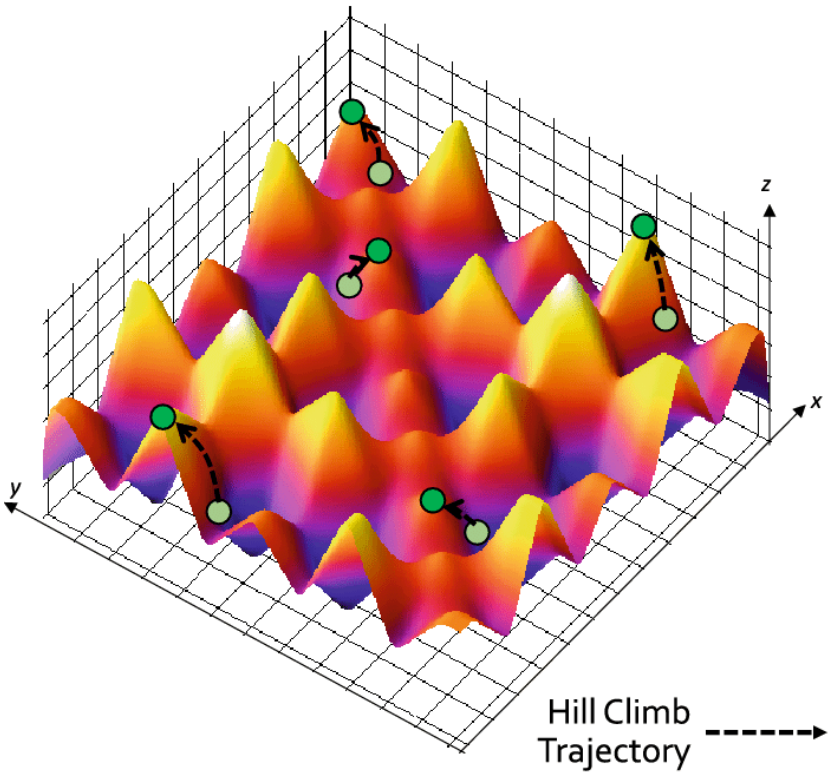
Estudio de paralelización en *Hill Climbing* y parametrización en *Simulated Annealing*



Índice

1. Hill Climbing..... 3
1.1. Introducción..... 3
1.2. Experimentos..... 5
2. Simulated Annealing..... 8
2.1. Introducción..... 8
2.2. Experimentos..... 10
2.2.1. Estudiar los resultados en función de la cantidad de iteraciones..... 11
2.2.2. Estudiar los resultados en función del parámetro stiter..... 13
2.2.3. Estudiar los resultados en función de los parámetros K y λ..... 15

1. Hill Climbing

1.1. Introducción

Hill Climbing es un algoritmo de búsqueda local basado en inteligencia artificial. La práctica de ordenación de productos en una estantería se puede reducir al problema del viajante de comercio, el cual no tiene solución mediante un algoritmo clásico en tiempo polinómico.

Además definir una heurística admisible y suficientemente informativa para un algoritmo de búsqueda heurística como A* parece inviable, sin embargo definir la calidad de una solución es muy sencillo. Se define como la suma acumulada de sinergias en la solución:

$$H = \Sigma_{p \in P} \Sigma_{q} sinergia(p, q) : p \sim q$$

Donde p es un producto perteneciente a la lista de productos P de la solución y q es un producto adyacente (según la simetría descrita en el enunciado). Adicionalmente $sinergia(p, q)$ es la función que devuelve la sinergia entre dos productos almacenada en la matriz de adyacencias.

Por lo tanto, en cuanto a técnicas de IA clásicas, las opciones viables son búsqueda local y algoritmos genéticos. Sin embargo codificar los estados como cadenas de bits u otro formato adecuado para un algoritmo genético plantea un reto considerable. Teniendo esto en mente la opción más adecuada son *Hill Climbing* y *Simulated Annealing*.

Hill Climbing requiere una solución inicial, una serie de operadores y una función heurística. Hemos optado por generar la solución inicial al azar, permitiendo así la posible evasión de óptimos locales al ejecutar el algoritmo varias veces. El único operador disponible *intercambiar_posiciones*, como su nombre indica, intercambia dos posiciones en la solución (ya sean productos o huecos), es suficiente para explorar todo el espacio de soluciones y tiene una ramificación de $O(n^2)$. Este factor de ramificación es suficientemente bajo para distribuciones que no sean masivas. La función heurística a maximizar es, obviamente, la suma acumulada de sinergias en la solución que define la calidad en la estantería.

