Distribución de datos en una red de sensores

Práctica de algoritmos de búsqueda local

César Mellado
Leo Arriola
Zixuan Sun

Inteligencia Artificial
Abril, 2021

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Facultat d'Informàtica de Barcelona (FIB)

CONTENIDO

1. Introducción

El objetivo de esta práctica es trabajar con dos algoritmos de *búsqueda local*: ***Hill Climbing*** y ***Simulated Annealing***. Para ello trabajaremos con la librería del **AIMA** en *Java,* la cual nos proporcionará la implementación de ambos algoritmos. Concretamente, en esta práctica se trabaja en la distribución de conexiones de una red dado un conjunto de sensores y centros de datos.
El objetivo secundario de la práctica será **experimentar** con las diferentes *variables* y los diferentes *parámetros* que admiten los algoritmos para seguidamente compararlos y evaluar la calidad de las soluciones generadas.

2. Descripción y elementos del problema

En esta práctica se nos plantea un problema de **optimización** de la distribución de las conexiones entre diferentes sensores y centros de datos en un **plano euclídeo**. Concretamente, queremos minimizar el coste de las conexiones y maximizar el número de datos transmitidos que llegan a los centros de datos por los sensores en una superficie de dimensión *100 x 100 km^2*.

2.1 Sensores

En esta área geográfica hay distribuidos *S* **sensores** que capturan distintos volúmenes de datos, concretamente, pueden capturar *1, 2* y *5 Mb/s*. Estos sensores tienen la capacidad de establecer una **conexión** (otro sensor o un centro de datos) y transmitir toda la información que captura y almacena. El máximo de datos almacenados por un sensor es el doble de su volumen de captura, y como consecuencia, el número máximo de datos transmitidos es el *triple*. Una característica de estos sensores es que como **máximo** pueden tener conectados ***tres*** **sensores** a la vez a ellos. Como simplificación asumiremos que toda información capturada en un segundo por todos los sensores es transmitida instantáneamente a los centros de datos sin importar el número de conexiones intermedias. Como restricción tenemos que todo sensor tiene que estar conectado a otro sensor o a un centro de datos.

2.2 Centros de datos

También tenemos distribuidos *C* **centros** de datos en esta misma área. Cada centro de datos puede recibir hasta *150 Mb/s* y tener hasta 25 diferentes sensores conectados **simultáneamente** a él.

2.3 Coste

Como hemos descrito anteriormente, queremos minimizar el coste y maximizar el número de datos recibidos por los centros de datos. El coste para conectar un **sensor *A*** a un **sensor *B*** está determinado por la siguiente ecuación:

$$\mathbf{coste(A,B) = distancia(A,B)^2 \times v(A)}$$

siendo $\mathbf{v(A)}$ el volumen de datos que transmite el *sensor A* al *sensor B*, y la distancia entre estos es su distancia euclídea:

$$\mathbf{distancia(A,B)} = \sqrt{(A_x - B_x)^2 + (A_y - B_y)^2}$$

3. Criterio de obtención de soluciones

Para el problema planteado para esta práctica, seguiremos los dos siguientes criterios:

- Intentar **maximizar** el número de **datos recibidos** en total por cada centro de datos
- Intentar **minimizar** el **coste** total de la red de conexiones de los sensores

Concretamente, tendremos una ponderación con un factor *lambda* λ que nos indique si queremos priorizar más el primer criterio o el segundo.

4. Representación de un estado del problema

Nuestras primeras ideas contemplan la representación del problema mediante el uso de *DAG*'s (**grafo dirigido acíclico**), estos estarían representados mediante una **matriz**. Esta matriz representaría las adyacencias entre los diferentes sensores y/o centros de datos. Después de contemplar esta estructura de datos vimos que es **innecesaria**, ya que no se consideran algunas de las restricciones del problema, específicamente que un sensor sólo puede apuntar a un elemento, este sea o un sensor o un centro de datos.

Teniendo en cuenta estas restricciones contemplamos la posibilidad de utilizar **vectores** en vez de utilizar matrices, la cual tiene un coste espacial elevado (sobre todo a la hora de generar sucesores dado un estado). Una **primera idea** ha sido la de implementar un **vector de vectores**: un vector de centros de datos *C* con *i* centros de datos en la que en la posición *i* se encontraría el centro de datos *i*. Cada centro de datos de *C* apuntaría a un vector de sensores con *j sensores*, donde cada sensor *C[i][j]* representa un sensor que apunta directa o indirectamente al centro de datos *i*.

Con esta idea nos hemos encontrado un problema que **complica** la resolución de los experimentos: en esta estructura de datos estaríamos teniendo en cuenta que un *Centro de datos* tiene una serie de sensores "predefinidos" que apuntan a este centro, pero si el algoritmo de resolución del problema detecta que es **más eficiente** en coste que un sensor pase de estar conectado a un centro de datos a otro, el operador de cambio de apuntado del sensor sería bastante complicado de implementar.

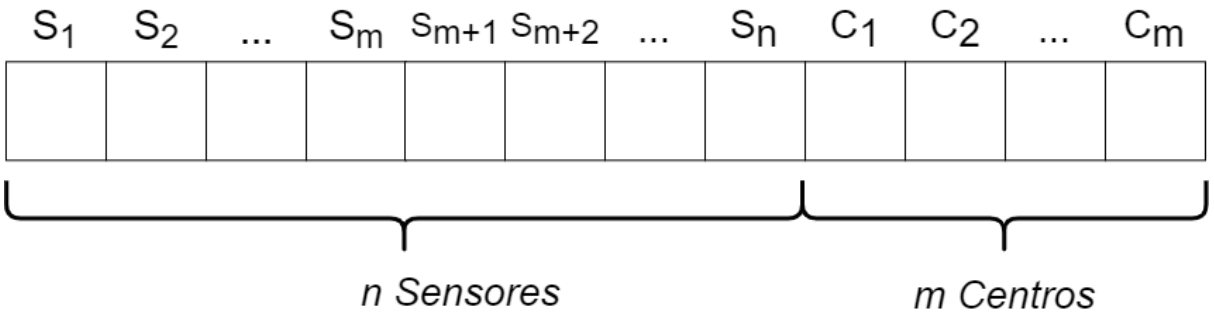
Por ejemplo, en el centro de datos *0* tenemos el sensor *C[0][1]*, pero el algoritmo ve que este sensor sería mejor que apuntase a *C[1][1]*, por lo que moveríamos este sensor al vector de sensores de *C[1]*, pero este vector puede que ya tenga un vector *S[1]*, por lo que habría que plantearse mover este sensor al final del vector, o el sensor que movemos ponerlo al principio y desplazar los sensores una posición, etc. También habría que reducir el tamaño del vector de donde movemos el vector. En fin, un **caos**. Además que esta representación no se aleja mucho de la representación por matriz de adyacencias.

La siguiente idea que tuvimos, un poco más óptima y fácil de tratar, sería mediante un objeto *red* que estaría formado por *dos* vectores:

- El *primer* **vector** sería de objetos de ***sensores***, el objeto sensor tendría como atributos un puntero a otro sensor y otro atributo que indica el nombre de sensores que tiene conectados a él, este atributo creemos que sería práctico para los operadores.
- El *segundo* **vector** de la clase *red* sería de objetos de ***centros de datos***, estos tendrían como atributo la capacidad acumulada y un contador con el número de sensores que tiene conectados para la configuración dada.

Pensándolo se nos ocurrió una duda con respecto a las capacidades de los sensores, y la cantidad de datos que han acumulado. Nos planteamos si valía la pena crear un vector para **consultar** rápidamente el almacenamiento de un sensor, este vector sería otro atributo de la clase red. Lo consideramos una opción, porque creemos que sería útil cara a la heurística.

Finalmente hemos decidido **simplificarlo** todo más, y representar un estado mediante un solo vector de tamaño *s+c* (*sensores* + *centros de datos*), donde en las primeras *s* posiciones guardaremos el índice del sensor el cual apunta el sensor *i*; mientras que en las *c* posiciones finales del vector guardaran en cada centro de datos *j* la suma de los datos que recibe este centro, que será la suma de datos que emiten los sensores conectados directamente o indirectamente a este centro de datos.



5. Operadores de estado

Teniendo en cuenta que nuestra representación de estados consiste en un solo vector de tamaño (*s+c*), hemos decidido implementar **un único operador**: *ChangeConnexion()*.

