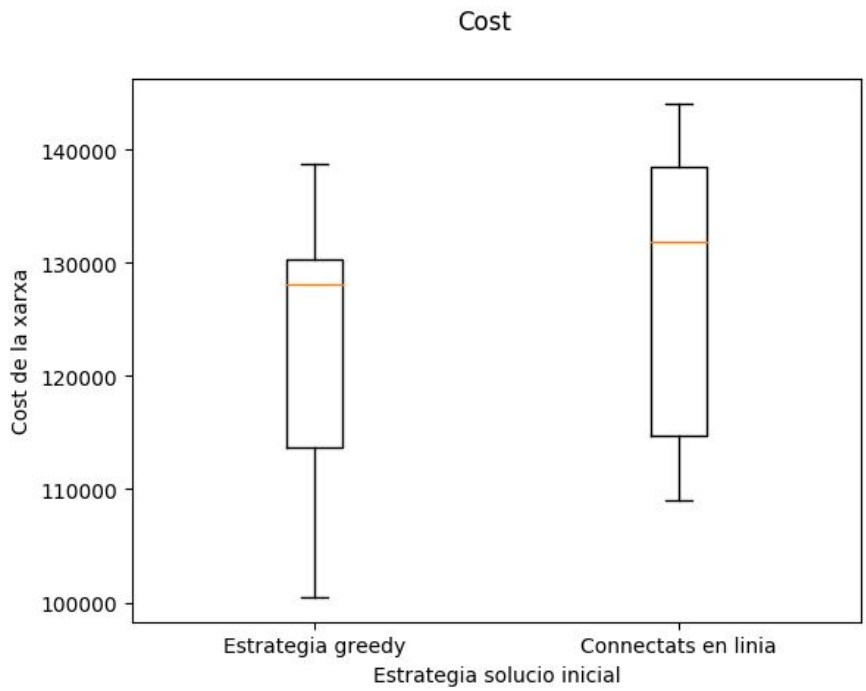
Tras analizar los resultados obtenidos con los tres operadores, podemos descartar el segundo operador dado que, aún teniendo el menor tiempo de ejecución y la mayor tasa de recogida de datos, la solución que da teniendo en cuenta el coste de la red es notablemente peor.

Respecto a los dos operadores restantes, hemos observado que las soluciones a las que llegan son bastante parecidas en cuanto a datos recogidos y coste de la red, pero hay una gran diferencia en el tiempo de ejecución de cada uno.

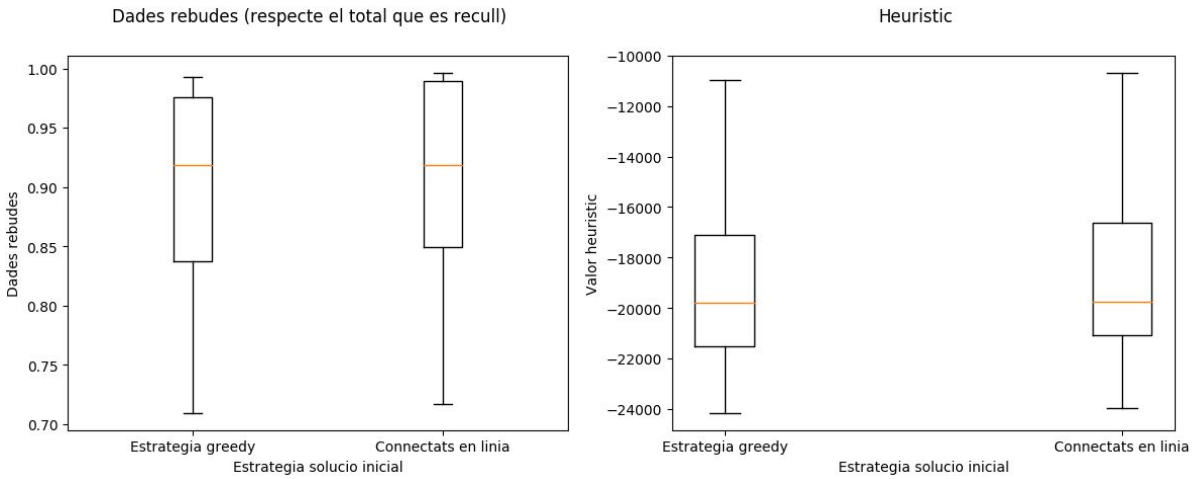
Finalmente, hemos concluido que si el tiempo no fuera un aspecto a tener en cuenta nos quedaríamos con el tercer operador, dado que es el que da mejores resultados. Pero como que el tiempo de ejecución sí nos importa, y la mejora de uno sobre el otro es mínima, decidimos quedarnos con el **primer operador**.

2. Estrategia de generación de la solución inicial

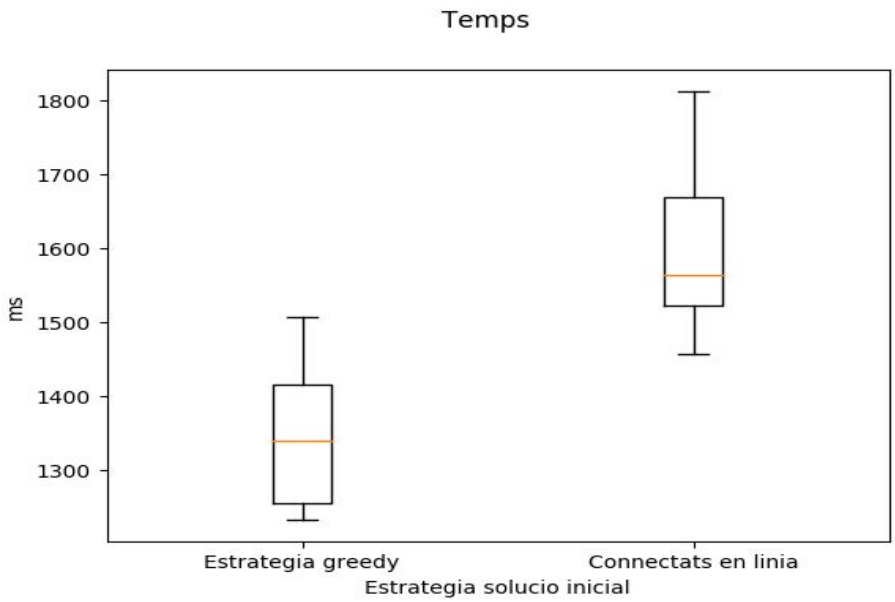
Observación	Pueden haber estrategias las cuales nos acerquen más a la solución óptima que otras.
Planteamiento	Escogemos diferentes estrategias para generar el estado inicial y analizamos las diferentes soluciones que nos dan.
Hipótesis	Nula (H0): todas las estrategias nos dan los mismos resultados. Alternativa (H1): hay estrategas que son mejores que otros.
Método	● Elegiremos 10 semillas arbitrarias para generar centros y otras 10 diferentes para generar sensores, una para cada réplica. ● Ejecutaremos 1 experimento para cada par de semillas y cada estrategia de generación de la solución inicial . ● Experimentamos con problemas de 4 centros y 100 sensores. ● Usaremos el algoritmo de Hill Climbing. ● Usaremos el operador cambiar conexión. ● Mediremos diferentes parámetros para realizar la comparación a posteriori.



Se observa que las soluciones que se obtienen con la estrategia greedy son ligeramente menos costosas. También, las soluciones que se obtienen con la estrategia de conectarlos inicialmente todos en línea son menos uniformes (la varianza es mayor en comparación con la estrategia greedy)



En las gráficas de datos recibidos y valor de heurístico se observa que ambas estrategias de generación de solución inicial obtienen resultados muy similares, de diferencias casi imperceptibles a grandes rasgos.



Se observa que las soluciones obtenidas partiendo de la solución inicial generada siguiendo una estrategia greedy tardan menos en encontrarse que las que se obtienen empezando desde una solución de estar conectados todos los sensores en línea.

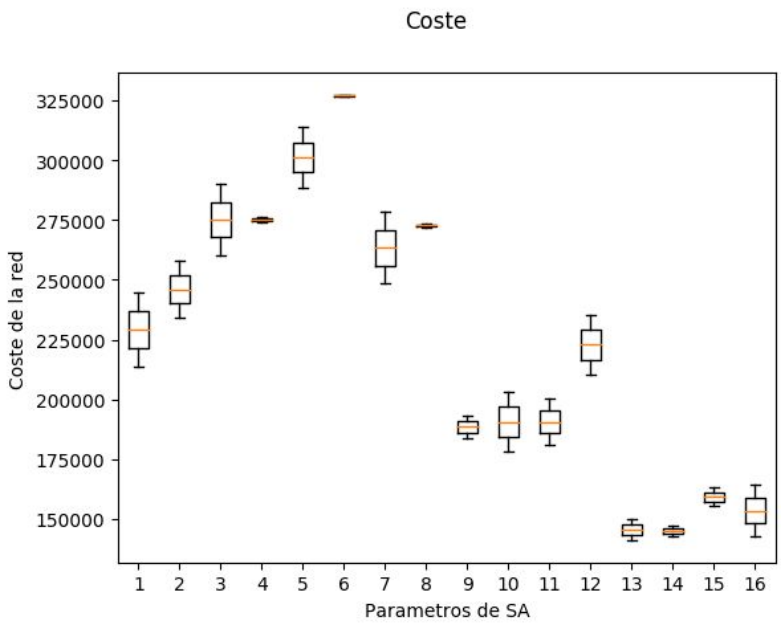
Los resultados que hemos obtenido no son exactamente los que esperábamos, ya que pensábamos que con la estrategia greedy, que es mucho más elaborada que la otra, se llegaría a soluciones mejores, pero no ha sido así. Lo que hemos obtenido es que se llega a soluciones muy similares al final, con la diferencia de que partiendo de la estrategia greedy se expanden una cantidad notablemente menor de nodos para llegar.

Después de analizar los resultados obtenidos en el experimento, hemos decidido que la estrategia de generación de solución inicial con la que nos quedaremos para los siguientes experimentos es la **greedy**, ya que, aunque la calidad de las soluciones que se obtienen con ambas estrategias es prácticamente idéntica, la estrategia greedy es la que hace que las ejecuciones tarden menos tiempo.

3. Parámetros para el Simulated Annealing

Observación	Pueden haber diferentes conjuntos de valores para los parámetros que nos den mejores soluciones que el resto.
Planteamiento	Escogemos diferentes grupos de parámetros y analizamos las diferentes soluciones que nos dan.
Hipótesis	Nula (H0): todos los conjuntos de parámetros dan los mismos resultados. Alternativa (H1): hay conjuntos de parámetros mejores que otros.
Método	● Elegiremos 10 semillas arbitrarias para generar centros y otras 10 diferentes para generar sensores, una para cada réplica. ● Ejecutaremos 1 experimento para cada par de semillas y cada conjunto de parámetros. ● Experimentamos con problemas de 4 centros y 100 sensores. ● Usaremos el algoritmo de Simulated Annealing. ● Mediremos diferentes aspectos de la solución para realizar la comparación a posteriori.

En esta explicación hablaré de las diferencias entre las muestras y de cómo impactan en los resultados. Podeis encontrar la lista de las muestras separadas por grupos y colores en los apéndices (3.2). En el apéndice (3.3) podéis encontrar dos gráficas que muestran cómo hemos obtenido los diferentes parámetros a partir de una K y una Lambda arbitrarias.