Se podrían haber planteado soluciones iniciales parciales basadas en algoritmos clásicos de aproximación a TSP, pero aunque su coste computacional es polinómico, es lo

suficientemente grande como para contrarrestar el objetivo de la búsqueda local, una solución buena de forma rápida. Además quizás se podrían haber introducido otros operadores, pero en este caso pensamos que solo ralentizarían al algoritmo.

Para aprovechar la naturaleza de la solución inicial que potencialmente evade óptimos locales sin ralentizar el algoritmo se plantea la posibilidad de **paralelizar el código**. Se realizaría con 8 threads, una medida estándar a día de hoy. De esta forma se ejecutaría el algoritmo 8 veces simultáneamente, cada instancia con una solución inicial diferente, y se escribiría la que mejor calidad tuviese en la interfaz. En otras palabras sería un paralelismo de tipo *Weak scaling* ya que aumentamos el tamaño del problema en función a la cantidad de threads.

1.2. Experimentos

Realizaremos pruebas utilizando una distribución de 30 productos donde hemos identificado varios máximos locales, además hemos generado 100 semillas aleatorias almacenadas de forma estática para garantizar las mismas soluciones iniciales en las ejecuciones secuenciales que las paralelas, par a par.

Observación	La cantidad de threads ejecutando *Hill Climbing* simultáneamente influye en el resultado.
Planteamiento	Ejecutaremos el código de forma secuencial y en paralelo con 8 threads para observar las diferencias en tiempo y calidad.
Hipótesis	Una cantidad de threads mayor permitirá al algoritmo empezar en diferentes puntos mejorando la calidad y al ser ejecutado en paralelo no ralentizará su ejecución (H0). O la solución secuencial dará mejores resultados (hipótesis alternativa).
Método	- Utilizamos 100 semillas aleatorias, una para cada ejecución del H.C. - Ejecutaremos 1 experimento para cada semilla con cantidad de threads determinados (1 o 8). - Repetimos el paso anterior con todas las otras cantidades utilizando **las mismas semillas**. - Utilizaremos el algoritmo de *Hill Climbing*. Mediremos y representaremos en un gráfico *boxplot* con la calidad de los resultados para comparar las posibles valores y también representaremos su tiempo de ejecución.