Este operador lo que hace básicamente es **cambiar** el **contenido** de **una posición** del vector que representa un sensor (las *s* primeras posiciones). De esta manera representaremos que un sensor ha cambiado el sensor o centro de datos al cual apunta.

A lo largo de la práctica, con ayuda de las indicaciones del profesor hemos visto que nos puede resultar práctico **añadir otro operador**, hemos decidido añadir el operador de *swap()* de dos conexiones. Lo que hace este operador es **intercambiar** las conexiones entre dos sensores. Por ejemplo: si el sensor *S1* apunta a *S2* y el sensor *S3* apunta a *S4*, con la llamada a *SwapConnexion()*, *S1* pasará a apuntar a *S4* y *S3* apuntará a *S2*.

Este operador suponemos que puede **reducir** la cantidad de sucesores que se generan y por eso nos da cierta ventaja respecto al anterior. Además nos da la opción de poder comparar y *testear* (realizado en el experimento 1) con diferentes operadores y poder ver cuál minimiza al máximo nuestro valor heurístico.

Para evitar salir del conjunto de soluciones añadimos las **restricciones** necesarias en la *getSuccesorFunction()* estas funciones garantizan que **no haya ciclos**, que no se superen el número de conexiones, etc...

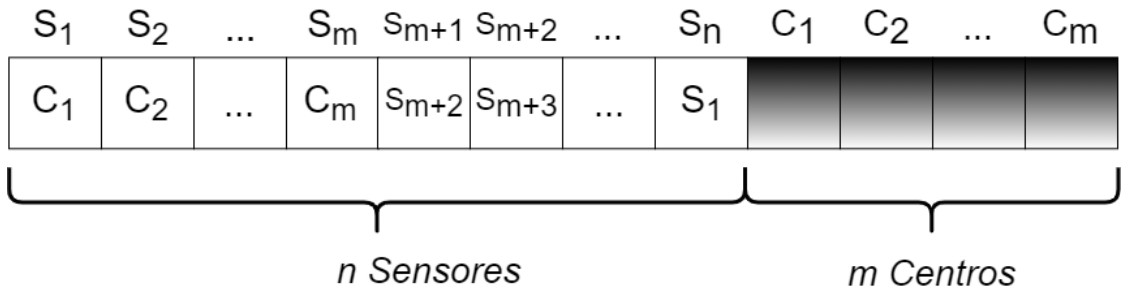
También, dado que nuestros operadores modifican los datos transmitidos tenemos que recorrer los antiguos caminos para actualizar la cantidad de datos que se transmiten después de la modificación.

6. Generación de una solución inicial

Como primera solución, optamos por generar la asignación de los sensores al **azar**, esta solución aunque no considera el coste de conexión, teóricamente sí nos generaba configuraciones iniciales muy diversas. Lo hacíamos mediante un generador de números *pseudoaleatorios* el cual nos indicaba a donde apuntaba el sensor tratado. Durante esta asignación se comprobaban las **restricciones** de los límites de los grados de los nodos sensores y centros. Desgraciadamente no consideramos la creación de ciclos y abandonamos esta idea.

6.1 *Versión 1*

Una manera de generar una solución inicial, bastante más **sencilla**, es la siguiente: si suponemos que hay *n* sensores y *m* centros de datos (donde *n >= m*), con nuestra representación del estado tendríamos un vector de tamaño ***n+m***. Para generar una solución inicial lo que podemos hacer es forzar que los primeros *m* sensores apunten a los *m* centros de datos directamente, de esta manera cubrimos la restricción principal de que todos los sensores son apuntados por al menos un sensor. Los sensores restantes apuntarán al siguiente sensor excepto el último que apuntará al primero de todos, de esta manera cubrimos la otra restricción de que todo sensor apunte a otro (o a un centro de datos). Véase que generar una solución de esta forma nunca se generaría un ciclo entre sensores.



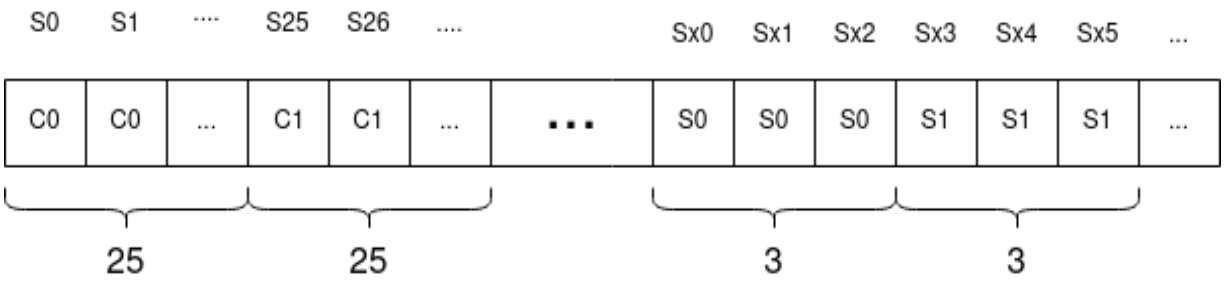
El coste para generar esta solución es **lineal** con respecto al tamaño del vector.

6.2 *Versión 2*

En este caso hemos optado por una solución inicial que intenta **priorizar tener** el máximo número de **sensores conectados** a los centros de datos. Esto lo podemos conseguir recorriendo los sensores, y haciéndolos apuntar al primer centro de datos que no haya superado el límite de tener 25 sensores ya conectados. Una vez llegásemos al número máximo de conexiones permitidas, los sensores restantes se empiezan a conectar entre ellos (de tres en tres se van conectando a los que previamente se habían conectado a los centros de datos).

Finalmente, en el caso que llegasen a llenarse todos los sensores que apuntan a los centros, se empezarían a apuntar **recursivamente**.

Aunque esta solución inicial no considera las distancias creemos que presenta cierta diversidad, similarmente a nuestro intento inicial, cosa que nos **facilitará** la búsqueda local.



7. Generación de los estados sucesores

La generación de los estados sucesores dependerá del algoritmo de búsqueda local que utilicemos, es decir, tendremos dos funciones distintas para generarlos:

- La primera función implementada generadora de sucesores es la dedicada al algoritmo de ***Hill Climbing***, esta genera todas las posibles configuraciones hijas del estado actual. Para hacerlo recorremos todos los sensores y para cada uno intentamos cambiar el sensor o centro de datos al cual apunta, para no generar estados inválidos hacemos el cambio de conexión bajo ciertas restricciones. La nueva conexión no se conectará a un sensor o centro que ya tenga el número de conexiones máximas, la nueva conexión no genera ciclos y finalmente, un sensor no se puede conectar a sí mismo.

- Para el algoritmo de ***Simulated Annealing***, generaremos un solo estado sucesor de forma aleatoria. El estado sucesor generado será el resultado de aplicar el operador sobre un sensor aleatorio para que pase a apuntar a otro (también escogido aleatoriamente; puede ser un centro de datos también) siempre y cuando se ajuste a las restricciones iniciales del enunciado.

8. Funciones heurísticas

Nuestra primera idea de heurística ha sido utilizar la **suma** de todos **los datos** que pueden transmitir todos los sensores del problema. Por ejemplo, si tenemos 4 sensores que transmiten *1, 2, 2* y *5 Mb/s* respectivamente, la heurística sería 10. Entonces, por cada estado sucesor se calculará la suma de datos que transmiten todos los sensores sin perderse, y la distancia de la solución en la que nos encontramos y la heurística sería los datos que se pierden.

Por ejemplo, si de la distribución anterior se pierden *3 Mb/s* debido a las restricciones del enunciado, nos encontraríamos a distancia *3* de una **solución ideal**. Para mejorar nuestra solución tendremos que reducir el número de *Mb/s* que se pierden, hasta llegar a 0 (o en el caso que sea imposible llegar a 0 minimizar al máximo esta distancia).

Finalmente optamos por una heurística que contemplase tanto el **coste de las conexiones** como la **cantidad de datos transmitidos** y que a su vez permitiera cierto "*tunning*", para hacerlo haremos una combinación de los dos valores con una variable de ponderación que llamaremos *lambda* (λ).