Los resultados mostrados en la primera gráfica se pueden agrupar en cuatro grupos de cuatro y asignarles un color distintivo a cada uno (verde, azul, rojo y naranja). Cada grupo a su vez se puede dividir en dos subgrupos, dependiendo del número de iteraciones. En los grupos verde, azul y naranja la primera pareja de muestras de cada uno de ellos tiene un número de iteraciones mayor que su segunda pareja, y todos ellos muestran peores costes con menor número de iteraciones. En el particular caso del grupo rojo (muestras 5, 6, 7 y 8) es

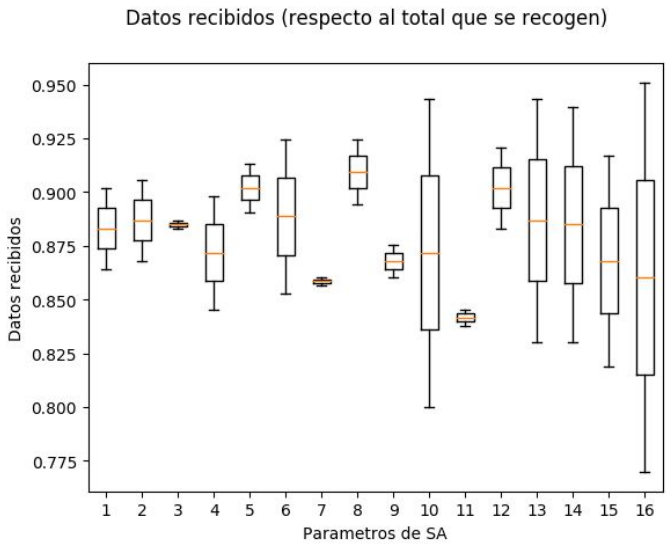
al revés, la primera pareja tiene un número de iteraciones menor que la segunda, y la gráfica mantiene la teoría de que a mayor número de iteraciones mejor es el coste obtenido por la solución.
En la primera gráfica podemos comprobar que el grupo naranja es el más eficiente en lo que el coste incumbe. Comparándolo con el grupo verde podemos ver que el valor de la K es el doble en el grupo naranja, esto hace que el coste de las soluciones encontradas con el grupo de muestras naranja sea casi la mitad que aquellas encontradas con el grupo verde.

Siguiendo con las comparaciones, ahora con el grupo verde con el rojo podemos ver que disminuir la mitad el valor de la Lambda no ha afectado de la misma forma que en el caso anterior, en este el coste encontrado con el grupo de muestras rojo es un poco más de dos tercios que aquellas encontradas con el grupo verde.

Por lo tanto podemos deducir que, aumentar la K siempre nos aportará unos mejores resultados que disminuir la Lambda en la misma proporción.

Así pues si aumentamos la K y disminuimos la Lambda el coste obtenido tendría que ser mucho mejor y eso es lo que vemos si comparamos el grupo azul con el rojo.

En el segundo gráfico vemos un impacto diferente. Esta vez si comparamos los datos recibidos entre el grupo verde con el naranja podemos ver que en ciertas ocasiones el grupo naranja ha tenido un valor de datos recibidos mayor que el verde, pero la mediana y la desviación que nos ofrece el verde es más estable y en general obtiene entre un 5 y un 10% más de datos que el naranja. Así pues aumentar el doble la K afecta enormemente a la estabilidad de los datos recibidos de las respuestas.

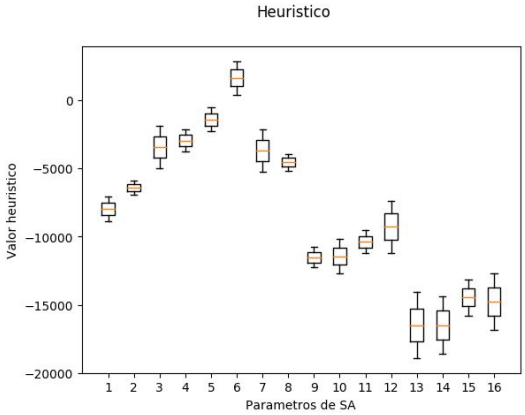


Comparando el grupo verde con el rojo podemos ver rápidamente que los resultados son peores, si es cierto que en general no se ve el rango de varianza que veía con el naranja, pero el valor de datos recibidos es pero. Solo hay un caso (12) que sea mejor y muy estable, por lo que desestimaremos su importancia.

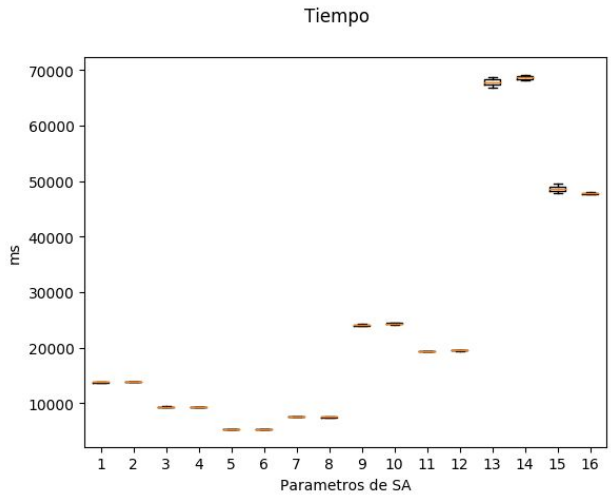
En la comparación del grupo azul con el rojo podemos establecer una relación clara. Aumentar la K lleva más varianza y reducir la Lambda lleva a mejores soluciones medias. La cuestión es que no siempre que se aumenta la K y se disminuye la Lambda se encuentran mejores soluciones.

Respecto al heurístico podemos observar unas tendencias parecidas a la gráfica del coste, esto es debido a que nuestro heurístico da más peso a los datos, y una vez han sido optimizados el rango de mejora del coste es muy pequeño.

En general podemos ver que para el heurístico las mismas resoluciones que nos hemos encontrado con el primer gráfico se mantienen. Si aumentamos la K aumenta la calidad del heurístico, eso quiere decir que disminuye su valor, y en el caso de aumentar la Lambda también vemos una mejora en el heurístico, pero se mantiene el hecho de que la proporción de mejora del heurístico entre el aumento de K y la disminución de Lambda es mucho mayor por la K.



Para finalizar si observamos esta gráfica de tiempo podemos ver que los dos primeros grupos (verde y azul) tienen un coste bastante parecido, al rededor de 10 segundos, pero estos dos primeros grupos los desechamos porque ya hemos visto que el coste y los datos recibidos de las soluciones son muy pobres. Así pues nos quedamos con los dos últimos.



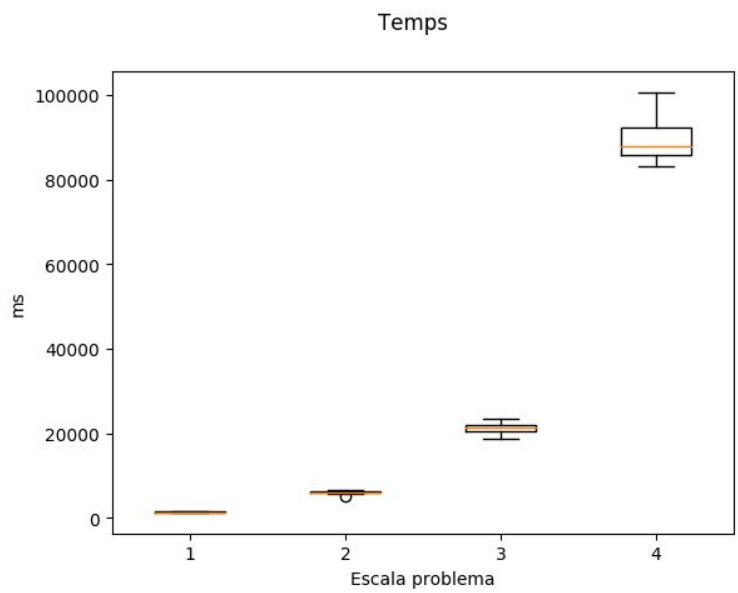
De estos dos que nos quedan podemos discutir que el grupo naranja es notablemente mejor que el rojo, el coste nos avala. Pero el gran problema es la inestabilidad que hay en en los datos recibidos. El naranja tiene una varianza en los datos recibidos que no aporta confianza y seguridad en su elección, así pues nos quedamos con el tercer grupo, y dentro de este grupo que lo forman las muestras 9, 10, 11 y 12.

Con este reducido conjunto escogeremos la muestra 9. Tiene un tiempo un poco superior a sus vecinos 11 y 12, tiene un valor de datos recibidos decente, aunque bastante inferior al 12, pero tiene un coste mucho mejor que el 12. El heurístico lo marca como el correcto así que podemos afirmar que escoger la muestra 9 dentro de las que nos ofrece el grupo rojo es una buena elección.

4. Tiempo de ejecución en función del tamaño del problema

Observación	Puede que al aumentar el tamaño del problema aumente también el tiempo de ejecución.
Planteamiento	Incrementamos iterativamente el tamaño del problema de 2 en 2 centros siguiendo la proporción 4:100.
Hipótesis	Nula (H0): El tiempo de ejecución no depende del tamaño del problema. Alternativa (H1): El tamaño del problema afecta al tiempo de ejecución.
Método	● Elegiremos 10 semillas arbitrarias para generar centros y otras 10 diferentes para generar sensores, una para cada réplica. ● Ejecutaremos 4 experimentos para cada par de semillas, aumentando en cada iteración el tamaño del problema. ● Empezamos con un tamaño de problema de 4 centros y 100 sensores y terminamos con uno de 10 centros y 250 sensores. ● Usaremos el algoritmo de Hill Climbing. ● Mediremos diferentes aspectos de la solución (valor del heurístico, coste de la red y tiempo de ejecución para realizar la comparación a posteriori.

Como se puede apreciar en la gráfica, a medida que vamos incrementando el tamaño del problema manteniendo la escala 4:100, el tiempo de ejecución se incrementa de manera exponencial. Esto no coincide con lo que esperábamos obtener, ya que pensábamos que manteniendo la escala, el tiempo aumentaría de forma lineal. Pero después de haber obtenido estos resultados, hemos pensado que tiene bastante lógica que esto suceda, ya que, como utilizamos el operador de cambiar conexión, al añadir nuevos sensores y centros, los posibles destinos de cada sensor se incrementan también, y como en cada sucesor se tienen que comprobar los posibles destinos de cada sensor, tiene sentido que el tiempo total aumente exponencialmente

