Problema del cartero chino

Victoria Beltrán Domínguez

October 2021

Índice

1. Descripción del problema

El problema del cartero chino es un problema de optimización muy conocido en el que un cartero debe repartir la correspondencia a cada una de las casas de su distrito, siendo la oficina de correos su punto de partida y llegada. En otras palabras, partiendo de la oficina de correos, el cartero deberá visitar todas las calles para poder realizar el reparto de toda la correspondencia y volver a la oficina de correos. Además, en este proyecto se ha escogido una variante del problema en la que el cartero va en moto, y por lo tanto también se debe considerar el sentido de circulación de las calles. Por lo tanto, para que una ruta sea válida debe:

- Partir de la oficina de correos
- Pasar por todas las calles de manera coherente (teniendo en cuenta el sentido de las calles y sus conexiones)
- Acabar en la oficina de correos

El problema del cartero chino no sólo reside en encontrar una ruta válida, pues como ya hemos comentado es un problema de optimización y por lo tanto el objetivo es encontrar aquella ruta que minimice la distancia recorrida por el cartero. En otras palabras, el objetivo en este problema es encontrar el camino más corto que pase al menos una vez por cada calle, tomando como punto inicial y final la oficina de correos.

2. Algoritmos genéticos

Una manera de abordar el problema del cartero chino es utilizando un algoritmo genético. Estos algoritmos pretenden llegar a una solución óptima del problema partiendo de una población inicial y aplicando diferentes funciones basadas en la evolución biológica como mutaciones, cruces, etc.

2.1. Diseño del algoritmo

Se procede a detallar de qué manera se ha diseñado cada una de las fases del algoritmo genético. Simplemente destacar que el algoritmo ha sido programado en *Python*, y la correspondiente implementación puede ser encontrada en el código adjunto.

2.1.1. Diseño del individuo

Una de las fases más importantes de un algoritmo genético es el diseño de un individuo. Antes de detallar cómo se ha diseñado el individuo, se quiere hacer hincapié en la información disponible del problema.

Disponemos de un mapa con las calles por las que el cartero debe pasar. Para manejar la información de manera adecuada, este mapa se transformará en un grafo donde las calles serán aristas y los nodos serán simples edificios, siendo el nodo A la oficina de correos. De esta manera, cada calle vendrá representada por una arista, y su dirección será especificada por el nodo partida y el nodo objetivo. Además, cada arista también dispondrá de una distancia asociada.

Una vez claro como hemos codificado la información del dominio, se procede a detallar de qué manera se ha codificado un individuo. Para nuestro problema, un individuo se codifica como un vector de talla variable cuyos contenidos representan el orden en el que las aristas son visitadas.

Se procede a representar un ejemplo de codificación de individuos partiendo del grafo de la Figura 1.

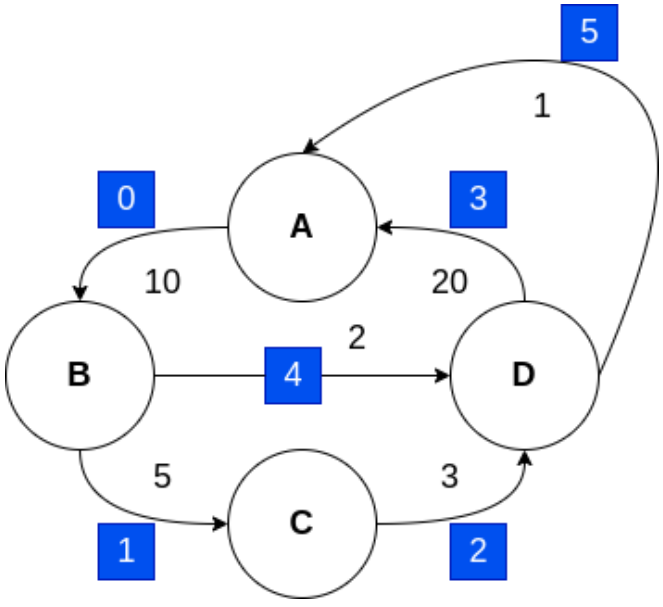


Figura 1: Grafo sencillo

En este, podemos observar cuatro aristas direccionadas {*0*, *1*, *2*, *3*, *4*, *5*} que representan calles (cada arista tiene un id remarcado en azul y una distancia) y cuatro nodos {A, B, C, D} que representan edificios. Asumiremos que la oficina de correos siempre se encuentra en la intersección A.