Después de analizar con más profundidad el cálculo de la heurística, llegamos a la conclusión de que necesitamos codificar más información de la que previamente habíamos supuesto, en este caso necesitaremos registrar la cantidad de datos que se están transmitiendo *a nivel de* sensor. Acabamos optando por esta heurística:

```
Double HeurVal1 = coste*(100-lambda) - actualData*lambda;
```

Con esta fórmula para calcular la heurística al principio tuvimos bastantes problemas para realizar una búsqueda correctamente. Veíamos que no nos servía para buscar una función **óptima**, y en ese momento no sabíamos el porqué, por lo que decidimos cambiar temporalmente a la heurística:

coste/actualData

Esta heurística nos garantiza que una solución mejor del problema dependiera tanto de la *reducción del coste* como del *aumento de los datos capturados*, y experimentando un poco nos daban resultados **coherentes** dentro de lo que cabía, por lo que optamos por esta opción para realizar el experimento especial.

Al realizar el experimento obtuvimos un porcentaje de pérdida de datos muy **elevada** (más del 40% de datos perdidos), y siguiendo las indicaciones del profesorado, decidimos buscar una manera con la que un posible cliente pudiese decidir si prefiere priorizar el "*pagar menos*" o el tener más datos sin perder, con el objetivo de realizar la búsqueda de una solución óptima con una pérdida de datos menor del 10%.

A causa de esto optamos por volver a la **primera heurística** (es decir, la que tenía el factor *lambda*). Seguíamos teniendo los problemas anteriores hasta que caímos en que el coste aumenta de valor mucho más rápido que se reduce el valor de *actualData*, por lo que si queríamos que los datos capturados fueran los máximos teníamos que darle un valor a *lambda* muy alto, para así reducir la heurística considerablemente en el caso que los datos capturados fueran más.

Finalmente nuestra heurística definitiva es:

```
Double lambda = 99.92;
Double HeurVal1 =  coste*(100-lambda) -  actualData*lambda;
```

Para hacerlo creamos un vector auxiliar llamado *DAGData* similar al de representación del *DAG*, pero que en este caso para cada posición *i*, que guarde la cantidad de datos que está transmitiendo el sensor con identificador *i* (este será *3*capacidad_Sensor*) en el caso de que los datos que se transmiten a este sensor supere el *triple* de su capacidad. Dada nuestra representación de estado, en el caso que la posición *i* apuntará a un centro de datos, entonces se estaría codificando la cantidad de información que este recibe.

Utilizando este vector auxiliar, mejoramos considerablemente la eficiencia del cálculo de la heurística, porque cuando apliquemos nuestro operador de cambio de conexión no hará falta ***recalcular*** de manera exhaustiva el coste de todo el *DAG*, sino solo actualizar los costes de los caminos afectados por el cambio, que son el antiguo camino del sensor en cuestión al centro de datos al cual acabase y el nuevo camino formado.

9. Experimentos

Para ejecutar los experimentos de manera más formalizada, estuvimos experimentando con las heurísticas para que el porcentaje de datos perdidos **no superara** el *10%* dados los parámetros que se nos daban para realizar el experimento especial. Después de experimentar un poco, nos decidimos finalmente con la heurística siguiente:

```
Double lambda = 99.92;
Double HeurVal1 = coste*(100-lambda) - actualData*lambda;
```

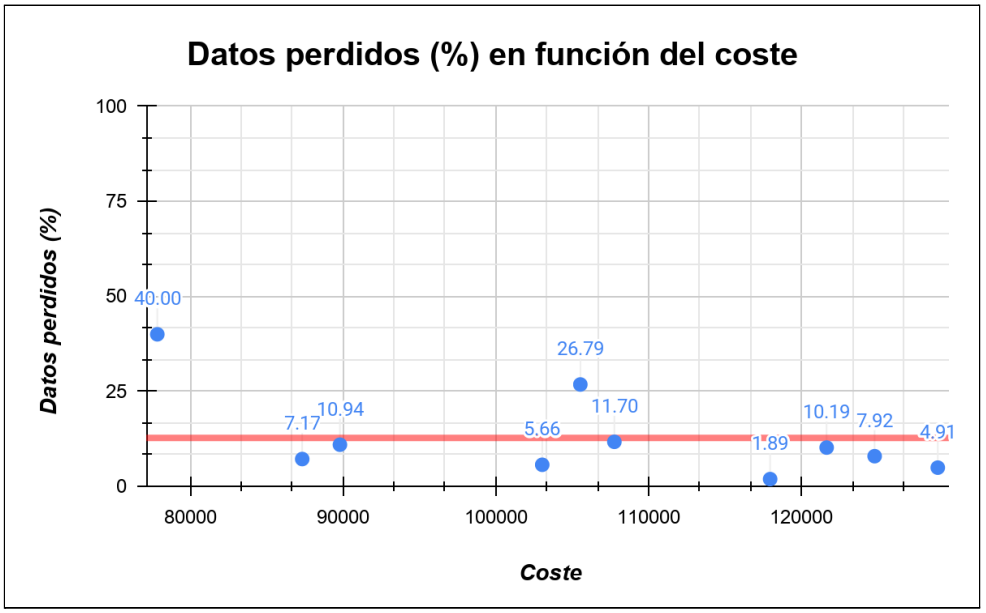
El parámetro *lambda* nos aporta cierta **variabilidad** para *minimizar* el coste o *maximizar* la cantidad de datos transmitidos.

9.1 *Experimento 1*: Operadores

Observación	La combinación del operador *swap* y la solución inicial 2 puede generar anomalías.
Planteamiento	Comprobar que operador optimiza la calidad de la solución propuesta, **maximizar** los datos transmitidos y **minimizar** el coste total.
Hipótesis	Para las condiciones en las cuales evaluamos los operadores, solución inicial, número de centros y sensores, etc. Creemos que el operador *change* podrá llegar a optimizar más que el *swap*.
Método	● Generamos una red con 4 centros y 100 sensores ● Inicializamos la solución inicial utilizando la 2 versión ● Seleccionamos el operador 1 ● Indicamos el algoritmo de búsqueda *Hill Climbing* ● Indicamos la función sucesora para *Hill Climbing* ● Generamos 10 semillas ● Ejecutamos 1 experimento para cada semilla ● Tomamos los valores resultantes para analizarlos ● Repetimos los pasos para el operador 2

Para la resolución del problema, como indicamos anteriormente, decidimos utilizar los operadores de *change connexion* y *swap connexion*. Respecto a la solución inicial, en este caso optamos a utilizar como **solución inicial** la *versión 2*. A continuación mostramos los valores de coste y datos perdidos obtenidos a lo largo de varias ejecuciones modificando la *seed* de generación pero manteniendo el número de centros y sensores.

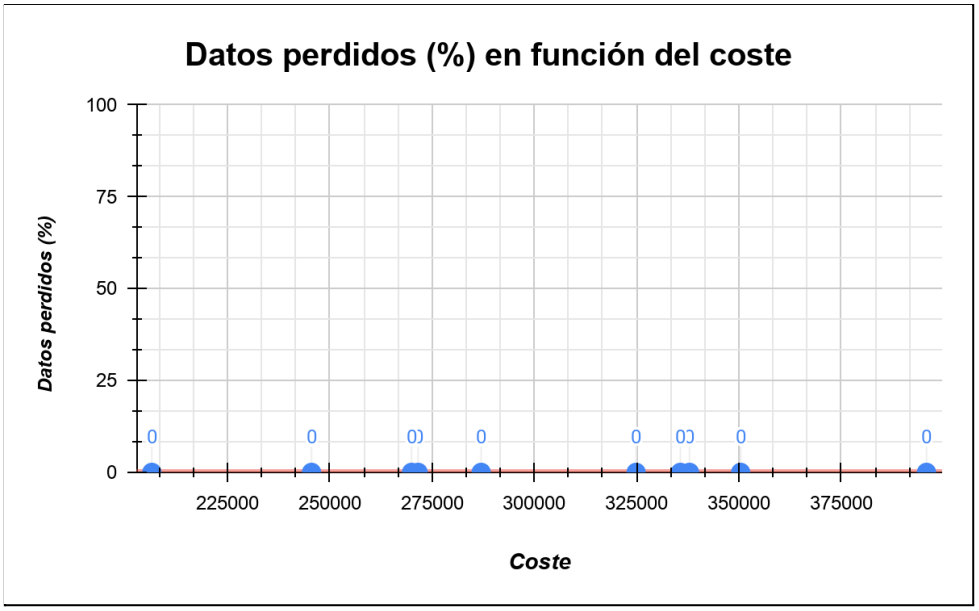
- **Operador *Change Connexion*:**



Para este primer operador vemos que hay mucha **variabilidad** respecto a los costes y la cantidad de datos perdidos, observamos que la media de los datos perdidos está alrededor del *12,72* mientras que el coste tiene una **media** de *106.459,8*.