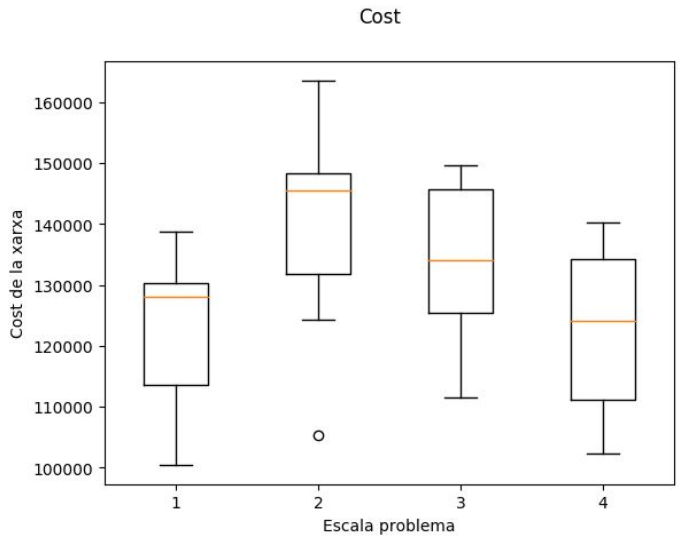


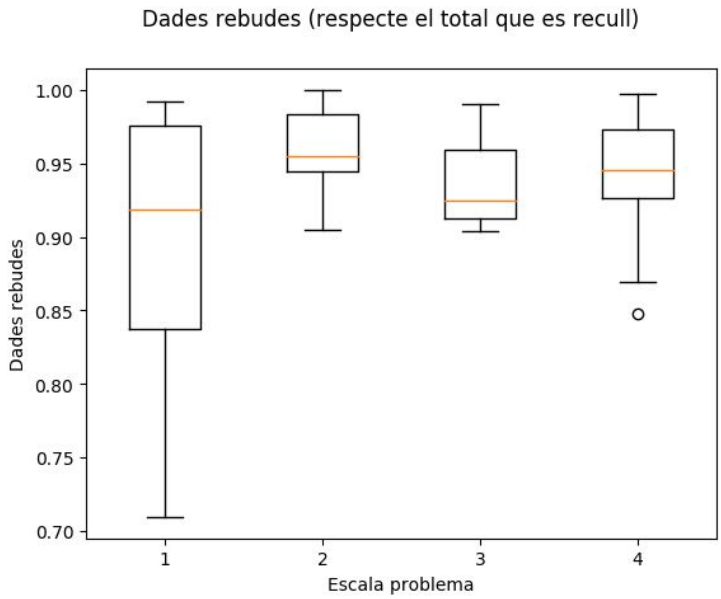
Aunque el experimento no lo pida explícitamente, también hemos recogido los resultados de cómo varía el coste de la red y el porcentaje de datos recibidos cuando variamos la escala.
Lo que hemos obtenido es que:

- mientras que el coste de la red no sufre grandes alteraciones con los cambios de escala (lo que tiene lógica ya que se añaden centros a la vez que sensores
- el porcentaje de datos recibidos es más estable (tiene menos desviación) en problemas en los que se ha aumentado la escala (esto lo podemos saber observando que el tamaño de la caja que se corresponde con escala 1 es mayor que el de las otras).

Por último, si hablamos en valores medios, en todos los tamaños de problema llegan a soluciones en las que se obtiene un porcentaje de datos de entre el 90 y el 100%.

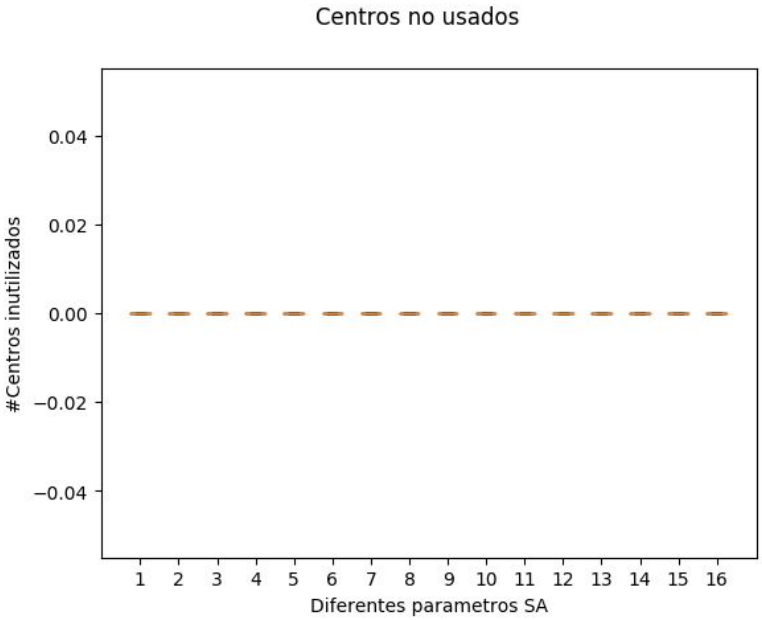
Lo comentado anteriormente se puede apreciar en las siguientes gráficas:



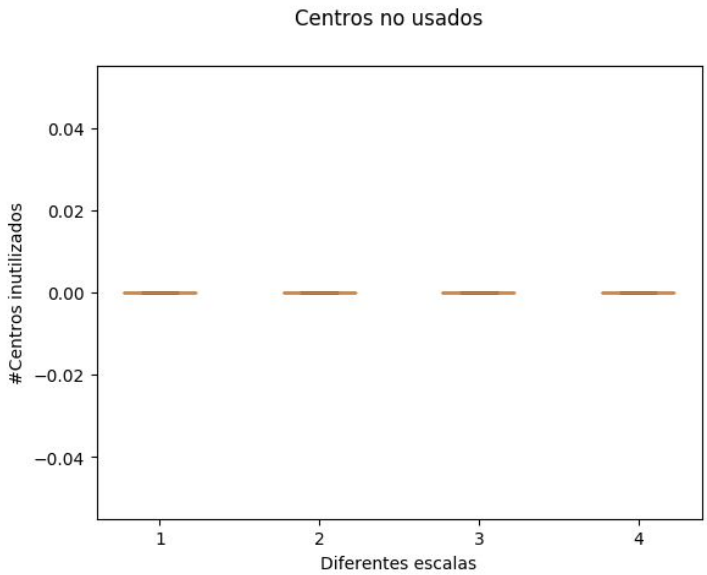


5. Inutilización de centros de datos

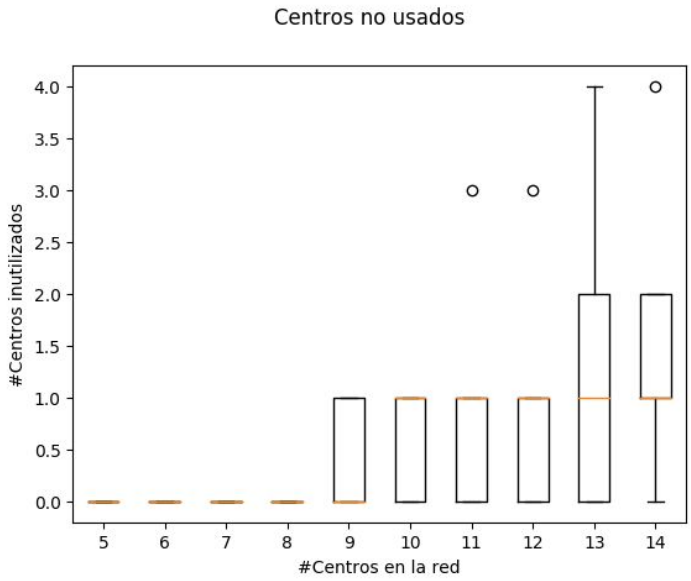
Observación	Puede ser que no todos los centros de datos estén en uso dependiendo de la distribución de los elementos del problema.
Planteamiento	Estudiaremos si hay centros que no se usan, usando Hill Climbing y Simulated Annealing.
Hipótesis	Nula (H0): Se utilizan todos los centros de datos dispuestos. Alternativa (H1): Hay centros de datos que finalmente no se utilizan.
Método	● Ejecutaremos 3 experimentos: 1 que utilizará Hill Climbing, 1 que usará Simulated Annealing con diferentes parámetros, y un tercero que cambiará la escala del problema usando Hill Climbing. ● Para cada experimento elegiremos 10 semillas arbitrarias para generar centros y otras 10 diferentes para generar sensores, una para cada réplica. ● Experimento con Hill Climbing: ○ Se ejecutará el algoritmo con 10 pares de semillas diferentes que proporcionarán diferentes distribuciones de centros y sensores. ● Experimento con Simulated Annealing: ○ Se ejecutará el algoritmo con 10 pares de semillas en cada réplica. ○ Para cada par de semillas, se probarán 16 diferentes configuraciones de parámetros de k, lambda, steps e iteraciones. ● Experimento variando la escala: ○ Se ejecutará el algoritmo con 10 pares de semillas en cada réplica. ○ Se usará Hill Climbing ○ En cada réplica se escalará el problema de 2 en 2 centros manteniendo la proporción 4:100, partiendo de 4 y 100 hasta 10 y 250 ● Contaremos el número de centros que no son utilizados en cada instancia del problema.



Usando Simulated Annealing también hemos visto que el número de centros inutilizados con diferentes semillas y diferentes configuraciones de k,lambda,iteraciones y steps es siempre 0. En el apartado 5.2 del apéndice se pueden encontrar dichas configuraciones. Al igual que en el experimento anterior, hemos utilizado una red con 4 centros de datos y 100 sensores.



En esta gráfica se puede observar que usando el algoritmo de Hill Climbing con diferentes semillas, diferentes números de centros de datos y sensores tampoco hay ningún centro inutilizado. Eso puede ser debido a que siempre estamos usando la misma proporción de centros y sensores. Para ver si es debido a esto hemos ejecutado el mismo experimento fijando el número de sensores a 100 y hemos ido aumentando el número de centros de 10 en 10 de 5 a 55.



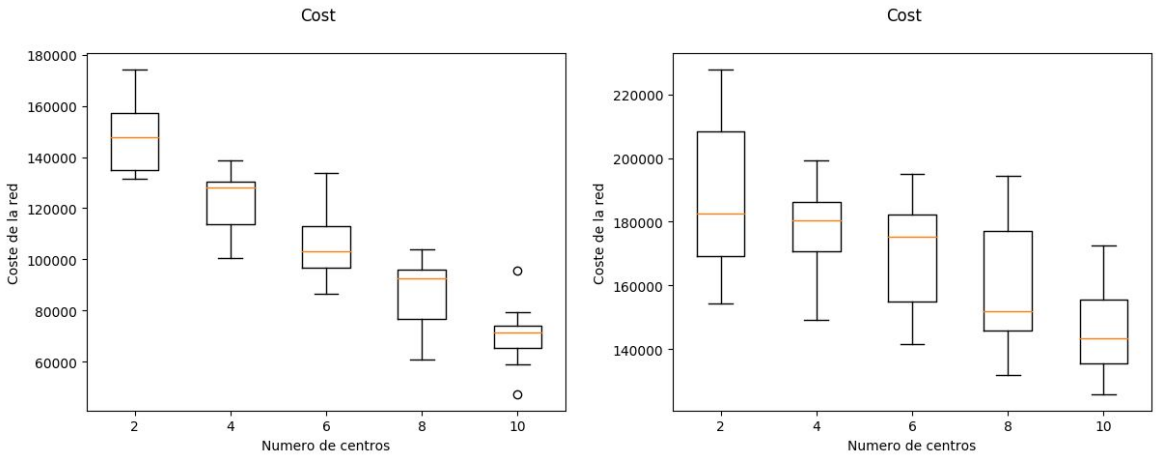
Para ver si el número de centros inutilizados varía en función del número de centros, hemos fijado del número de sensores a 100 y hemos ido incrementando el número de centros. En la gráfica se puede observar que a partir de los 10 centros ya hay alguno no utilizado. Por lo tanto, a partir de la proporción **1:10 empieza a haber centros inutilizados.**

(El rango de número de centros se ha decidido después de realizar el mismo experimento con un rango mayor como se puede ver en el apéndice apartado 5.5).