Algunos individuos posibles serían:

$$[0, 1, 2, 3, 0, 4, 3, 0, 4, 5]$$

$$[0, 1, 2, 3, 0, 4, 5]$$

$$[0, 4, 3]$$

Todos ellos representan un camino (sea este válido o no), y el orden en el que aparecen las aristas representa el orden en el que el cartero visitaría las calles correspondientes.

2.1.2. Función de evaluación

La función de evaluación tiene el objetivo de poder evaluar la calidad de un individuo.

En este problema, la calidad de un individuo dependerá de la distancia recorrida, puesto que buscamos la solución de menor distancia. De esta manera, la función de evaluación consistirá en un sumatorio de las distancias correspondientes a las aristas de la solución.

Sin embargo, tal y como hemos comentado en la descripción del problema, existen muchos individuos que no cumplen con todas las restricciones de este (no parten del nodo A, no pasan por todas las aristas, la secuencia de aristas es incorrecta o no acaban en el nodo A). Estas soluciones no son factibles.

Para diferenciar esta situación, la función de evaluación devolverá la distancia recorrida si la solución es factible, e infinito si la solución no es factible.

Como ejemplo, partiendo del grafo de la Figura 1, en la siguiente tabla se muestran diferentes individuos y cómo serían evaluados.

Individuo	**Factibilidad**	**Función Evaluación**
[0,1,2,3,0,4,5]	Sí	51
[0,4,3]	No (no pasa por todas las aristas)	infinito
[0,2,1]	No (camino no coherente)	infinito
[1,2,3,0,4,5]	No (no empieza en nodo A)	infinito
[0,1,2,3,0,4,5,0]	No (no acaba en nodo A)	infinito

Cuadro 1: Ejemplo resultados función de evaluación

2.1.3. Generación población inicial

Se han propuesto dos alternativas para la generación de población inicial:

- **Generación de población aleatoria**: El primer gen del individuo corresponde a alguna de las aristas que tenga como origen el nodo A. A partir de esta, haciendo uso de una matriz de adyacencia, se empieza a trazar un individuo de manera pseudoaleatoria, teniendo en cuenta las aristas que no se han visitado. La longitud del individuo corresponderá a un número aleatorio comprendido entre el número total de aristas definidas en el problema y la máxima longitud que el usuario haya definido.

  Si durante el proceso encontramos una solución factible, la añadimos a la población. Por otro lado, si durante el proceso llegamos a una calle sin salida, simplemente añadimos el individuo hasta el momento.

- **Generación de población utilizando heurística**: Esta forma de generar a la población inicial es idéntica a la anterior, con la diferencia de que las aristas no se eligen de manera aleatoria, sino que se elige aquella con más grado de conexión, de entre las aristas que no se han visitado. Durante el proceso, cuando ya se han visitado todas las aristas, la siguiente se toma de manera aleatoria.

En ambas alternativas se intenta guiar al individuo a la factibilidad (coherencia en el orden en el que se visitan las aristas, partir de la oficina de correos, intentar visitar todas las calles). Esto se ha implementado así con el objetivo de maximizar las posibilidades de encontrar soluciones factibles, que puedan dar lugar a algunas óptimas.

2.1.4. Selección

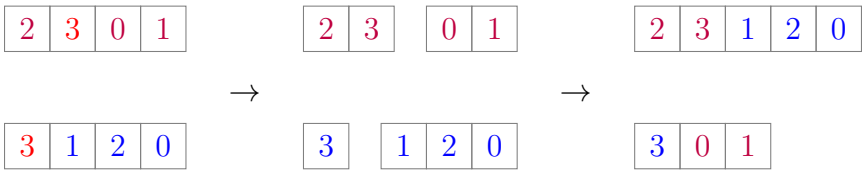
El objetivo de la selección es escoger a aquellos individuos de la población que van a poder reproducirse.