Esta variabilidad podría explicarse al hecho de que el operador, ya sea por su simplicidad u otras razones esté explorando espacios reducidos y por lo tanto tienda a estancarse en **mínimos locales** con más facilidad. También influyen las **distancias** entre sensores y centros de datos, ya que en cada repetición las coordenadas de estas cambian.

- **Operador *Swap Connexion*:**



Observamos que usando el operador *swap*, la pérdida de datos es **nula**, pero muestra un coste medio más elevado que con el *change connexion*.

Dado que tenemos 100 sensores, 4 centros de datos y que *versión 2* de nuestra solución inicial, que consiste en conectar cada sensor al mismo centro de datos hasta que este llegue a 25 conexiones, seguidamente de conectar los sensores al siguiente centro de datos, esta solución inicial conectará 25 sensores a cada centro de datos **directamente**, por lo que no se perderán datos. Como el operador *Swap* consiste en el cambio de conexiones y no existen conexiones entre sensores, nunca se llegará al caso en el que un sensor se conecte a otro utilizando este operador, y por consecuencia **nunca** se podrá dar el caso de que se pierdan datos.

Este operador es **ideal** para este tipo de solución inicial en la que todos los sensores se conectan directamente a un centro de datos, ya que el algoritmo de búsqueda sólo se encargará de buscar la solución con **menor** coste posible. Pero para otra generación de solución inicial (por ejemplo, nuestra *versión 1* del generador de la solución inicial) este operador swap será muy poco útil, ya que no logramos añadir conexiones directas de un sensor a un centro de datos a medida que avanzamos en el espacio de soluciones, sino que solo cambiamos las conexiones ya existentes. Esto provoca que estos datos puedan ser fácilmente perdidos, por lo que el número de datos perdidos será muy alto.

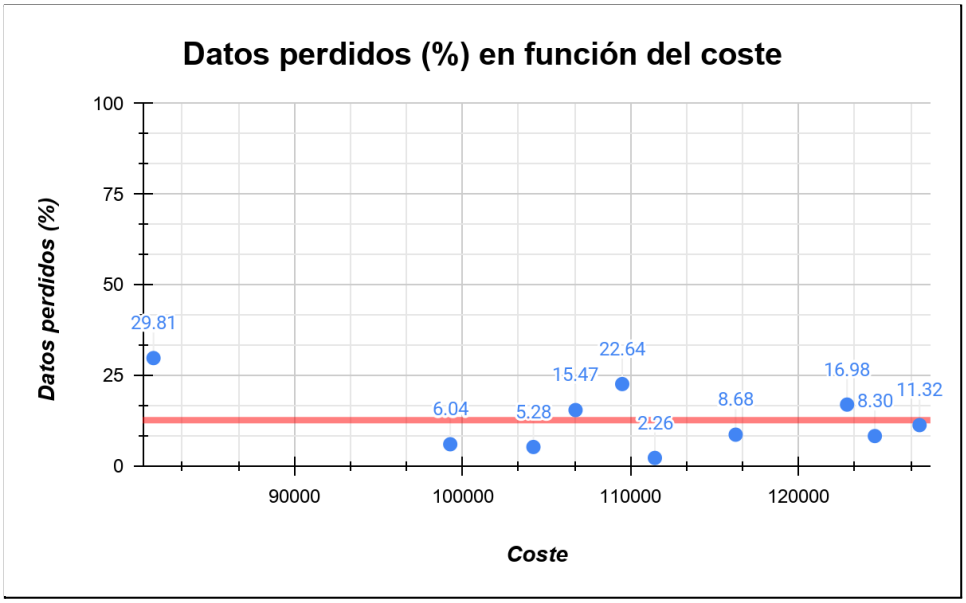
Con los resultados que hemos obtenido y las conclusiones a las cuales hemos llegado, podemos afirmar que el operador *changeConnexion* es **mejor** que *swapConnexion. Change* nos ofrece mucha más maniobrabilidad y **versatilidad** para encontrar una solución eficiente en cuanto al coste y los datos capturados, mientras que con *Swap* el algoritmo de búsqueda no mejorará en gran medida el número de datos capturados, solo podrá trabajar centrándose en reducir el coste.

9.2 *Experimento 2*: Estrategia de generación de solución inicial

En este experimento trataremos de decidir cuál de nuestros dos generadores de solución inicial nos ofrece un **mejor resultado** después de realizar la búsqueda local teniendo en cuenta la heurística escogida y el operador por el que nos hemos decantado en el primer experimento (*changeConnexion)*.

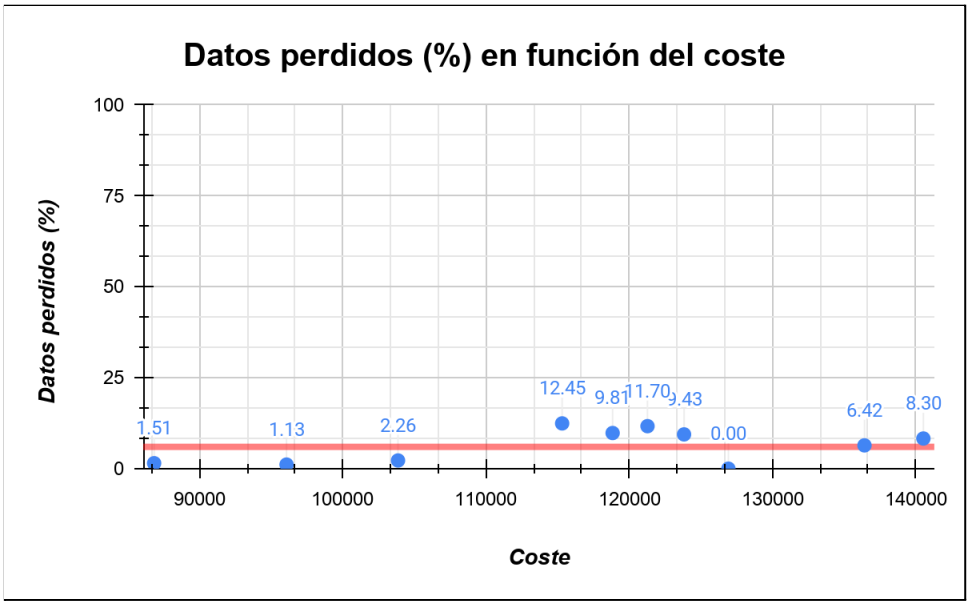
Observación	La *solución inicial 1* contiene más pérdidas de datos debido a su estructura.
Planteamiento	Comparar las dos soluciones iniciales, y encontrar cuál facilita la exploración y la optimización de la solución pedida.
Hipótesis	Creemos que la *solución inicial 2* presentará mejores resultados, ya que en su planteamiento ya se intenta optimizar uno de los parámetros de la heurística, los datos totales transmitidos.
Método	• Generamos una red con 4 centros y 100 sensores • Inicializamos la solución inicial utilizando la 2ª versión • Indicamos el algoritmo de búsqueda *Hill Climbing* • Indicamos la función sucesora para *Hill Climbing* • Generamos 10 semillas • Ejecutamos 1 experimento para cada semilla • Tomamos los valores resultantes para analizarlos • Repetimos el experimento para la versión de *solución inicial 2*

- **Solución Inicial 1:**



Vemos que con esta estrategia de generación la media de datos perdidos está alrededor de un ***12%***, y el coste se mueve generalmente entre *100.000* y *130.000*.

- **Solución inicial 2**



En este caso la media de datos perdidos se encuentra alrededor del rango del ***6%*** y el ***6,5%***, con valores de coste que se mueven entre *90.000* y *140.000*.

Comparando los resultados de ejecutar las dos búsquedas con las dos estrategias de generación inicial planteadas podemos observar que la media de datos perdidos de la segunda es claramente **inferior** a la de la primera estrategia. Esto puede ser debido a que la solución inicial que da la estrategia 2 ya de por sí es mejor que la 1 respecto a los datos perdidos, y en este caso el algoritmo tendrá que realizar menos iteraciones, lo que reduce la probabilidad de encontrar mínimos locales.

También podemos observar que la variabilidad del coste de la red es mayor en la segunda estrategia, pero esto puede ser influenciado considerablemente por la aleatoriedad de las **posiciones espaciales** de los centros de datos y los sensores. Además, esta diferencia no es suficientemente significativa como para decantarnos por una estrategia basándonos en el coste final.