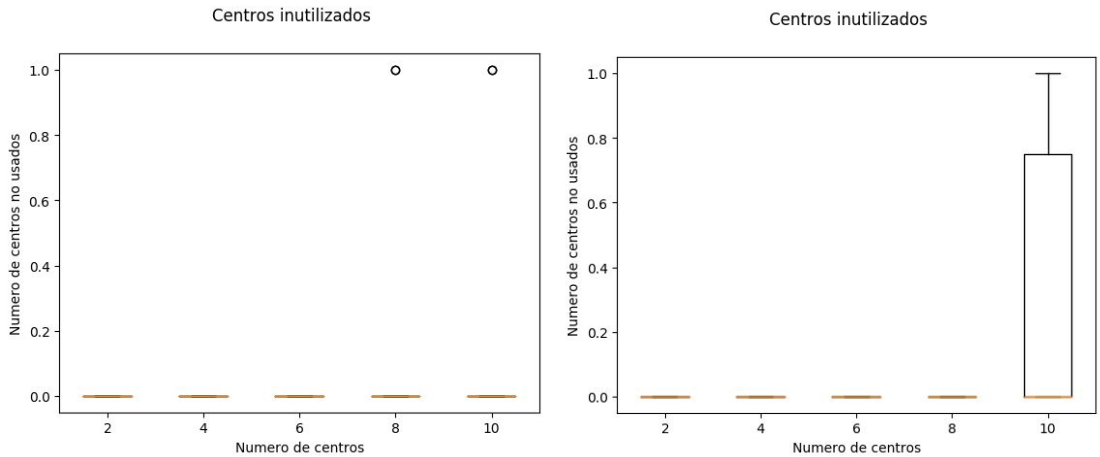
Podemos concluir que el número de centros no utilizados depende del número de centros de la red y del número de sensores. Se ha visto que **a partir de la proporción 1:10** ya hay algún centro inutilizado. También que no depende del algoritmo que usemos ya sea *Hill Climbing* o *Simulated Annealing.*

6. Efecto de añadir más centros

Observación	Es posible que al añadir más centros de datos, cambien el coste de la red, los centros usados y el tiempo de ejecución.
Planteamiento	Estudiaremos si hay cambios en el coste de la red, en los centros usados y en el tiempo de ejecución a medida que se van añadiendo más centros a la red.
Hipótesis	Nula (H0): Ni el tiempo de ejecución, ni el coste de la red, ni el número de centros utilizados cambian con la adición de más centros. Alternativa (H1): El tiempo de ejecución y/o el coste de la red y/o el número de centros utilizados cambian con la adición de más centros.
Método	● Ejecutaremos 2 experimentos: 1 que utilizará Hill Climbing, 1 que usará Simulated Annealing con diferentes parámetros. ● Para cada experimento elegiremos 10 semillas arbitrarias para generar centros y otras 10 diferentes para generar sensores, una para cada réplica. ● Experimento con Hill Climbing: ○ Se ejecutará el algoritmo con 10 pares de semillas diferentes que proporcionarán diferentes distribuciones de centros y sensores. ○ Para cada distribución, se empezará con 2 centros y se irán incrementando iterativamente hasta llegar hasta 10. ● Experimento con Simulated Annealing: ○ Se ejecutará el algoritmo con 10 pares de semillas diferentes que proporcionarán diferentes distribuciones de centros y sensores. ○ Para cada distribución, se empezará con 2 centros y se irán incrementando iterativamente hasta llegar hasta 10. ○ Los parámetros serán los que hemos escogido después de hacer el experimento 3. ● Mediremos coste de la red, tiempo de ejecución y el número de centros que no son utilizados en cada instancia del problema.



Como podemos ver en el primer par de gráficos el coste de la red disminuye de forma lineal a medida que vamos aumentando el número de centros, siendo 10 centros el coste más bajo y 2 centros el coste más elevado, como es lógico. Podemos también apreciar que aunque el coste de la primera gráfica (asociado al algoritmo Hill Climbing) es mucho mejor que el de la segunda gráfica (asociado al algoritmo Simulated Annealing), todos dos mantienen dicha relación.

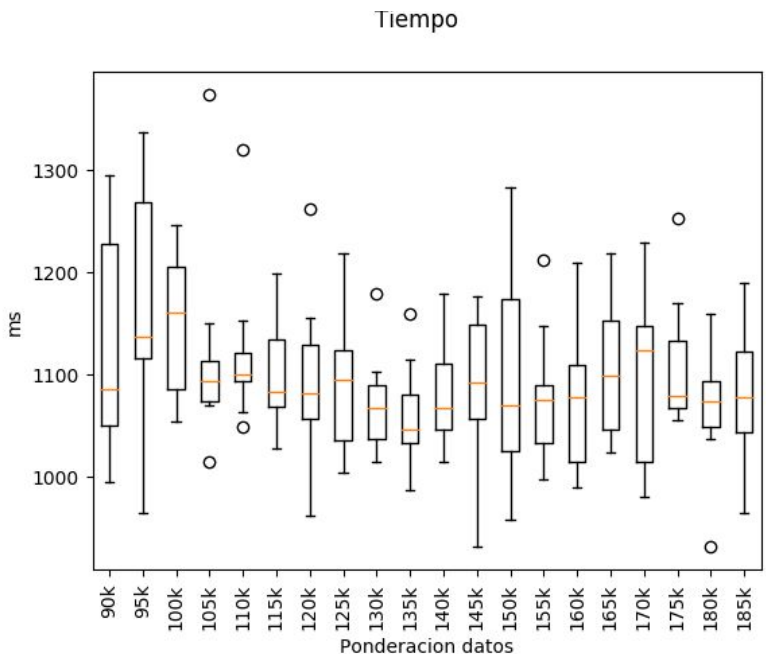


También podemos ver que el número de centros sin utilizar es parecido, como en general las soluciones encontradas por el algoritmo Simulated Annealing son peores que las encontrados con el algoritmo Hill Climbing podemos decir que no hay centros sin utilizar hasta llegar a 8 centros.

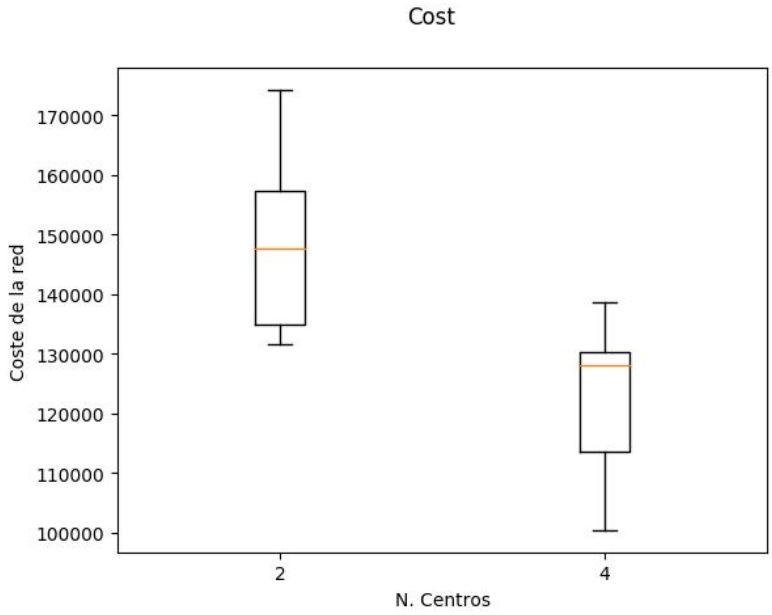
Con este experimento hemos podido comprobar que el coste de la red disminuye a la par que sube el número de centros, cosa que ya pensábamos. También podemos ver que cuando el número de centros es muy elevado empiezan a aparecer centros sin usar, ergo que no tienen ningún sensor conectados a ellos.

7. Efectos de cambiar la ponderación de datos en el heurístico

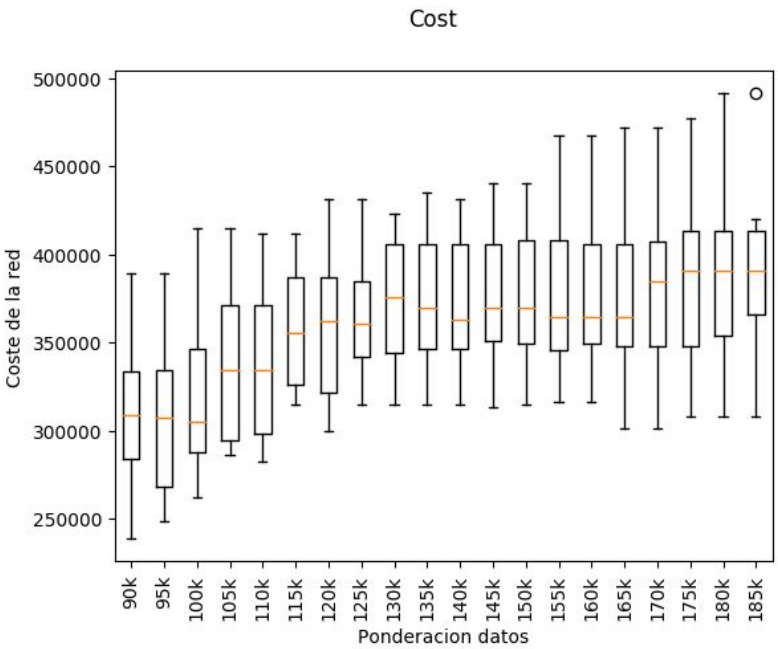
Observación	Es posible que cambiando la ponderación de los datos recogidos en la función heurística cambie el tiempo de ejecución y la calidad de las soluciones obtenidas.
Planteamiento	Incrementamos iterativamente la ponderación que se le da al factor datos en la función heurística.
Hipótesis	Nula (H0): Cambiar la ponderación de los datos no tiene efecto en las soluciones obtenidas ni en el tiempo que se tarda en obtenerlas. Alternativa (H1): Cambiar la ponderación de los datos afecta las soluciones obtenidas y/o al tiempo que se tarda en obtenerlas.
Método	● Elegiremos 10 semillas arbitrarias para generar centros y otras 10 diferentes para generar sensores, una para cada réplica. ● Ejecutaremos 10 experimentos para cada par de semillas, aumentando en cada iteración el tamaño del problema. ● Empezamos con un valor inferior a la ponderación usada en nuestra función heurística y vamos incrementando esta ponderación. ● Usaremos el algoritmo de Hill Climbing.



En esta gráfica se puede ver que el tiempo de hallar una solución se reduce hasta un cierto punto en el que se estanca cuando la ponderación de los datos en el heurístico pasa los 105k. En el **apartado 7.2 del apéndice** se puede ver la gráfica del tiempo con valores superiores e inferiores. El hecho de que el tiempo de ejecución se reduzca es debido a que como más altas son las ponderaciones más pesan en la función heurística y hay más diferencias entre los estados por lo que la función heurística encontrará estados mejores más rápidamente.



Si usamos la misma ponderación para los datos que en los experimentos anteriores (35k), se puede observar como en una red con 4 centros y 100 sensores el coste medio de la red es de aproximadamente 129k. En cambio en una red con 2 centros el coste es de 147k. Por lo tanto, al pasar de 4 centros a 2, el coste de la red ha aumentado en una proporción de 147000/129000 = 1.13. *Como extra, en el apartado 7.3 del apéndice se ha realizado el mismo experimento mesurando los datos.*



Al aumentar la ponderación de los datos en la función heurística también aumenta la ponderación de la red. Se ve que a partir de los 170k el coste de la red deja de aumentar. Eso es debido a que el sistema no puede conseguir que lleguen más datos aunque le demos mucha más prioridad. Es por eso que el coste de la red deja de aumentar. *En el apartado 7.4 del apéndice se puede observar la misma gráfica en diferentes rangos.*

7. Conclusiones

Después de haber estudiado los resultados obtenidos del conjunto de experimentos realizados hemos concluido lo siguiente:

- En general el algoritmo de Hill Climbing es mejor que el de Simulated Annealing para enfocar la resolución de este problema.
- La diferencia en las soluciones obtenidas utilizando el operador cambio de conexión y el swap plus ha sido menor de la que esperábamos.
- La diferencia en las soluciones obtenidas partiendo de las diferentes estrategias de generación de la solución inicial ha sido menor de la que esperábamos.
- El impacto en el coste de la red de añadir más centros al mismo problema inicial ha sido mucho mayor del que esperábamos.
- Hemos podido ver que al aumentar el número de centros al mismo problema inicial aparecían centros de datos los cuales no tenían ningún sensor conectado a ellos.
- Hemos podido ver empíricamente cómo afectan a la calidad de las soluciones los cambios en los parámetros K y Lambda del algoritmo de Simulated Annealing.
- Manteniendo la proporción 4:100 centros a sensores, hemos podido ver que aumentar el tamaño del problema hace que el tiempo de ejecución de cualquier algoritmo crezca exponencialmente.