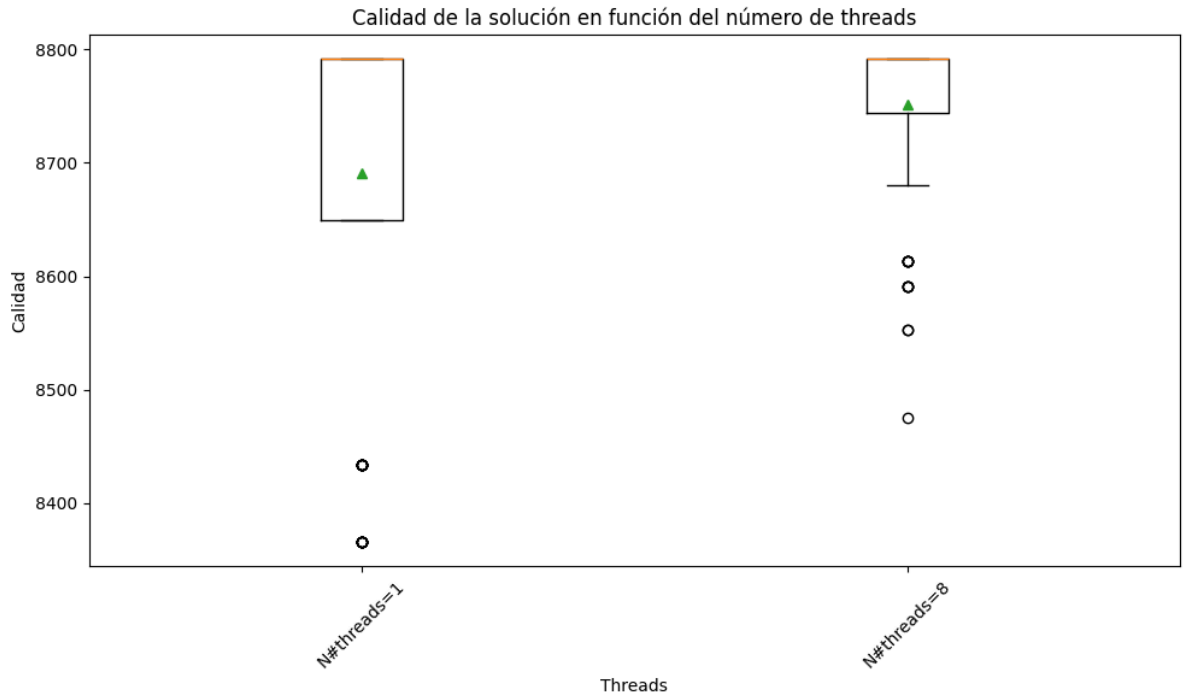


Figura 1: Boxplot calidad vs número de threads

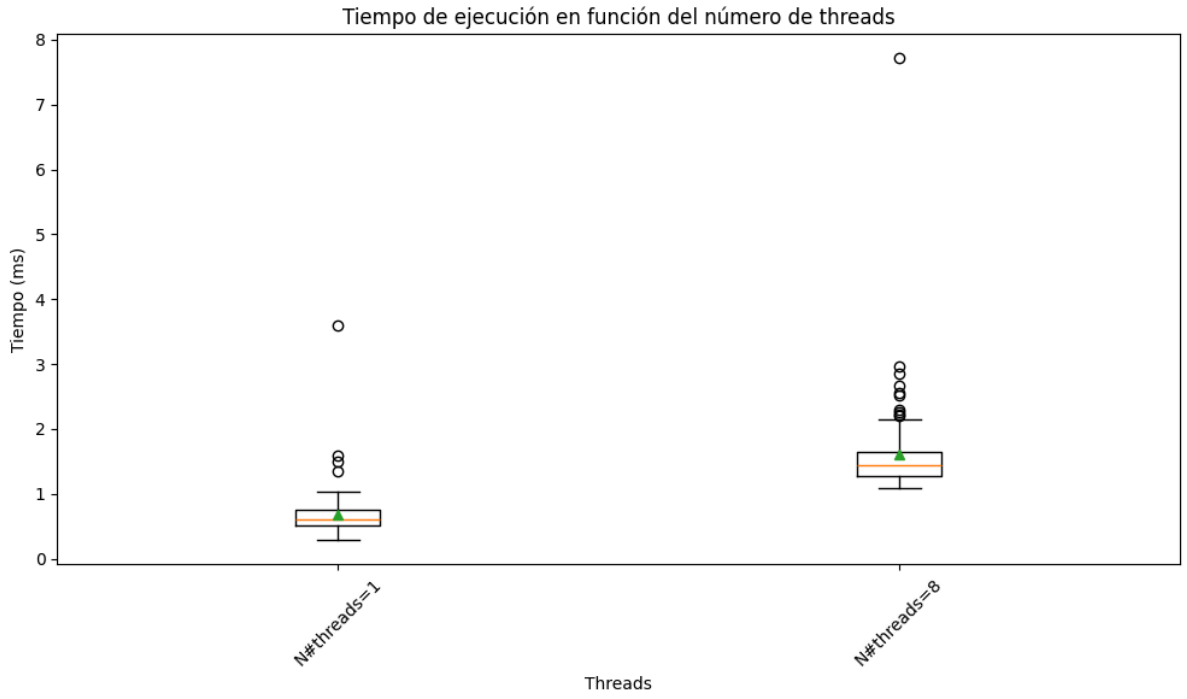


Figura 2: Boxplot tiempo vs número de threads

Conclusiones: La hipótesis inicial es correcta respecto a la calidad de la solución. Aunque ambos obtienen la misma calidad mediana, la media es más alta utilizando 8 threads. Esto nos indica que la mayoría de ejecuciones encuentra el óptimo global, aunque algunas no lo hacen. Observamos que es más común quedarse atascado en un máximo local utilizando un solo thread, por esta razón parece que es adecuado implementar paralelización.

Sin embargo el tiempo de ejecución es más o menos el doble al paralelizar, esto es probablemente debido a *overheads* de sincronización y creación de tareas para los threads. Estimamos que esta diferencia debe ser negligible en distribuciones mayores. De todas formas los tiempos de ejecución no superan los 8ms, lo cual es sorprendentemente rápido teniendo en cuenta el tamaño del problema.