Por lo tanto podemos concluir que la segunda estrategia nos dará mejores resultados que la estrategia 1.

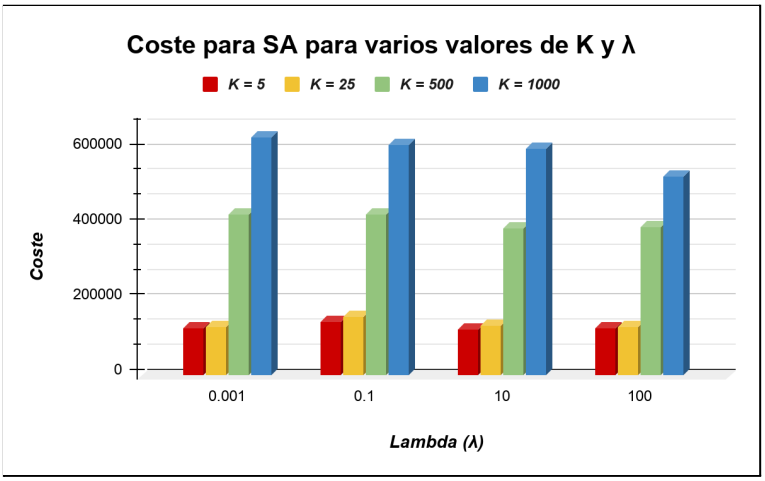
9.3 *Experimento 3*: Experimentación con *Simulated Annealing*

Para este experimento, exploraremos los resultados al problema utilizando el algoritmo de búsqueda local *Simulated Annealing*. Partiremos de una base, y analizaremos los cambios que se puedan dar para diferentes valores de ***K*** y ***lambda λ***. En concreto, ajustamos como referencia el número de *steps* a *10.000* y el valor de iteraciones por cada incremento de temperatura a *1.000*.

Observación	Creemos que los cambios de temperatura serán poco frecuentes debido a nuestra solución inicial.
Planteamiento	Comparar los valores de costes y datos perdidos para los diferentes parámetros de *K* y *lambda λ*.
Hipótesis	Para valores altos de *K* y/o valores bajos de *lambda λ* la exploración se verá reducida, y por lo tanto tendremos peores resultados.
Método	• Generamos una red con 4 centros y 100 sensores • Inicializamos la solución inicial utilizando la 2 versión • Indicamos el algoritmo de búsqueda *Simulated Annealing* • Indicamos la función sucesora para *Simulated Annealing* • Fijamos un valor de *K* y variamos *lambda λ* • Fijamos un valor de *lambda λ* y variamos *K*

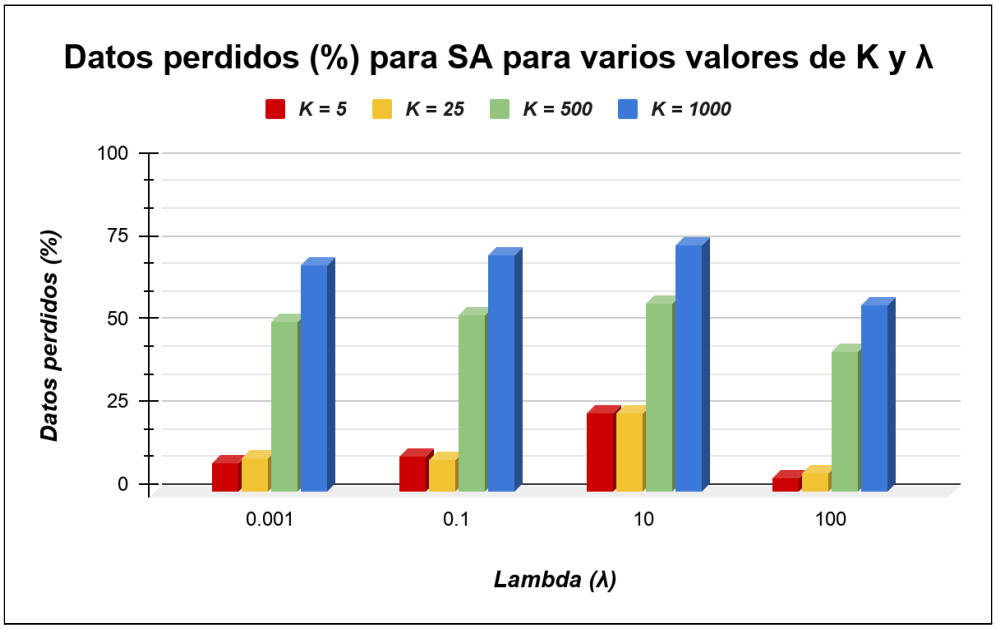
Para poder empezar a experimentar decidimos unos valores de *K* y *lambda λ* **aleatorios**. Una vez decididos estudiaremos cómo los costes y la cantidad de datos perdidos varían al fijar como un valor constante de *K* o *lambda λ* y hacer variar al otro.

Los valores de *lambda λ* con los que trabajaremos serán [*100, 10, 0.1, 0.001*] y los valores para *K* serán de [*1000, 500, 25, 5*], con estos valores intentamos **abastecer** un rango elevado de posibles valores iniciales, para analizar y entender el comportamiento de *Simulated Annealing*.



Con la gráfica anterior, observamos que nuestra hipótesis se mantiene al evaluar *K*, ya que la tendencia que muestra es que a medida que disminuimos su valor, también lo hace el coste, es decir, son **directamente proporcionales**. Nótese que variar *K* tiene una influencia muy elevada en el coste final, en comparación con *lambda* λ.

En cambio para los diferentes valores de *lambda* λ observamos que hay cierta tendencia a la baja respecto al coste. Pero la influencia que tiene sobre el coste final es **bastante menor** que la del parámetro *K*.

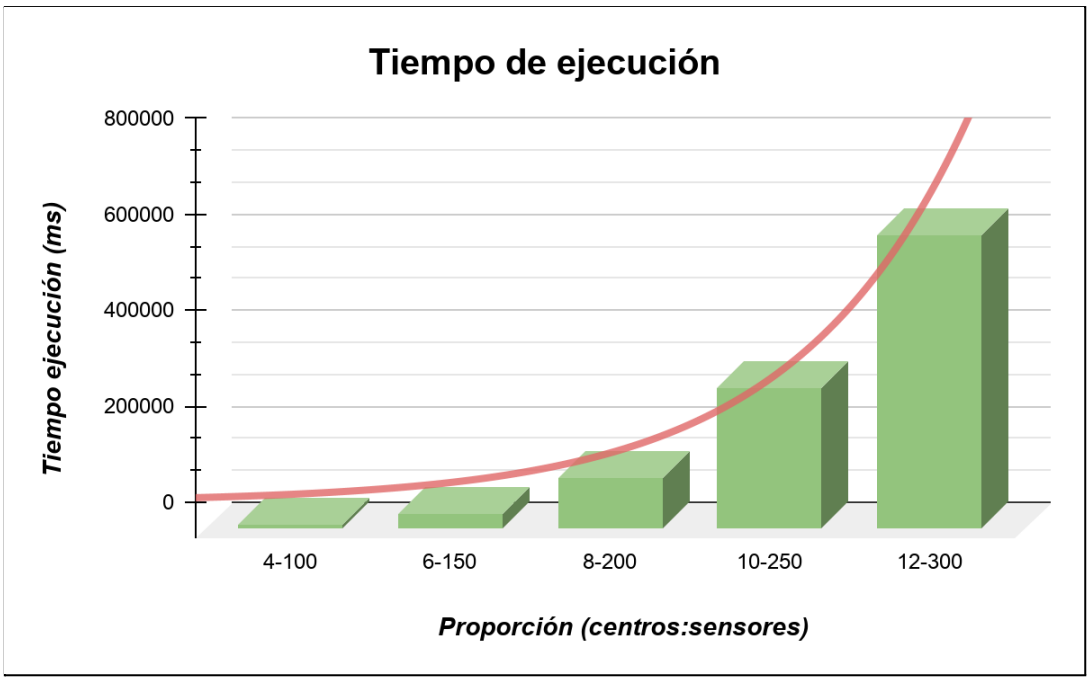


Respecto a los datos perdidos, encontramos que el comportamiento no sigue una tendencia decreciente similar a la del coste, de hecho encontramos un auge en la pérdida de datos al evaluar ***lambda = 10***. Aun así encontramos que la pérdida de datos por lo general disminuye con valores de *K* bajos y/o valores de *lambda* λ altos.