8. Competencia transversal

El tema que hemos escogido para la competencia transversal es el *autopilot* que incorporan algunos vehículos de la compañía Tesla.

Hasta el momento hemos buscado información sobre el tema.

Las referencias que hemos encontrado son las siguientes:

Link	Fecha acceso	Partes relevantes
https://www.tesla.com/presskit/autopilot?redirect=no	15/02/2017	-Evolución del autopilot -Que es capaz de hacer -Hardware necesario
https://www.tesla.com/autopilot	15/02/2017	-Explicación más a fondo de sus funcionalidades
http://fortune.com/2015/10/16/how-tesla-autopilot-learns/	01/03/2017	-Como es (en aspectos muy generales) el *machine learning* del *autopilot*.

A la hora de buscar información nos hemos encontrado con que la mayoría de fuentes no dan información detallada sobre el tema, simplemente dicen lo mismo que en la web de tesla con otras palabras.

9. Apéndice

1. Script experimento 1

```
#imports
import subprocess
import csv
from matplotlib import pyplot as plt
import numpy as np

#Configuration of the experiment
seed_c = 1234
seed_s = 4321

num_c = 4
num_s = 100

operant_id = 3
initial_solution_id = 1

program = []

program.append("java")
program.append("-jar")
program.append("CodiPractica/out/artifacts/CodiPractica_jar/CodiPractica.jar")

program.append(str(num_c))
program.append(str(num_s))
program.append(str(seed_c))
program.append(str(seed_s))
program.append(str(operant_id))
program.append(str(initial_solution_id))

seeds_s = [1313, 1122, 2233, 3344, 4455, 5566, 6677, 7788, 8899, 9900]
seeds_c = [1100, 2211, 3322, 4433, 5544, 6655, 7766, 8877, 9988, 3141]

listaHeurist = [[],[],[]]
listaCoste = [[],[],[]]
listaDatos = [[],[],[]]

listaMilis = [[],[],[]]
listaExpanded= [[],[],[]]

REPS = 10

for rep in range(REPS):
    program[5] = str(rep*10)
    program[6] = str(rep*20)
    for op in range(1,4):
        program[-2] = str(op)
```

```
        output = subprocess.Popen(program, stdout=subprocess.PIPE).communicate()[0]
        reader = csv.reader(output.splitlines())
        listaCsv = list(reader)
        listaHeurist[op-1].append(eval(listaCsv[-3][2]))
        listaCoste[op-1].append(eval(listaCsv[-3][1]))
        listaDatos[op-1].append(eval(listaCsv[-3][0]))
        listaMilis[op-1].append(eval(listaCsv[-2][0]))
        listaExpanded[op-1].append(eval(listaCsv[-1][0]))

plt.boxplot(listaHeurist)
plt.suptitle('Heuristic')
plt.autoscale()
plt.xticks([1,2,3],['Canviar connexio','Intercanviar connexio','Intercanviar + canviar'])
plt.xlabel('Operador seleccionat')
plt.ylabel('Valor heuristic')
plt.savefig('./experiment1/heuristic.png',bbox_inches='tight')
plt.clf()
plt.boxplot(listaCoste)
plt.suptitle('Cost')
plt.xticks([1,2,3],['Canviar connexio','Intercanviar connexio','Intercanviar + canviar'])
plt.xlabel('Operador seleccionat')
plt.ylabel('Cost de la xarxa')
plt.savefig('./experiment1/coste.png',bbox_inches='tight')
plt.clf()
plt.boxplot(listaDatos)
plt.suptitle('Dades rebudes (respecte el total que es recull)')
plt.xticks([1,2,3],['Canviar connexio','Intercanviar connexio','Intercanviar + canviar'])
plt.xlabel('Operador seleccionat')
plt.ylabel('Dades rebudes')
plt.savefig('./experiment1/datos.png',bbox_inches='tight')
plt.clf()
plt.boxplot(listaMilis)
plt.suptitle('Temps')
plt.xticks([1,2,3],['Canviar connexio','Intercanviar connexio','Intercanviar + canviar'])
plt.xlabel('Operador seleccionat')
plt.ylabel('ms')
plt.savefig('./experiment1/tiempo.png',bbox_inches='tight')
plt.clf()
```

2. Script experimento 2

```
#imports
import subprocess
import csv
from matplotlib import pyplot as plt
import numpy as np

#Configuration of the experiment
seed_c = 1234
seed_s = 4321

num_c = 4
num_s = 100

operant_id = 1
initial_solution_id = 1

program = []

program.append("java")
program.append("-jar")
program.append("CodiPractica/out/artifacts/CodiPractica_jar/CodiPractica.jar")

program.append(str(num_c))
program.append(str(num_s))
program.append(str(seed_c))
program.append(str(seed_s))
program.append(str(operant_id))
program.append(str(initial_solution_id))

seeds_s = [1313, 1122, 2233, 3344, 4455, 5566, 6677, 7788, 8899, 9900]
seeds_c = [1100, 2211, 3322, 4433, 5544, 6655, 7766, 8877, 9988, 3141]

listaHeurist = [[],[]]
listaCoste = [[],[]]
listaDatos = [[],[]]

listaMilis = [[],[]]
listaExpanded= [[],[]]

for rep in range(10):
    program[5] = str(seeds_c[rep])
    program[6] = str(seeds_s[rep])
    for op in range(1,3):
        program[-1] = str(op)
        output = subprocess.Popen(program, stdout=subprocess.PIPE).communicate()[0]
        reader = csv.reader(output.splitlines())
        listaCsv = list(reader)
        listaHeurist[op-1].append(eval(listaCsv[-3][2]))
        listaCoste[op-1].append(eval(listaCsv[-3][1]))
        listaDatos[op-1].append(eval(listaCsv[-3][0]))
        listaMilis[op-1].append(eval(listaCsv[-2][0]))
```

```
        listaExpanded[op-1].append(eval(listaCsv[-1][0]))

plt.boxplot(listaHeurist)
plt.suptitle('Heuristic')
plt.autoscale()
plt.xticks([1,2],['Estrategia greedy','Connectats en linia'])
plt.xlabel('Estrategia solucio inicial')
plt.ylabel('Valor heuristic')
plt.savefig('./experiment2/heuristic.png',bbox_inches='tight')
plt.clf()
plt.boxplot(listaCoste)
plt.suptitle('Cost')
plt.xticks([1,2],['Estrategia greedy','Connectats en linia'])
plt.xlabel('Estrategia solucio inicial')
plt.ylabel('Cost de la xarxa')
plt.savefig('./experiment2/coste.png',bbox_inches='tight')
plt.clf()
plt.boxplot(listaDatos)
plt.suptitle('Dades rebudes (respecte el total que es recull)')
plt.xticks([1,2],['Estrategia greedy','Connectats en linia'])
plt.xlabel('Estrategia solucio inicial')
plt.ylabel('Dades rebudes')
plt.savefig('./experiment2/datos.png',bbox_inches='tight')
plt.clf()
plt.boxplot(listaMilis)
plt.suptitle('Temps')
plt.xticks([1,2],['Estrategia greedy','Connectats en linia'])
plt.xlabel('Estrategia solucio inicial')
plt.ylabel('ms')
plt.savefig('./experiment2/tiempo.png',bbox_inches='tight')
plt.clf()
```

3. Experimento Simulated Annealing

3.1. Script experimento 3

```
#imports
import subprocess
import csv
from matplotlib import pyplot as plt
import numpy as np

#Configuration of the experiment
seed_c = 1234
seed_s = 4321

num_c = 4
num_s = 100

operant_id = 1
initial_solution_id = 1

#Search Alianning parameters
k = 10
l = 0.01
i = 3000
s = 30

program = []

program.append("java")
program.append("-jar")
program.append("CodiPractica/out/artifacts/CodiPractica_jar/CodiPractica.jar")

program.append(str(num_c))
program.append(str(num_s))
program.append(str(seed_c))
program.append(str(seed_s))
program.append(str(operant_id))
program.append(str(initial_solution_id))

program.append(str(k))
program.append(str(l))
program.append(str(i))
program.append(str(s))

#numc nums seedc seeds opid genid

seeds_s = [1313, 1122, 2233, 3344, 4455, 5566, 6677, 7788, 8899, 9900]
seeds_c = [1100, 2211, 3322, 4433, 5544, 6655, 7766, 8877, 9988, 3141]

k = [10,10,10,10,5,5,5,5,10,10,10,10,20,20,20,20]
lamb = [0.01,0.01,0.01,0.01,0.1,0.1,0.1,0.1,0.005,0.005,0.005,0.005,0.01,0.01,0.01,0.01]
iters = [2244,2244,1500,1500,830,830,1200,1200,4000,4000,3200,3200,11460,11460,8000,8000]
```

```
steps = [4,561,15,100,10,83,12,100,4,1000,32,100,10,1146,8,1000]

listaHeurist = [[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]
listaCoste = [[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]
listaDatos = [[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]

listaMilis = [[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]
listaExpanded= [[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]

for rep in range(2):
    print rep
    program[5] = str(seeds_c[rep])
    program[6] = str(seeds_s[rep])
    for ite in range(16):
        print ite
        program[-1] = str(steps[ite])
        program[-2] = str(iters[ite])
        program[-3] = str(lamb[ite])
        program[-4] = str(k[ite])
        output = subprocess.Popen(program, stdout=subprocess.PIPE).communicate()[0]
        reader = csv.reader(output.splitlines())
        listaCsv = list(reader)
        listaHeurist[ite].append(eval(listaCsv[-3][2]))
        listaCoste[ite].append(eval(listaCsv[-3][1]))
        listaDatos[ite].append(eval(listaCsv[-3][0]))
        listaMilis[ite].append(eval(listaCsv[-2][0]))
        listaExpanded[ite].append(eval(listaCsv[-1][0]))

plt.boxplot(listaHeurist)
plt.suptitle('Heuristico')
plt.autoscale()
plt.xlabel('Parametros de SA')
plt.ylabel('Valor heuristico')
plt.savefig('./experiment3/heuristic.png',bbox_inches='tight')
plt.clf()
plt.boxplot(listaCoste)
plt.suptitle('Coste')
plt.xlabel('Parametros de SA')
plt.ylabel('Coste de la red')
plt.savefig('./experiment3/coste.png',bbox_inches='tight')
plt.clf()
plt.boxplot(listaDatos)
plt.suptitle('Datos recibidos (respecto al total que se recogen)')
plt.xlabel('Parametros de SA')
plt.ylabel('Datos recibidos')
plt.savefig('./experiment3/datos.png',bbox_inches='tight')
plt.clf()
plt.boxplot(listaMilis)
plt.suptitle('Tiempo')
plt.xlabel('Parametros de SA')
plt.ylabel('ms')
plt.savefig('./experiment3/tiempo.png',bbox_inches='tight')
plt.clf()
```

3.2. Tabla de muestras

#MUESTRA	K	LAMBDA	#ITER	#STEPS
1	10	0.01	2244	4
2	10	0.01	2244	561
3	10	0.01	1500	15
4	10	0.01	1500	100
5	5	0.1	830	10
6	5	0.1	830	83
7	5	0.1	1200	12
8	5	0.1	1200	100
9	20	0.005	4000	4
10	20	0.005	4000	1000
11	20	0.005	3200	32
12	20	0.005	3200	100
13	10	0.01	11460	10
14	10	0.01	11460	1146
15	10	0.01	8000	8
16	10	0.01	8000	1000

Las muestras están separadas por grupos de colores teniendo en cuenta los valores de K y Lambda.