2. Simulated Annealing

2.1. Introducción

Tras haber implementado *Hill Climbing* queda clara su vulnerabilidad para quedarse atascado en óptimos locales. Para evitar esto se ha diseñado una versión en *Simulated Annealing* con unas características algo diferentes. SA es un algoritmo de inteligencia artificial basado en metalurgia, un proceso llamado *annealing* o templado en el que se calienta un metal rápidamente hasta una cierta temperatura y más tarde se enfría.

Esta versión no prueba todas las acciones posibles de nuestro operador *intercambiar_productos* por cada estado que atraviesa y selecciona el vecino de mayor calidad o como se denomina en S.A., energía, si no que accede al primero que mejore el estado actual sin comprobar si otra operación podría mejorarlo más.

También accede a otro estado, aunque tenga peor calidad, dada una cierta probabilidad dependiendo de la **temperatura**. Es decir que intentaremos evitar óptimos locales atravesando mesetas o pequeñas irregularidades en la heurística (suma acumulada de sinergias) aceptando de vez en cuando estados peores ya que cuanto más avance el algoritmo menos probabilidad existe de que acepte estos estados inferiores.

La probabilidad de que una acción lleve a un cambio de estado es la siguiente:

$$P(estado) = e^{\frac{\Delta E}{f(T)}}$$

Donde ΔE es la diferencia de energía entre el estado candidato y el original y $f(T)$ la función de temperatura:

$$f(T) = k \cdot e^{-\lambda T}$$

Donde k y λ son parámetros que permiten ajustar la probabilidad de aceptación sin afectar directamente a la temperatura.
Además se reduce la temperatura cada cierto número de iteraciones llamado ***stiter***.
Esta estructura lleva a independencia entre el parámetro *stiter* y los parámetros de la función de temperatura. En otras palabras es posible controlar la frecuencia de enfriamiento (*stiter*),

su intensidad (λ) y la magnitud de la temperatura (k) y también se puede estudiar el impacto en la calidad de *stiter* por separado a λ y k.
Ya que estos parámetros permiten controlar el enfriamiento y la exploración inicial libremente se ha fijado una temperatura inicial constante arbitraria (1000) y un factor de enfriamiento constante (0.03) que se aplica de forma multiplicativa, es decir:

$$T_n = T_{n-1} \cdot (1 - coolingRate)$$

Donde T_0 es 1000 y $coolingRate$ es 0.03. Estos valores fueron escogidos debido a una práctica de la asignatura de inteligencia artificial, aunque debido al control sobre el enfriamiento y la exploración inicial que otorgan *stiter*, k y λ podrían haber sido cualquier otro par de valores.

Adicionalmente se ha establecido un límite de iteraciones máximo para el algoritmo, de forma que tenga una fita de tiempo constante y pueda resultar siempre rápido o con suerte instantáneo para el usuario. Esto significa que quizás podría no dar tan buenos resultados en distribuciones enormes, pero podríamos denominarlo en la aplicación como algoritmo *ultra rápido* ya que jamás superará cierta cantidad de tiempo de ejecución.

Además simplificará nuestro proceso de experimentación. Para escoger una cantidad de pasos que aporte calidad suficiente en la mayoría de casos hemos realizado todas las pruebas siguientes con distribuciones de 30 productos. Estimamos que es una cantidad bastante grande que debería ser la media en estanterías reales, siempre y cuando el producto se generalice lo suficiente (es decir que el queso havarti y el gouda se seleccionen ambos como queso).

En caso de realizar este proyecto para una empresa real probablemente sería adecuado calcular la cantidad de iteraciones con respecto al tamaño de la entrada, teniendo en cuenta también la evolución de temperatura para que llegue a ser lo suficientemente baja el las últimas iteraciones. Sin embargo esta versión simplificada otorga resultados satisfacibles, comunmente superiores a *Hill Climbing*.