9.4 *Experimento 4*: Evolución del tiempo de ejecución con *Hill Climbing*

En este experimento estudiaremos la evolución del **tiempo de ejecución** para hallar la solución de una búsqueda para valores crecientes de los parámetros siguiendo la proporción inicial de *4:100* (por cada 4 sensores, 100 sensores), utilizando el algoritmo de *Hill Climbing*.

Observación	Según esta proporción se pueden capturar todos los datos de todos los sensores, ya que quedan 25 sensores por centro de datos.
Planteamiento	Incrementar el número de centros de datos de 2 en 2 y los sensores para que se siga manteniendo la proporción *4:100*.
Hipótesis	Al mantenerse la proporción constante no variará en exceso el tiempo de búsqueda.
Método	• Generar una red inicial con 100 sensores y 4 centros de datos, con la misma heurística y estrategia de generación de solución inicial escogidos en los experimentos anteriores • Ejecutar el algoritmo 10 veces y calcular el promedio • Aumentar los valores de los sensores y los centros de datos manteniendo la proporción *4:100* (6-150, 6-200, etc.) • Ejecutar el algoritmo 10 veces para cada cambio de valores hasta que veamos una tendencia



En esta gráfica podemos ver la evolución del tiempo de ejecución en milisegundos de la búsqueda local dependiendo del crecimiento del número de sensores y de centros de datos, siempre manteniendo la **misma proporción** de *4:100*.

Observamos que el aumento del tiempo de ejecución crece exponencialmente a medida que aumentamos los sensores y centros de datos. Este resultado **contradice** nuestra hipótesis inicial, en la que creíamos que el tiempo no variaría en exceso. Después de pensarlo, creemos que esto es debido a que al aumentar los valores de estas variables crece **exponencialmente** el **espacio de soluciones**, por lo que es lógico que el algoritmo de búsqueda tarde más tiempo, ya que tiene que generar muchos más sucesores por la naturaleza del *Hill Climbing*.

9.5 *Experimento 5*: Centros de datos utilizados

En este experimento evaluamos las proporciones de sensores conectados a centros de datos. Nuestra solución inicial, como hemos comentado anteriormente, inicializa la red conectando directamente todos los sensores que pueda a los centros de datos, por lo tanto siempre empezaremos con estados con una proporción de sensores conectados a centros muy **elevada**. Dado que no se está considerando las distancias en esta solución inicial, esperamos ver que estas proporciones **decrezcan** significativamente a lo largo de la búsqueda local.

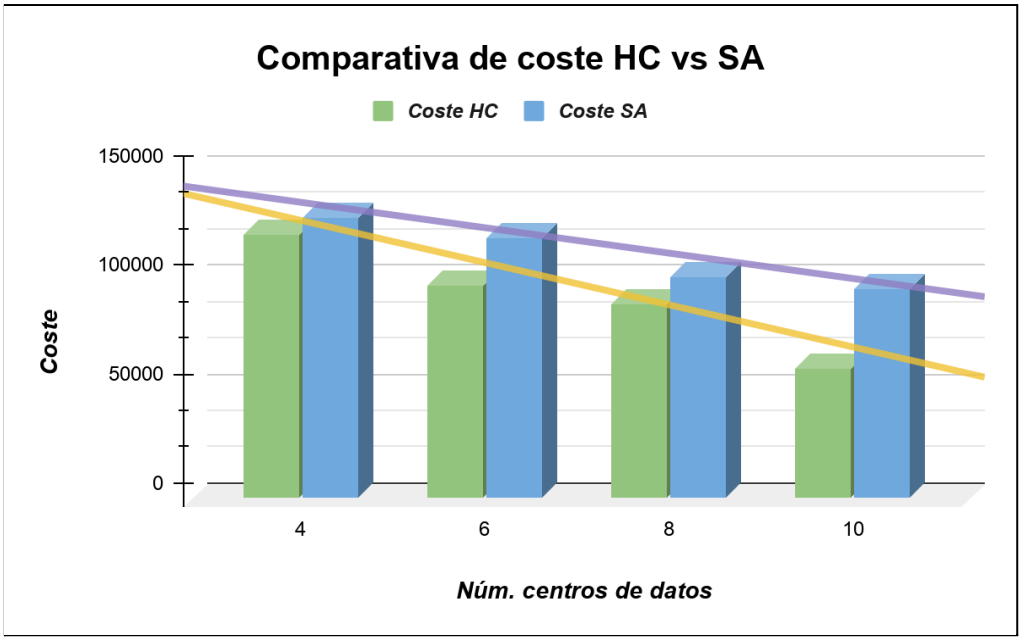
Observación	Nuestra solución inicial y heurística puede influir en la cantidad de centros de datos usados y en la proporción de sensores conectados a centros.
Planteamiento	Evaluar el número de sensores conectados a los centros.
Hipótesis	Esperamos ver que se prioricen ciertas conexiones a sensores, en vez de a centros, pero creemos que será poco probable encontrar centros sin sensores conectados.
Método	● Modificar el código para que se muestre el número de conexiones a los centros ● Repetir los experimentos anteriores ● Evaluar los valores extraídos

En la mayoría de las ejecuciones de los experimentos realizados, encontramos que **nunca** llegan a quedarse centros de datos inutilizados. Observamos que sí que se reducen considerablemente estas conexiones respecto a nuestra solución inicial, pero que estas nunca llegan a dejar aislado algún centro de datos. El número de **sensores conectados** a cada centro **fluctúa entre 4 y 19** y la proporción total de sensores conectados a centros respecto a la cantidad de sensores es de entre *30* y *40%***.**

9.6 *Experimento 6*: Dependencia del coste de una red en función del número de centros de datos

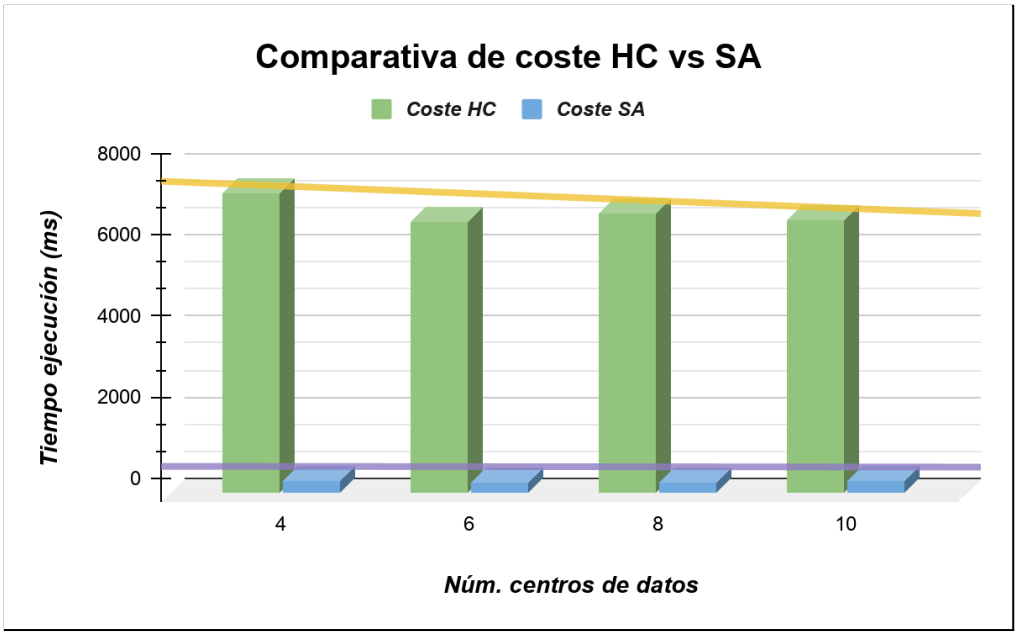
A continuación analizaremos qué influencia tiene el añadir centros de datos a una red con un número de sensores fijo en términos de **coste de la red** y el **coste temporal** (es decir, tiempo de ejecución), suponiendo que los centros de datos no sean costosos. Mediremos estos datos con los algoritmos *Hill Climbing* y *Simulated Annealing* así a la vez haciendo una comparativa entre los dos algoritmos.