3.3. Obtención de los parámetros

Input:

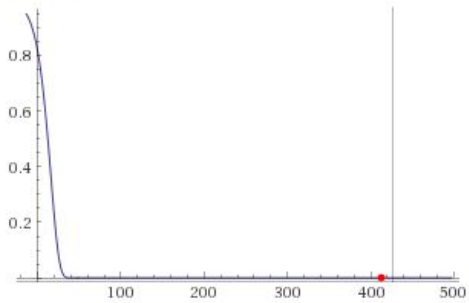
$$\exp\left(-\frac{1}{5\exp(-0.1\,x)}\right)=0$$

Open code

Result:

$$e^{-\frac{e^{0.1\,x}}{5}}=0$$

Root plot:



Input:

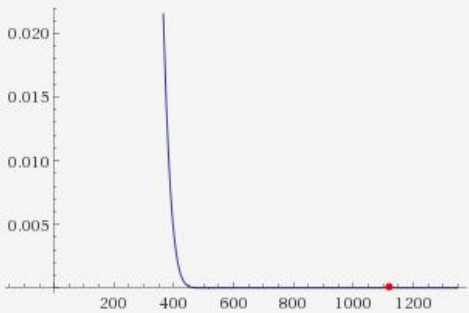
$$\exp\left(-\frac{1}{10\exp(-0.01\,x)}\right)=0$$

Open code

Result:

$$e^{-\frac{e^{0.01\,x}}{10}}=0$$

Root plot:



4. Script experimento 4

```
#imports
import subprocess
import csv
from matplotlib import pyplot as plt
import numpy as np

#Configuration of the experiment
seed_c = 1234
seed_s = 4321

num_c = 4
num_s = 100

operant_id = 1
initial_solution_id = 1

#Search Alianning parameters
k = 10
l = 0.01
i = 3000
s = 30

program = []

program.append("java")
program.append("-jar")
program.append("CodiPractica/out/artifacts/CodiPractica_jar/CodiPractica.jar")

program.append(str(num_c))
program.append(str(num_s))
program.append(str(seed_c))
program.append(str(seed_s))
program.append(str(operant_id))
program.append(str(initial_solution_id))

program.append(str(k))
program.append(str(l))
program.append(str(i))
program.append(str(s))

seeds_s = [1313, 1122, 2233, 3344, 4455, 5566, 6677, 7788, 8899, 9900]
seeds_c = [1100, 2211, 3322, 4433, 5544, 6655, 7766, 8877, 9988, 3141]

listaHeurist = [[],[],[],[]]
listaCoste = [[],[],[],[]]
listaDatos = [[],[],[],[]]

listaMilis = [[],[],[],[]]
listaExpanded= [[],[],[],[]]
```

```
for rep in range(10):
    print 'rep',rep
    program[5] = str(seeds_c[rep])
    program[6] = str(seeds_s[rep])
    for ite in range(4):
        print ite
        program[3]=str(num_c+2*ite)
        program[4]=str(num_s+50*ite)
        output = subprocess.Popen(program, stdout=subprocess.PIPE).communicate()[0]
        reader = csv.reader(output.splitlines())
        listaCsv = list(reader)
        listaHeurist[ite].append(eval(listaCsv[-3][2]))
        listaCoste[ite].append(eval(listaCsv[-3][1]))
        listaDatos[ite].append(eval(listaCsv[-3][0]))
        listaMilis[ite].append(eval(listaCsv[-2][0]))
        listaExpanded[ite].append(eval(listaCsv[-1][0]))

plt.boxplot(listaHeurist)
plt.suptitle('Heuristic')
plt.autoscale()
plt.xlabel('Escala problema')
plt.ylabel('Valor heuristic')
plt.savefig('./experiment4/heuristic.png',bbox_inches='tight')
plt.clf()
plt.boxplot(listaCoste)
plt.suptitle('Cost')
plt.xlabel('Escala problema')
plt.ylabel('Cost de la xarxa')
plt.savefig('./experiment4/coste.png',bbox_inches='tight')
plt.clf()
plt.boxplot(listaDatos)
plt.suptitle('Dades rebudes (respecte el total que es recull)')
plt.xlabel('Escala problema')
plt.ylabel('Dades rebudes')
plt.savefig('./experiment4/datos.png',bbox_inches='tight')
plt.clf()
plt.boxplot(listaMilis)
plt.suptitle('Temps')
plt.xlabel('Escala problema')
plt.ylabel('ms')
plt.savefig('./experiment4/tiempo.png',bbox_inches='tight')
plt.clf()
```

5. Experimento 5

5.1. Script para ver los centros inutilizados en Hill Climbing con diferentes semillas

```
#imports
import subprocess
import csv
from matplotlib import pyplot as plt
import numpy as np

#Configuration of the experiment
seed_c = 1234
seed_s = 4321

num_c = 4
num_s = 100

operant_id = 1
initial_solution_id = 1

#Search Alianning parameters
k = 10
l = 0.01
i = 3000
s = 30

program = []

program.append("java")
program.append("-jar")
program.append("CodiPractica/out/artifacts/CodiPractica_jar/CodiPractica.jar")

program.append(str(num_c))
program.append(str(num_s))
program.append(str(seed_c))
program.append(str(seed_s))
program.append(str(operant_id))
program.append(str(initial_solution_id))

program.append(str(k))
program.append(str(l))
program.append(str(i))
program.append(str(s))

seeds_s = [1313, 1122, 2233, 3344, 4455, 5566, 6677, 7788, 8899, 9900]
seeds_c = [1100, 2211, 3322, 4433, 5544, 6655, 7766, 8877, 9988, 3141]

listaHeurist = [[],[],[],[],[],[],[],[],[],[]]
listaCoste = [[],[],[],[]]
listaDatos = [[],[],[],[]]
```

```
listaMilis = [[],[],[],[]]
listaExpanded= [[],[],[],[]]

listaUnusedCenters = [[],[],[],[],[],[],[],[],[],[]]

for rep in range(10):
    print 'rep',rep
    program[5] = str(seeds_c[rep])
    program[6] = str(seeds_s[rep])
    for ite in range(1):
        output = subprocess.Popen(program, stdout=subprocess.PIPE).communicate()[0]
        #print output
        reader = csv.reader(output.splitlines())
        listaCsv = list(reader)
        listaUnusedCenters[rep].append(eval(listaCsv[-3][3]))

plt.boxplot(listaUnusedCenters)
plt.autoscale()
plt.suptitle('Centros no usados')
plt.xlabel('Diferentes semillas Hill Climbing')
plt.ylabel('#Centros inutilizados')
plt.savefig('./experiment5/centrosInutilizadosHill.png',bbox_inches='tight')
plt.clf()
```

5.2. Script para ver los centros inutilizados usando Simulated Annealing con diferentes semillas y diferentes parámetros k, lambda, iteraciones y steps

```
#imports
import subprocess
import csv
from matplotlib import pyplot as plt
import numpy as np
#Configuration of the experiment
seed_c = 1234
seed_s = 4321
num_c = 4
num_s = 100
operant_id = 1
initial_solution_id = 1

#Search Alianning parameters
k = 10
l = 0.01
i = 3000
s = 30

program = []

program.append("java")
program.append("-jar")
program.append("CodiPractica/out/artifacts/CodiPractica_jar/CodiPractica.jar")
```

```
program.append(str(num_c))
program.append(str(num_s))
program.append(str(seed_c))
program.append(str(seed_s))
program.append(str(operant_id))
program.append(str(initial_solution_id))

program.append(str(k))
program.append(str(l))
program.append(str(i))
program.append(str(s))

seeds_s = [1313, 1122, 2233, 3344, 4455, 5566, 6677, 7788, 8899, 9900]
seeds_c = [1100, 2211, 3322, 4433, 5544, 6655, 7766, 8877, 9988, 3141]

k = [10,10,10,10,5,5,5,5,10,10,10,10,20,20,20,20]
lamb = [0.01,0.01,0.01,0.01,0.1,0.1,0.1,0.1,0.005,0.005,0.005,0.005,0.01,0.01,0.01,0.01]
iters = [2244,2244,1500,1500,830,830,1200,1200,4000,4000,3200,3200,11460,11460,8000,8000]
steps = [4,561,15,100,10,83,12,100,4,1000,32,100,10,1146,8,1000]

listaHeurist = [[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]
listaCoste = [[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]
listaDatos = [[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]

listaMilis = [[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]
listaExpanded= [[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]

listaUnusedCenters = [[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]

for rep in range(10):
    print 'rep',rep
    program[5] = str(seeds_c[rep])
    program[6] = str(seeds_s[rep])
    for ite in range(16):
        print ite
        program[-1] = str(steps[ite])
        program[-2] = str(iters[ite])
        program[-3] = str(lamb[ite])
        program[-4] = str(k[ite])
        output = subprocess.Popen(program, stdout=subprocess.PIPE).communicate()[0]
        reader = csv.reader(output.splitlines())
        listaCsv = list(reader)
        listaUnusedCenters[ite].append(eval(listaCsv[-3][3]))

plt.boxplot(listaUnusedCenters)
plt.autoscale()
plt.suptitle('Centros no usados')
plt.xlabel('Diferentes parametros SA')
plt.ylabel('#Centros inutilizados')
plt.savefig('./experiment5/centrosInutilizadosSA.png',bbox_inches='tight')
plt.clf()
```

5.3. Script para ver si el número de centros inutilizados varía en función del número de centros y/o del número de sensores manteniendo siempre la misma proporción

```
#imports
import subprocess
import csv
from matplotlib import pyplot as plt
import numpy as np

#Configuration of the experiment
seed_c = 1234
seed_s = 4321

num_c = 4
num_s = 100

operant_id = 1
initial_solution_id = 1

#Search Alianning parameters
k = 10
l = 0.01
i = 3000
s = 30

program = []

program.append("java")
program.append("-jar")
program.append("CodiPractica/out/artifacts/CodiPractica_jar/CodiPractica.jar")

program.append(str(num_c))
program.append(str(num_s))
program.append(str(seed_c))
program.append(str(seed_s))
program.append(str(operant_id))
program.append(str(initial_solution_id))

program.append(str(k))
program.append(str(l))
program.append(str(i))
program.append(str(s))

seeds_s = [1313, 1122, 2233, 3344, 4455, 5566, 6677, 7788, 8899, 9900]
seeds_c = [1100, 2211, 3322, 4433, 5544, 6655, 7766, 8877, 9988, 3141]

listaHeurist = [[],[],[],[]]
listaCoste = [[],[],[],[]]
listaDatos = [[],[],[],[]]

listaMilis = [[],[],[],[]]
```