2.2. Experimentos

Ya que la cantidad de iteraciones (*steps*), *stiter* y k y λ son independientes (k y λ como pareja ya que son dependientes) procederemos con el estudio fijando unos valores para cada parámetro iniciales y estudiando el rendimiento del resto. En cada experimento actualizaremos el valor del parámetro (o pareja de parámetros) estudiado al que otorgue mejores resultados. Comenzaremos con:

- *steps*: primer parámetro estudiado
- *stiter*: *steps* / 10
- k: 5
- λ: 0.001

Además estos son los conjuntos de valores que probaremos para cada parámetro:

- *steps:* {100, 1000, 10000, 25000, 50000}
- *stiter*: {1, 2, 5, 10, 100, 250, 2500, 5000}
- k: {1, 5, 25, 100}
- λ: {0.001, 0.01, 0.1, 0.5}

En caso de realizar un estudio para optimizar estos parámetros para una empresa o un entorno exigible sería adecuado seleccionar una cantidad más amplia de posibles valores, realizar pruebas dicotómicas al encontrar los dos mejores en los conjuntos de pruebas y repetir este proceso hasta encontrar los mejores parámetros posibles con cierta precisión. Sin embargo estos conjuntos más limitados nos permiten realizar el estudio sin esperar horas para los resultados y son suficientes para un propósito didáctico.

2.2.1. Estudiar los resultados en función de la cantidad de iteraciones

Observación	La cantidad de pasos que toma el simulated annealing influye en el resultado.
Planteamiento	Escogemos un conjunto de posibles cantidades de pasos (*steps*) para observar las diferencias en la calidad de las soluciones de las combinaciones.
Hipótesis	Una cantidad de pesos mayor permitirá al algoritmo explorar de forma más exhaustiva el espacio de soluciones y por lo tanto tendrá más posibilidades de hallar el óptimo global (H0). O otra cantidad dará mejores resultados (hipótesis alternativa).
Método	- Utilizamos 100 semillas aleatorias, una para cada ejecución del S.A.. - Ejecutaremos 1 experimento para cada semilla con cantidad de pasos determinada. - Repetimos el paso anterior con todas las otras cantidades utilizando **las mismas semillas**. - Utilizaremos el algoritmo de *Simulated Annealing*. Mediremos y representaremos en un gráfico *boxplot* con la calidad de los resultados para comparar las posibles valores y también representaremos su tiempo de ejecución.