Observación	Los centros de datos no tienen conexiones de salida con otros centros o sensores, por lo que no tienen un coste.
Planteamiento	Estudiar si el coste general de una red puede crecer o reducirse en función de que se aumenten los centros de datos de la red.
Hipótesis	Al ir creciendo el número de centros de datos, los sensores tendrán más posibilidades de conectarse con centros que estén más cercanos a ellos, por lo que el coste de la red puede verse reducido. Respecto al coste temporal, este crecerá, ya que el espacio de soluciones aumenta.
Método	● Comenzar con una red *4:100*, el generador de solución inicial escogido en el primer experimento y misma heurística ● Realizar 10 pruebas utilizando *Hill Climbing* con 4 centros de datos, a continuación con 6, 8 y 10 centros ● Anotar los resultados de los costes de red y los costes temporales ● Realizar el mismo experimento esta vez empleando *Simulated Annealing* (K = 5 y λ = 0.001). ● Comparar resultados



En esta primera gráfica de barras tenemos la evolución de **valores medios** del coste de una red cualquiera con proporciones *4:100*, *6:100*, *8:100* y *10:100*, utilizando los algoritmos de búsqueda *Hill Climbing* (barras y línea de tendencia amarilla) y *Simulated Annealing* (barras y línea de tendencia morado).

Lo importante a destacar en esta gráfica es el **decrecimiento** del valor del coste de la red (en ambos algoritmos) a medida que aumentan los centros de datos, punto que **concuerda** con nuestra hipótesis inicial.



Con esta segunda gráfica mostramos la evolución del **coste temporal** (en milisegundos) que tiene la búsqueda local con los algoritmos antes mencionados, también con los mismos valores de número de sensores y centros de datos utilizados en la primera parte del experimento.

Nótese que aquí es más que evidente la diferencia de tiempos medios entre HC y SA. Observamos que los costes temporales medios **no varían en exceso** cuando aumentan los centros de datos, sino que se mantienen bastante constantes. Como mucho se observa que utilizando HC, la tendencia es decreciente. Sin embargo, estos resultados son completamente **opuestos** a los que esperábamos (nuestra hipótesis decía que los tiempos aumentarían al hacer crecer el número de centros de datos). Esto se puede deber a que, al haber más centros de datos, los sensores tienen más opciones de tener conexiones directas con estos, por lo que se llega a una solución "*óptima*" en menos tiempo.

9.7 *Experimento 7*: Ponderación de la función heurística

En este experimento modificamos la estructura de la red reduciendo el número de centros de datos a *2*. Además, añadimos un factor en el cálculo de la función heurística para poder ponderar la cantidad de información enviada.

Observación	Reducir los centros de datos a 2 originará una red bastante más densa que las estudiadas anteriormente.
Planteamiento	Evaluar mediante un factor ponderador los tiempos de ejecución para la nueva red.
Hipótesis	Creemos que los tiempos de ejecución podrían aumentar debido a una probabilidad mayor de crear ciclos.
Método	● Modificar la función heurística ● Crear la red con 2 centros y 100 sensores ● Ejecutar HillClimbing repetidamente 10 iteraciones cada vez con *seeds* diferentes ● Ir variando la ponderación ● Evaluar resultados

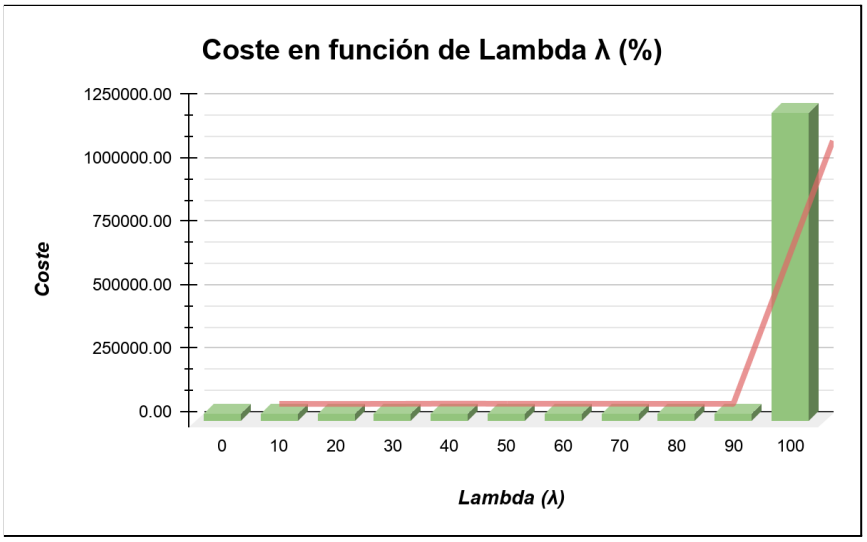
- **¿Se ha reducido el tiempo para hallar una solución?**

  No, de hecho el tiempo de ejecución ha aumentado.

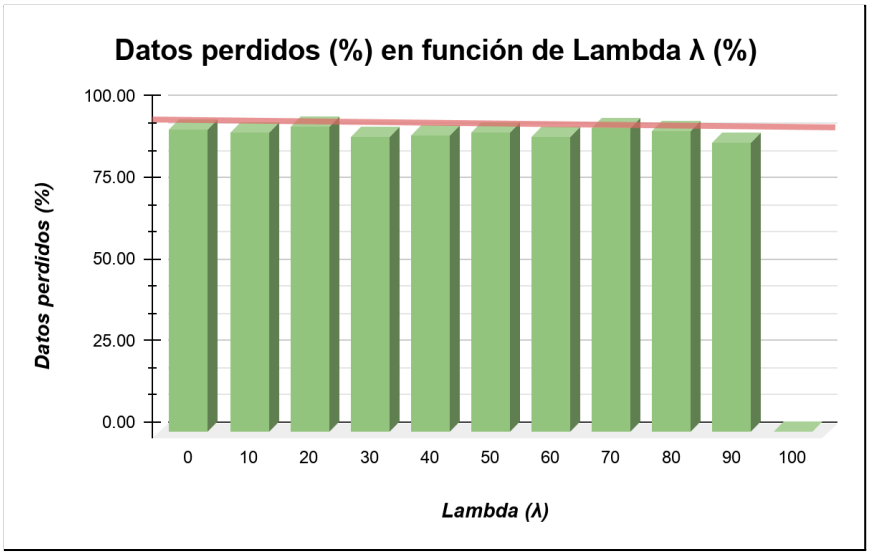
- **Si usamos una ponderación para el factor que mide la cantidad de información enviada igual que en los primeros escenarios ¿En qué proporción aumenta el coste de la red?**

  Los tiempos de ejecución aumentan con respecto a la red con 4 centros de datos. Aumentan en una media de aproximadamente de *1527,2* milisegundos**.**

- **¿Cómo cambia el coste de la red en función de la ponderación que se da al envío de los datos? ¿Hay una ponderación a partir de la que el coste de la red ya no aumenta?**

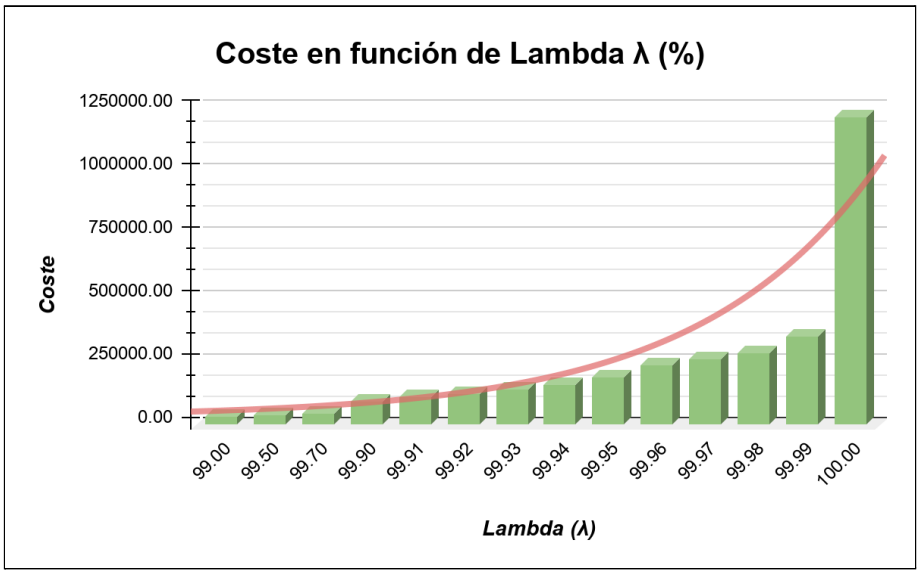


En la gráfica anterior observamos que para el coste y nuestro parámetro de ponderación *lambda λ* se da una transición **muy pronunciada** entre el valor **90** y **100**. Nuestra función heurística utiliza tanto el coste de las conexiones, que contempla las distancias, como la cantidad de datos que se transmiten. Por lo tanto este comportamiento es de esperar, ya que estamos combinando dos parámetros considerablemente diferentes. También hace falta considerar que sus **órdenes de magnitud** no tienen por qué ser similares, y este hecho este mismo motivo podría traducirse en la transición tan **abrupta** que contemplamos.