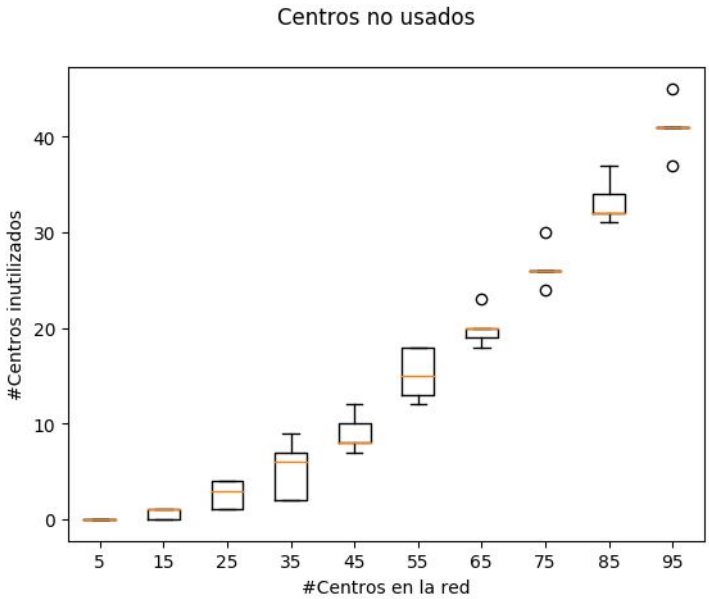
```
listaExpanded= [[],[],[],[]]

listaUnusedCenters = [[],[],[],[]]

for rep in range(10):
    print 'rep',rep
    program[5] = str(seeds_c[rep])
    program[6] = str(seeds_s[rep])
    for ite in range(4):
        print ite
        program[3]=str(num_c+2*ite)
        program[4]=str(num_s+50*ite)
        output = subprocess.Popen(program, stdout=subprocess.PIPE).communicate()[0]
        reader = csv.reader(output.splitlines())
        listaCsv = list(reader)
        listaUnusedCenters[ite].append(eval(listaCsv[-3][3]))
        listaHeurist[ite].append(eval(listaCsv[-3][2]))
        listaCoste[ite].append(eval(listaCsv[-3][1]))
        listaDatos[ite].append(eval(listaCsv[-3][0]))
        listaMilis[ite].append(eval(listaCsv[-2][0]))
        listaExpanded[ite].append(eval(listaCsv[-1][0]))

plt.boxplot(listaUnusedCenters)
plt.autoscale()
plt.suptitle('Centros no usados')
plt.xlabel('Diferentes semillas')
plt.ylabel('#Centros inutilizados')
plt.savefig('./experiment5/centrosInutilizadosScale.png',bbox_inches='tight')
plt.clf()
```

5.4. Script para ver si el número de centros inutilizados varía en función del número de centros

```
#imports
import subprocess
import csv
from matplotlib import pyplot as plt
import numpy as np

#Configuration of the experiment
seed_c = 1234
seed_s = 4321

num_c = 5
num_s = 100

operant_id = 1
initial_solution_id = 1

#Search Alianning parameters
k = 10
l = 0.01
i = 3000
s = 30
```

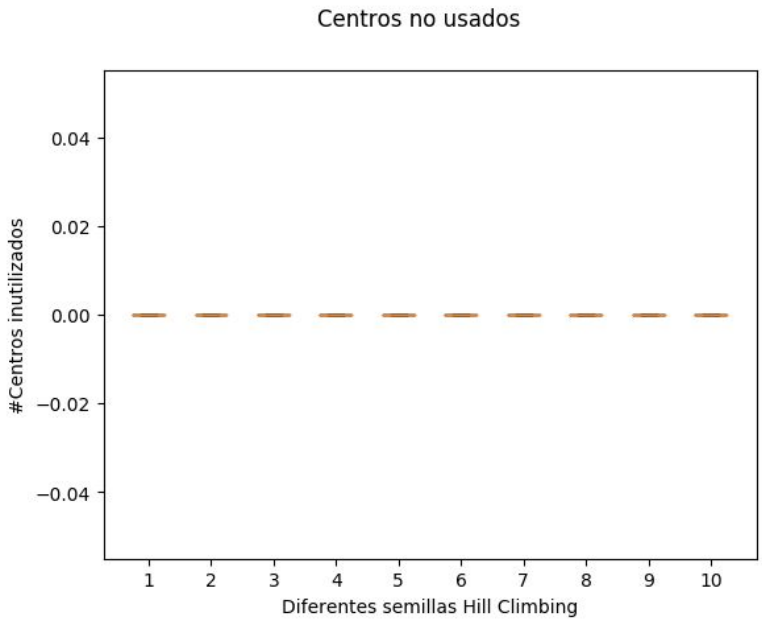
```
program = []

program.append("java")
program.append("-jar")
program.append("CodiPractica/out/artifacts/CodiPractica_jar/CodiPractica.jar")

program.append(str(num_c))
program.append(str(num_s))
program.append(str(seed_c))
program.append(str(seed_s))
program.append(str(operant_id))
program.append(str(initial_solution_id))

program.append(str(k))
program.append(str(l))
program.append(str(i))
program.append(str(s))

program.append('h')

pond = 35000
program.append(str(pond))

seeds_s = [1313, 1122, 2233, 3344, 4455, 5566, 6677, 7788, 8899, 9900]
seeds_c = [1100, 2211, 3322, 4433, 5544, 6655, 7766, 8877, 9988, 3141]

listaUnusedCenters = [[],[],[],[],[],[],[],[],[],[]]

for rep in range(5):
    print 'rep',rep
    program[5] = str(seeds_c[rep])
    program[6] = str(seeds_s[rep])
    for ite in range(10):
        print ite
        program[3]=str(num_c+2*ite)
        output = subprocess.Popen(program, stdout=subprocess.PIPE).communicate()[0]
        reader = csv.reader(output.splitlines())
        listaCsv = list(reader)
        listaUnusedCenters[ite].append(eval(listaCsv[-3][3]))

plt.autoscale()
plt.boxplot(listaUnusedCenters)
plt.suptitle('Centros no usados')
plt.xlabel('#Centros en la red')
plt.xticks([1,2,3,4,5,6,7,8,9,10],[str(i) for i in range(5,200)])
plt.ylabel('#Centros inutilizados')
plt.savefig('./experiment5/centrosInutilizadosDifProporcio.png',bbox_inches='tight')
plt.clf()
```

5.5. Gráfica obtenida a partir de la cual se ha acotado el rango de centros para el experimento



Se ha obtenido usando el script del apartado 5.4 del apéndice incrementando de 10 en 10 el número de centros.

5.6. Experimento 5 con 4 centros, 10 sensores y diferentes semillas usando Hill Climbing



6. Experimento 6

6.1. Script usado para Hill Climbing

```
#imports
import subprocess
import csv
from matplotlib import pyplot as plt
import numpy as np

#Configuration of the experiment
seed_c = 1234
seed_s = 4321

num_c = 2
num_s = 100

operant_id = 1
initial_solution_id = 1

alg_m = "h"

program = []

program.append("java")
program.append("-jar")
program.append("CodiPractica/out/artifacts/CodiPractica_jar/CodiPractica.jar")

program.append(str(num_c))
program.append(str(num_s))
program.append(str(seed_c))
program.append(str(seed_s))
program.append(str(operant_id))
program.append(str(initial_solution_id))

#Search Alianning parameters
k = 10
l = 0.01
i = 3000
s = 30
program.append(str(k))
program.append(str(l))
program.append(str(i))
program.append(str(s))
program.append(alg_m)

#numc nums seedc seeds opid genid

seeds_s = [1313, 1122, 2233, 3344, 4455, 5566, 6677, 7788, 8899, 9900]
seeds_c = [1100, 2211, 3322, 4433, 5544, 6655, 7766, 8877, 9988, 3141]

listaHeurist = [[],[],[],[],[]]
```

```
listaCoste = [[],[],[],[],[]]
listaDatos = [[],[],[],[],[]]
listaMilis = [[],[],[],[],[]]
listaExpanded= [[],[],[],[],[]]

listaUnusedCenters = [[],[],[],[],[]]

for rep in range(10):
    print 'r',rep
    program[5] = str(seeds_c[rep])
    program[6] = str(seeds_s[rep])
    for op in range(5):
        print op
        program[3] = str(num_c + 2*op)
        output = subprocess.Popen(program, stdout=subprocess.PIPE).communicate()[0]
        #file = open("./experiment2/"+ str(rep) + "-"+ str(op) +".out","w")
        #file.write(output)
        #file.close()
        #print output
        reader = csv.reader(output.splitlines())
        listaCsv = list(reader)
        listaUnusedCenters[op].append(eval(listaCsv[-3][3]))
        listaHeurist[op].append(eval(listaCsv[-3][2]))
        listaCoste[op].append(eval(listaCsv[-3][1]))
        listaDatos[op].append(eval(listaCsv[-3][0]))
        listaMilis[op].append(eval(listaCsv[-2][0]))
        listaExpanded[op].append(eval(listaCsv[-1][0]))

plt.autoscale()
plt.boxplot(listaCoste)
plt.suptitle('Cost')
plt.xticks([1,2,3,4,5],['2','4','6','8','10'])
plt.xlabel('Numero de centros')
plt.ylabel('Coste de la red')
plt.savefig('./experiment6/costeHill.png',bbox_inches='tight')
plt.clf()
plt.boxplot(listaMilis)
plt.suptitle('Tiempo')
plt.xticks([1,2,3,4,5],['2','4','6','8','10'])
plt.xlabel('Numero de centros')
plt.ylabel('ms')
plt.savefig('./experiment6/tiempoHill.png',bbox_inches='tight')
plt.clf()
plt.boxplot(listaUnusedCenters)
plt.suptitle('Centros inutilizados')
plt.xticks([1,2,3,4,5],['2','4','6','8','10'])
plt.xlabel('Numero de centros')
plt.ylabel('Numero de centros no usados')
plt.savefig('./experiment6/tiempoHill.png',bbox_inches='tight')
plt.clf()
```