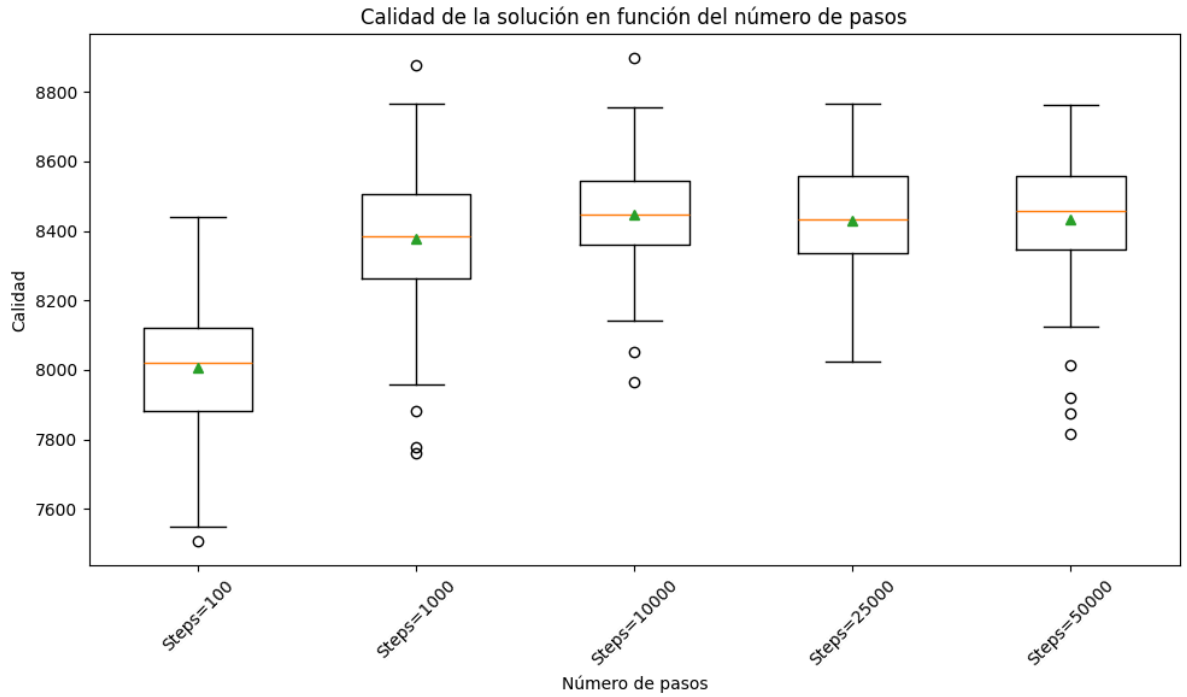


Figura 1: Boxplot Calidad vs número de pasos

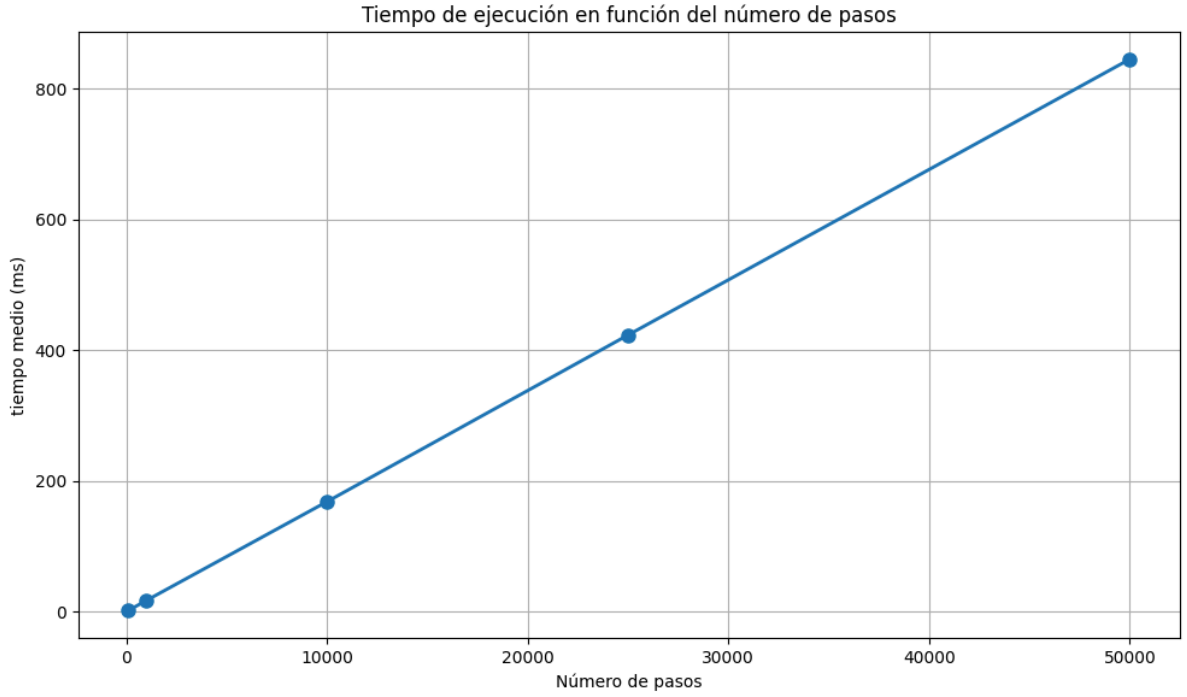


Figura 2: gráfico de línea

Conclusión: Al parecer la hipótesis inicial H0 era incorrecta, a partir de 10000 pasos la calidad del algoritmo parece no mejorar, de hecho los datos indican una media ligeramente superior al ejecutar con 10000 *steps* en comparación con 25000 y 50000. Además el tiempo de ejecución crece linealmente con la cantidad de pasos, como era de esperar, por lo que de ahora en adelante seguiremos con los experimentos con 10000 pasos, lo cual debería asegurar un tiempo de ejecución alrededor de los 0.2s en todos los casos.

2.2.2. Estudiar los resultados en función del parámetro stiter.

Observación	La cantidad de pasos que toma el simulated annealing antes de aplicar un cambio en la temperatura influye en el resultado.
Planteamiento	Escogemos un conjunto de posibles cantidades de pasos (*stiter*) para observar las diferencias en la calidad de las soluciones de las combinaciones.
Hipótesis	Una mayor cantidad de pasos por decremento en la temperatura permitirá al algoritmo explorar mejor el espacio de soluciones y por lo tanto tener más posibilidades de encontrar el óptimo global (H0). O otra cantidad dará mejores resultados (hipótesis alternativa).
Método	- Utilizamos 100 semillas aleatorias, una para cada ejecución del S.A.. - Ejecutaremos 1 experimento para cada semilla con cantidad de pasos por decremento determinada. - Repetimos el paso anterior con todas las otras cantidades utilizando **las mismas semillas**. - Utilizaremos el algoritmo de *Simulated Annealing*. Mediremos y representaremos en un gráfico *boxplot* con los resultados para comparar los posibles valores.