Para ello, hemos empleado la técnica de selección por torneo. En este método se realizan varios torneos entre algunos individuos de la población y el ganador de cada torneo es seleccionado para el cruce. En esta técnica, cuánto mayores sean las dimensiones del torneo menores serán las probabilidades de reproducirse de individuos más débiles. El número de individuos para reproducirse por defecto será la mitad de la población.

Si es preciso, se realizarán varias rondas de selección por torneo hasta obtener todos los individuos deseados.

2.1.5. Cruce

La operación de cruce consiste en dados dos padres aleatorios, cruzarlos para obtener descendientes. Este cruce entre dos padres consistirá en encontrar aristas comunes entre ellos. De entre todas las comunes, se escogerá una al azar aleatoria. Buscaremos el índice en ambos individuos y los partiremos por ahí, mezclando los trozos restantes entre ellos. Así, imaginemos que tenemos dos individuos: $[2, 3, 0, 1]$ y $[3, 1, 2, 0]$. Tal y como se puede ver en la imagen a continuación, seleccionamos al azar una arista común (en este caso tres), y partimos el individuo por ahí, mezclando el pedazo de un padre con el del otro para obtener los hijos.



2.1.6. Mutación

La operación de mutación trata de modificar la carga genética atendiendo a una probabilidad p de mutación. En este problema, para cada individuo de la nueva generación se intercambiará un gen con otro aleatorio (mutación por intercambio recíproco) con cierta probabilidad p (siendo p una probabilidad bastante baja para cada gen).

2.1.7. Reemplazo

La operación de reemplazo permite establecer cuál será la población de la siguiente generación o, dicho de otra manera, quiénes serán los supervivientes de esta generación. Esta fase se ha descrito en dos partes.

1. **Arreglar no factibles**: en este primera fase se intenta arreglar aquellos individuos no factibles en la población. Sin embargo, pasar de un individuo no factible a uno factible no es trivial. Se han contemplado principalmente dos situaciones para guiar a la solución a la factibilidad:
   - El camino no es correcto: si la secuencia de aristas es simplemente imposible (porque no son adyacentes), realizamos un barajado aleatorio del individuo.
   - El camino es correcto pero no pasa por todas las aristas del grafo: si existe alguna arista no visitada la añadimos al final de la secuencia.
2. **Reemplazo por estado estacionario**: escoger los N mejores individuos y descartar al resto. Nótese que si no tenemos N individuos factibles escogeremos los individuos no factibles más cortos (puesto que los hemos ordenado por fitness y en caso de empate por longitud) hasta tener una población de talla N.

2.1.8. Condición de parada

El bucle del algoritmo genético parará al llegar al número máximo de iteraciones o en caso de encontrar la solución óptima. Para nuestra ventaja, en este particular problema se conoce el coste del individuo óptimo, puesto que sería el coste de pasar solamente una vez por cada una de las aristas. Este camino óptimo se conoce como camino euleriano y si existe, sabemos que va a ser la mínima distancia que el cartero recorra. Sin embargo, no en todos los grafos existen caminos eulerianos. De esta manera, aunque desconocemos si hemos encontrado el camino óptimo para un problema en particular, conocemos la cota inferior de este, que sería la suma del peso de todas las aristas del grafo.

2.2. Evaluación

Una vez se ha explicado cómo se han diseñado e implementado las diferentes partes del algoritmo genético, en este apartado vamos a estudiar su comportamiento siguiendo diversos criterios. Para todos los resultados se realizarán cinco experimentos y sólo se expondrá la mediana de estos en vez de la media para evitar sesgos.

Antes de proceder, simplemente se quiere destacar que los grafos utilizados para la evaluación de este algoritmo han sido generados de manera aleatoria en *Python*, y se desconoce el valor de la solución óptima (solamente nos podemos guiar por la cota inferior que nos proporciona el hipotético camino euleriano). También debemos destacar que, si no se indica lo contrario, los parámetros por defecto que se van a utilizar son:

- cantidad población inicial = 100
- tipo de población = heurística
- tamaño del torneo = 2
- umbral = distancia del hipotético camino euleriano (suma del peso de todas las aristas)
- máximo número de iteraciones = 800
- probabilidad de mutación = 0.05
- mínima longitud del individuo = número de aristas diferentes
- máxima longitud del individuo = 2 x mínima longitud del individuo. Este parámetro es crítico y conviene tenerlo al mínimo posible, puesto que de este parámetro depende la talla de los mayores individuos.