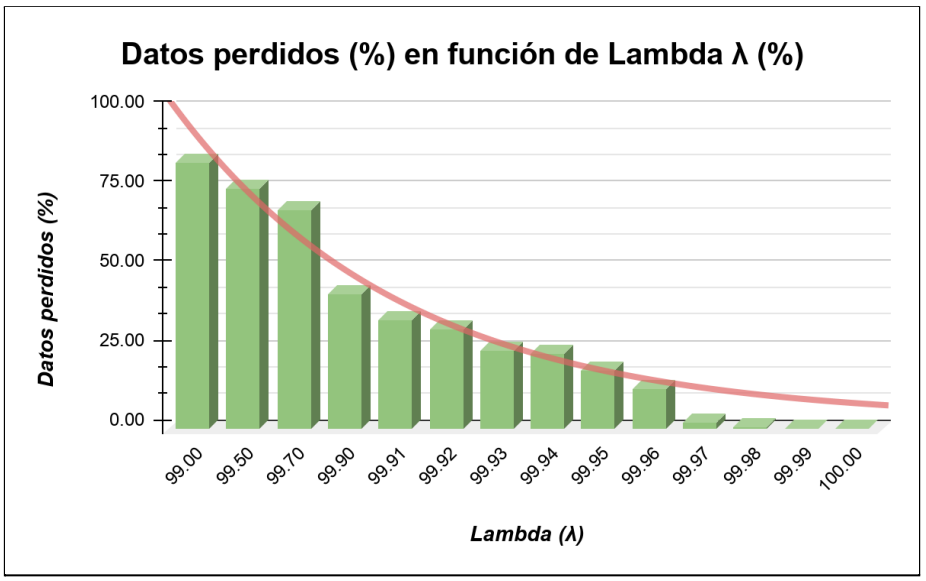


Para el caso de los datos perdidos, observamos el mismo efecto, se da una **transición** muy **accidentada** entre los valores de *90* y *100*.

Para indagar y estudiar con más detalle este efecto de transición, analizamos el comportamiento del coste y de los datos perdidos escogiendo **valores más precisos** en el intervalo de [*90, 100*] y volviendo a repetir el experimento.



Analizando la transición con más detalle, vemos que el coste **aumenta considerablemente** al llegar a valores de *lambda λ* del orden de *99,9* en contraposición con las anteriores gráficas, donde hay al menos cierta progresión. Concretamente, si nos fijamos en la línea de tendencia, el **crecimiento** es **exponencial** en función de *lambda λ*.



En cambio en esta gráfica, con respecto al coste (de la anterior gráfica) vemos que los datos perdidos disminuyen de manera **exponencial** pero más **moderada** y progresiva. Por lo tanto concluimos que al ponderar más los datos transmitidos, el coste aumenta y este parece no tender a estabilizarse alrededor de un valor en concreto.

10. Conclusiones

Finalmente, después de realizar la práctica, como principal conclusión podemos extraer es que el problema es de una **mayor complejidad** de lo que esperábamos desde un principio. Y, gracias a ello, para la realización de esta misma tuvimos debates y **discusiones** constructivas **enriquecedoras** sobre muchos puntos de la práctica, además de consultas con el docente de la asignatura. Por consiguiente, al avanzar en la práctica, se nos ocurrían mejoras para implementar sobre los puntos anteriores ya hechos, pero que no llegamos a implementar (la gran mayoría) por escasez de tiempo.

En cuanto a los algoritmos vistos, lo que **nos sorprendió** fue las *buenas* soluciones del problema que generaba el algoritmo de ***Simulated Annealing*** con una buena parametrización. Nos era inconcebible que *SA*, que con tan poco uso de memoria y un tiempo de ejecución mínimo, podría generar soluciones que en muchos casos eran igual de *buenas* que las soluciones generadas *Hill Climbing* habiendo explorado muchísimos más sucesores. Cabe recalcar que ninguno de nosotros tenía mucho conocimiento sobre este campo de algoritmos de búsqueda local de la inteligencia artificial, y que *Simulated Annealing* y *Hill Climbing* son los primeros que hemos aprendido a utilizar.

En última instancia, podemos afirmar que el desempeño de esta práctica (planteamiento del problema, implementación y experimentación con las diferentes variables) nos ha dado una **visión** mucho más **clara** sobre este ámbito de la *Inteligencia Artificial.* Nos asombra que a partir de un problema, que a primera vista no parece ser de *IA*, se puede solucionar con *IA* planteando bien el estado de problema, un/os operador/es, y una buena heurística se pueda solucionar con un algoritmo, y de manera relativamente "*sencilla*".

Apéndice

A Proyecto de Innovación

A.1 Descripción del tema

Para el trabajo de innovación escogimos el **GPT-3** (*Generative Pre-trained Transformer 3*), se trata de un modelo de lenguaje autorregresivo publicado en 2020 por *OpenAI*. GPT-3 es un modelo que forma parte del campo de estudio del procesamiento de lenguaje natural.

A.2 Reparto del trabajo

Nos hemos repartido el trabajo siguiendo los apartados del enunciado de manera equitativa, por lo tanto la búsqueda de información previa ha dependido del apartado que nos asignamos. También dado que los apartados están muy correlacionados entre sí, en el caso que alguno de nosotros encontrará información útil nos la compartimos mutuamente.

Empezamos reuniéndonos de manera online y poniéndonos *deadlines* aproximadamente cada semana, esta manera de trabajar no duró mucho, ya que a medida que avanzó el curso teníamos menos disponibilidad por la carga de trabajo de otras asignaturas y adoptamos una dinámica de trabajo más individualizada y por lo tanto más flexible.

A.3 Referencias

Paper original de la publicación de GPT-3:

https://arxiv.org/pdf/2005.14165.pdf

Idea general y descripción de GPT-3:

https://www.xataka.com/robotica-e-ia/gpt-3-nuevo-modelo-lenguaje-openai-capaz-programar-disenar-conversar-politica-economia

Entrevista a tres expertos técnicas de IA GPT-3:

https://www.xataka.com/robotica-e-ia/que-tres-expertos-que-trabajan-inteligencia-artificial-opinan-gpt-3

Aprendizaje profundo:

https://www.netapp.com/es/artificial-intelligence/what-is-deep-learning/#:~:text=A%20diferencia%20de%20los%20algoritmos,lo%20que%20es%20lo%20mismo%2C

Más aprendizaje profundo:

https://www.bbvaopenmind.com/tecnologia/mundo-digital/que-es-el-aprendijaze-profundo/

Técnica de Few-Shot learning:

https://research.aimultiple.com/few-shot-learning/

Task-agnostic Exploration in Reinforcement Learning:

https://arxiv.org/pdf/2006.09497v1.pdf

Lambda labs coste entrenamiento GPT-3:

https://lambdalabs.com/blog/demystifying-gpt-3/

Priming:

https://innovacioneducativa.wordpress.com/2013/12/16/que-es-priming-educacional/

Info general de GPT-3, API:

https://www.technologyreview.com/2020/07/20/1005454/openai-machine-learning-language-generator-gpt-3-nlp/

Estudio de la toxicidad en NLM's:

https://www.aclweb.org/anthology/2020.findings-emnlp.301.pdf

Sesgo racista en GPT-3:

https://arxiv.org/pdf/2101.05783v1.pdf

Dot CSV : La Siguiente Gran Revolución: NLP (Procesamiento del Lenguaje Natural):

https://www.youtube.com/watch?v=cTQiN9dewIg

Computerphile **-** GPT3: An Even Bigger Language Model:

https://www.youtube.com/watch?v=8yVOC4ciXc&t=440s

Computerphile - Unicorn AI:

https://www.youtube.com/watch?v=89A4jGvaaKk

A.4 Dificultades

Hemos encontrado que hay un desequilibrio bastante importante entre la cantidad y calidad de artículos de GPT-3 entre español y inglés. También hemos encontrado cierta dificultad al familiarizarnos con ciertos términos técnicos que desconocemos pero que son ampliamente usados en el procesamiento del lenguaje natural.