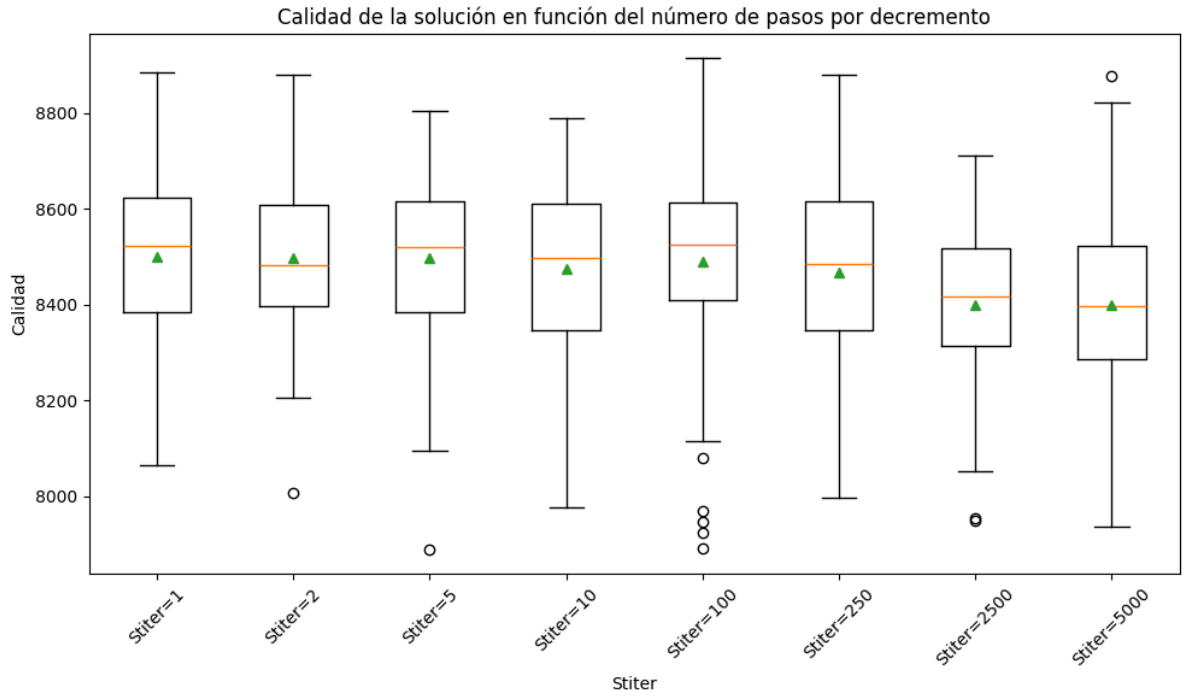


Figura 2.2.1: Distribución del Coste en función de K y λ

Conclusión: La hipótesis inicial era completamente errónea, parece que en nuestro caso el algoritmo se beneficia de cambios frecuentes en la temperatura, decrementarla cada iteración ha proporcionado los mejores resultados con una media ligeramente superior a los otros. Aunque sorprendentemente no parece haber bastante diferencia hasta superar las 250 iteraciones por decremento. A partir de ahora utilizaremos un *stiter* de 1.

2.2.3. Estudiar los resultados en función de los parámetros K y λ.

Observación	Los valores de K y λ influyen en el resultado final.
Planteamiento	Escogemos un conjunto de Ks y de λs para observar las diferencias en la calidad de las soluciones de las combinaciones.
Hipótesis	Una λ pequeña y una K grande permitirán una búsqueda más amplia. Al tener un rango de búsqueda de 10000 pasos y un problema de tamaño relativamente pequeño la combinación de λ = 0.001 y K = 100 debería dar el mejor resultado (H0). O otra combinación tendrá mejor calidad.
Método	- Elegiremos 100 semillas aleatorias. - Ejecutaremos 1 experimento para cada semilla con una combinación de K y λ. - Repetimos el paso anterior con todas las otras combinaciones utilizando **las mismas semillas**. - Utilizaremos el algoritmo de *Simulated Annealing*. Mediremos y representaremos en un gráfico tridimensional los resultados para comparar las posibles combinaciones.