6.2 Script usado para Simulated Annealing

```
#imports
import subprocess
import csv
from matplotlib import pyplot as plt
import numpy as np

#Configuration of the experiment
seed_c = 1234
seed_s = 4321

num_c = 2
num_s = 100

operant_id = 1
initial_solution_id = 1

alg_m = "s"

program = []

program.append("java")
program.append("-jar")
program.append("CodiPractica/out/artifacts/CodiPractica_jar/CodiPractica.jar")

program.append(str(num_c))
program.append(str(num_s))
program.append(str(seed_c))
program.append(str(seed_s))
program.append(str(operant_id))
program.append(str(initial_solution_id))

#Search Alianning parameters
k = 10
l = 0.01
i = 3000
s = 30
program.append(str(k))
program.append(str(l))
program.append(str(i))
program.append(str(s))
program.append(alg_m)

#numc nums seedc seeds opid genid

seeds_s = [1313, 1122, 2233, 3344, 4455, 5566, 6677, 7788, 8899, 9900]
seeds_c = [1100, 2211, 3322, 4433, 5544, 6655, 7766, 8877, 9988, 3141]

k = [10,10,10,10,5,5,5,5,10,10,10,10,20,20,20,20]
lamb = [0.01,0.01,0.01,0.01,0.1,0.1,0.1,0.1,0.005,0.005,0.005,0.005,0.01,0.01,0.01,0.01]
iters = [2244,2244,1500,1500,830,830,1200,1200,4000,4000,3200,3200,11460,11460,8000,8000]
steps = [4,561,15,100,10,83,12,100,4,1000,32,100,10,1146,8,1000]
```

```
listaHeurist = [[],[],[],[],[]]
listaCoste = [[],[],[],[],[]]
listaDatos = [[],[],[],[],[]]

listaMilis = [[],[],[],[],[]]
listaExpanded= [[],[],[],[],[]]

listaUnusedCenters = [[],[],[],[],[]]

for rep in range(10):
    print 'r',rep
    program[5] = str(seeds_c[rep])
    program[6] = str(seeds_s[rep])
    for op in range(5):
        print op
        program[3] = str(num_c + 2*op)
        program[-2] = str(steps[8])
        program[-3] = str(iters[8])
        program[-4] = str(lamb[8])
        program[-5] = str(k[8])
        output = subprocess.Popen(program, stdout=subprocess.PIPE).communicate()[0]
        reader = csv.reader(output.splitlines())
        listaCsv = list(reader)
        listaUnusedCenters[op].append(eval(listaCsv[-3][3]))
        listaHeurist[op].append(eval(listaCsv[-3][2]))
        listaCoste[op].append(eval(listaCsv[-3][1]))
        listaDatos[op].append(eval(listaCsv[-3][0]))
        listaMilis[op].append(eval(listaCsv[-2][0]))
        listaExpanded[op].append(eval(listaCsv[-1][0]))

plt.autoscale()
plt.boxplot(listaCoste)
plt.suptitle('Cost')
plt.xticks([1,2,3,4,5],['2','4','6','8','10'])
plt.xlabel('Numero de centros')
plt.ylabel('Coste de la red')
plt.savefig('./experiment6/costeSim.png',bbox_inches='tight')
plt.clf()
plt.boxplot(listaMilis)
plt.suptitle('Tiempo')
plt.xticks([1,2,3,4,5],['2','4','6','8','10'])
plt.xlabel('Numero de centros')
plt.ylabel('ms')
plt.savefig('./experiment6/tiempoSim.png',bbox_inches='tight')
plt.clf()
plt.boxplot(listaUnusedCenters)
plt.suptitle('Centros inutilizados')
plt.xticks([1,2,3,4,5],['2','4','6','8','10'])
plt.xlabel('Numero de centros')
plt.ylabel('Numero de centros no usados')
plt.savefig('./experiment6/tiempoSim.png',bbox_inches='tight')
plt.clf()
```

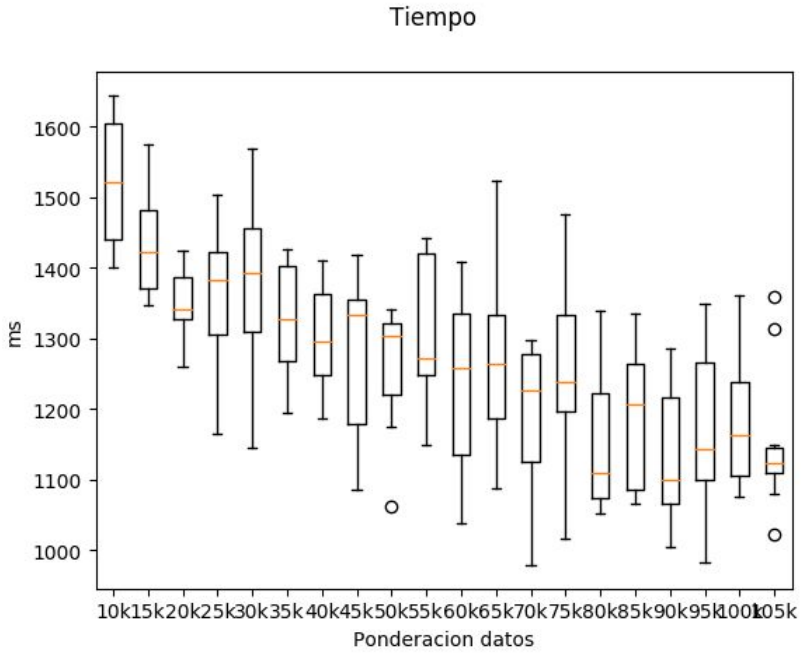
7. Experimento 7

7.1 Script

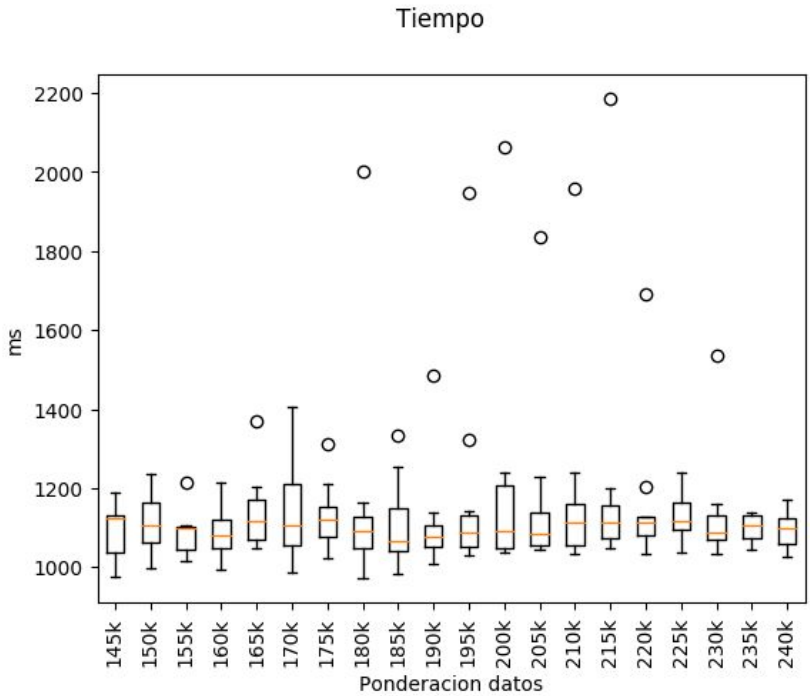
```
#imports
import subprocess
import csv
from matplotlib import pyplot as plt
import numpy as np

#Configuration of the experiment
seed_c = 1234
seed_s = 4321

num_c = 2
num_s = 100

operant_id = 1
initial_solution_id = 1

alg_m = "h"

program = []

program.append("java")
program.append("-jar")
program.append("CodiPractica/out/artifacts/CodiPractica_jar/CodiPractica.jar")

program.append(str(num_c))
program.append(str(num_s))
program.append(str(seed_c))
program.append(str(seed_s))
program.append(str(operant_id))
program.append(str(initial_solution_id))

#Search Alianning parameters
k = 10
l = 0.01
i = 3000
s = 30
program.append(str(k))
program.append(str(l))
program.append(str(i))
program.append(str(s))
program.append(alg_m)

pond = 145000#35000
program.append(str(pond))

#numc nums seedc seeds opid genid

seeds_s = [1313, 1122, 2233, 3344, 4455, 5566, 6677, 7788, 8899, 9900]
```

```
seeds_c = [1100, 2211, 3322, 4433, 5544, 6655, 7766, 8877, 9988, 3141]

listaHeurist = [[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]
listaCoste = [[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]
listaDatos = [[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]
listaMilis = [[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]
listaExpanded= [[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]
listaUnusedCenters = [[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[],[]]

for rep in range(10):
    print 'r',rep
    program[5] = str(seeds_c[rep])
    program[6] = str(seeds_s[rep])
    for ite in range(20):
        print ite
        #program[3] = str(2*(ite+1))
        program[14] = str(pond + 5000*ite)
        output = subprocess.Popen(program, stdout=subprocess.PIPE).communicate()[0]
        reader = csv.reader(output.splitlines())
        listaCsv = list(reader)
        listaUnusedCenters[ite].append(eval(listaCsv[-3][3]))
        listaHeurist[ite].append(eval(listaCsv[-3][2]))
        listaCoste[ite].append(eval(listaCsv[-3][1]))
        listaDatos[ite].append(eval(listaCsv[-3][0]))
        listaMilis[ite].append(eval(listaCsv[-2][0]))
        listaExpanded[ite].append(eval(listaCsv[-1][0]))

plt.autoscale()
plt.boxplot(listaCoste)
plt.suptitle('Cost')
plt.xticks([i for i in range(1,21)],[str(i)+'k' for i in range(145,145+(5*20)+1,5)],
rotation='vertical')
plt.xlabel('Ponderacion datos')
plt.ylabel('Coste de la red')
plt.savefig('./experiment7/costeHill.png',bbox_inches='tight')
plt.clf()

plt.autoscale()
plt.boxplot(listaDatos)
plt.suptitle('Datos')
plt.xticks([i for i in range(1,21)],[str(i)+'k' for i in range(145,145+(5*20)+1,5)],
rotation='vertical')
plt.xlabel('Ponderacion datos')
plt.ylabel('Datos recibidos')
plt.savefig('./experiment7/datosHill.png',bbox_inches='tight')
plt.clf()
plt.boxplot(listaMilis)
plt.suptitle('Tiempo')
plt.xticks([i for i in range(1,21)],[str(i)+'k' for i in range(145,145+(5*20)+1,5)],
rotation='vertical')
plt.xlabel('Ponderacion datos')
plt.ylabel('ms')
plt.savefig('./experiment7/tiempoHill.png',bbox_inches='tight')
plt.clf()
```

7.2 Gráficas de tiempo de hallar una solución con diferentes ponderaciones de los datos en el heurístico

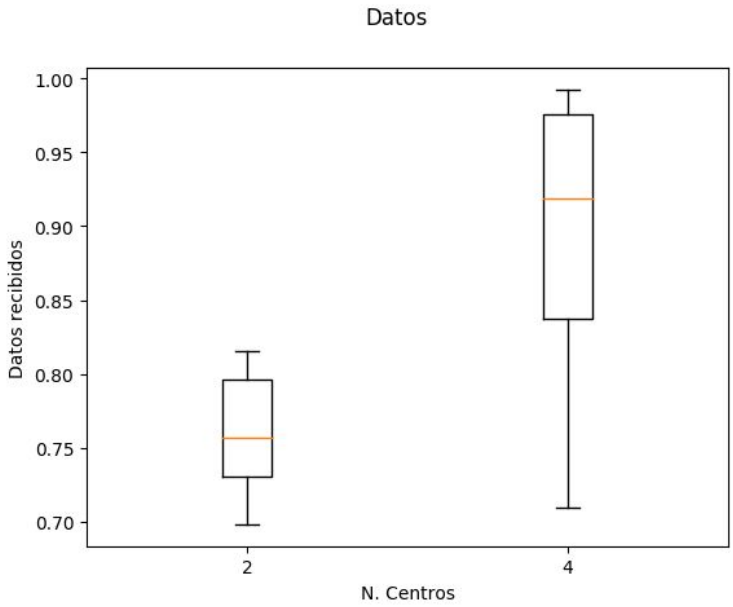


En la gráfica se puede apreciar como a mesura que aumenta la ponderación de los datos del heurístico disminuye el tiempo de ejecución. Eso es debido a que como más altas son las ponderaciones más pesan en la función heurística y hay más diferencias entre los estados por lo que la función heurística encontrará estados mejores más rápidamente.



En esta gráfica se puede ver que el estancamiento del tiempo de ejecución es permanente para valores superiores a 145k. *(Esta gráfica es la continuación de la mostrada en el experimento 7).*

7.3. Aumento en proporción del coste de la red al pasar de 4 centros a 2



La porción de datos que llega a los 4 centros en una red con 100 sensores es de 0.92 y en una red con 2 centros es de 0.76. Por lo tanto al pasar de 4 centros a 2 la proporción varía en 0.76/0.92 = 0.826.

7.4. Estancamiento del coste de la red al aumentar la ponderación de los datos en la función heurística