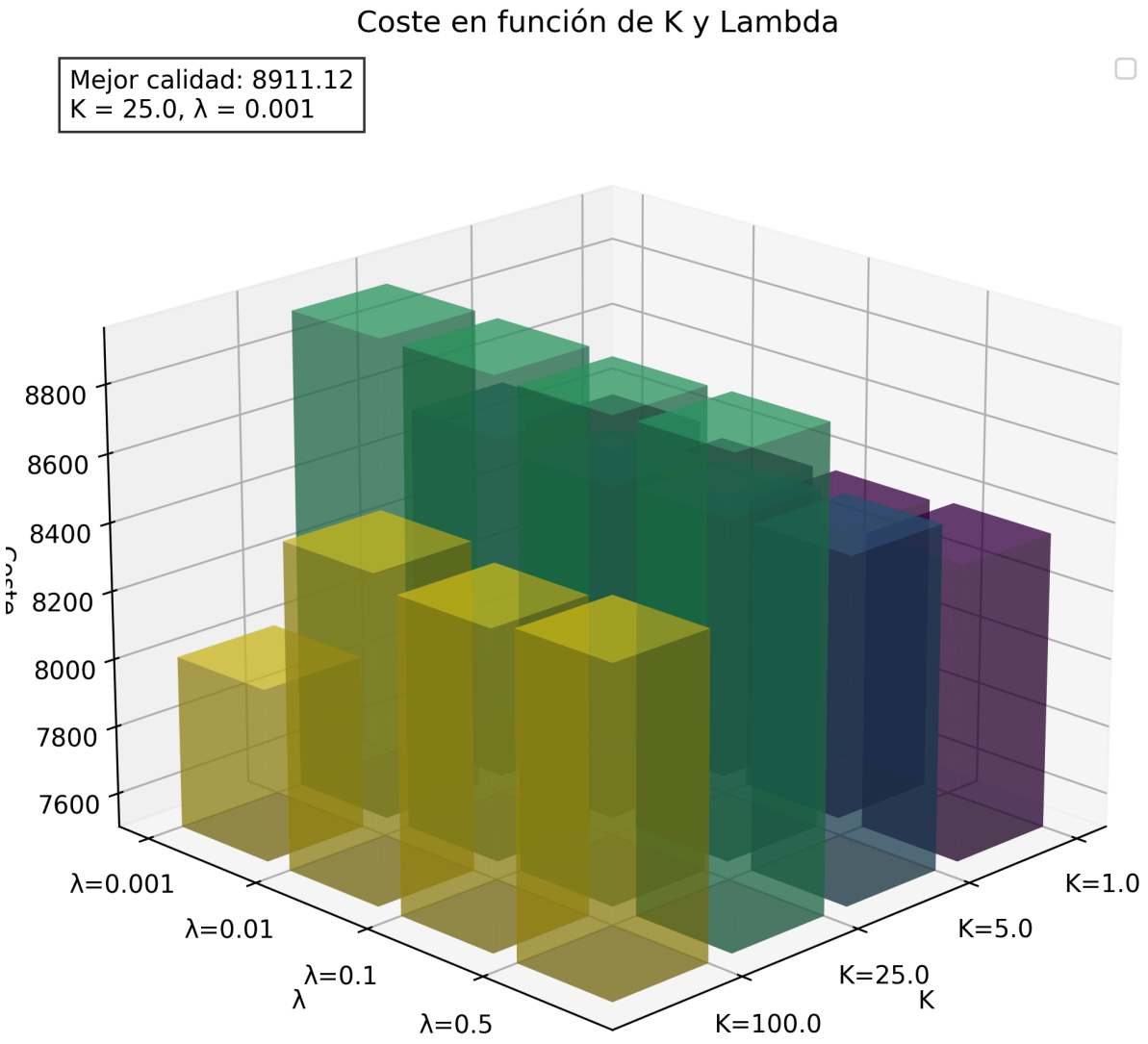


Figura 2.3.1: Distribución de la calidad en función de K y λ

Conclusión: Contrariamente a lo que esperábamos la mejor combinación de parámetros ha sido K = 25 y λ = 0.001. Al parecer una K demasiado alta afecta negativamente a la calidad, y la diferencia es notable alcanzando casi los 400 puntos.

De ahora en adelante el *Simulated Annealing* se ejecutará con parámetros K = 25 y λ = 0.001.