2.2.1. Calidad de la solución vs iteraciones y tiempo de cómputo

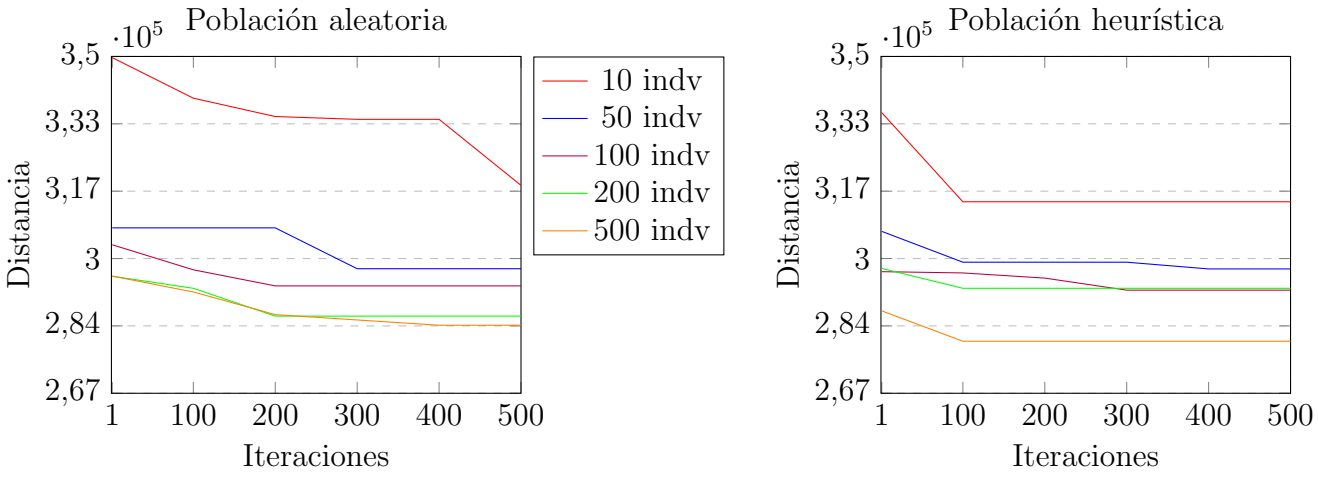
Para comparar la calidad de la solución respecto a las iteraciones y al tiempo de cómputo, se ha empleado un grafo completo y conexo de 651 aristas. Estos son los resultados obtenidos:

Experimento	**Población inicial**	**Número individuos**	**Número iteraciones solución**	**Fitness**
1	Heurística	2	100 (max)	346700
2	Heurística	2	100 (max)	333409
3	Heurística	5	100 (max)	334580
4	Heurística	5	1	328836 (optimal)
5	Heurística	10	1	328836 (optimal)
6	Heurística	10	100 (max)	329472
7	Heurística	10	1	328836 (optimal)
8	Pseudoaleatoria	2	100 (max)	339273
9	Pseudoaleatoria	2	100 (max)	341548
10	Pseudoaleatoria	5	100 (max)	333013
11	Pseudoaleatoria	5	1	328836 (optimal)
12	Pseudoaleatoria	10	1	328836 (optimal)
13	Pseudoaleatoria	10	100 (max)	330465
14	Pseudoaleatoria	10	1	328836 (optimal)

Para este tipo de grafos (dirigidos, completos y conexos), tanto una población inicial pseudoaleatoria como una población heurística consiguen encontrar la solución óptima. Como podríamos esperar, con una cantidad menor de individuos, aumenta la dificultad en encontrar el camino euleriano o el individuo óptimo. En cuanto al tiempo requerido, cada uno de los experimentos tarda menos de 1 segundo.

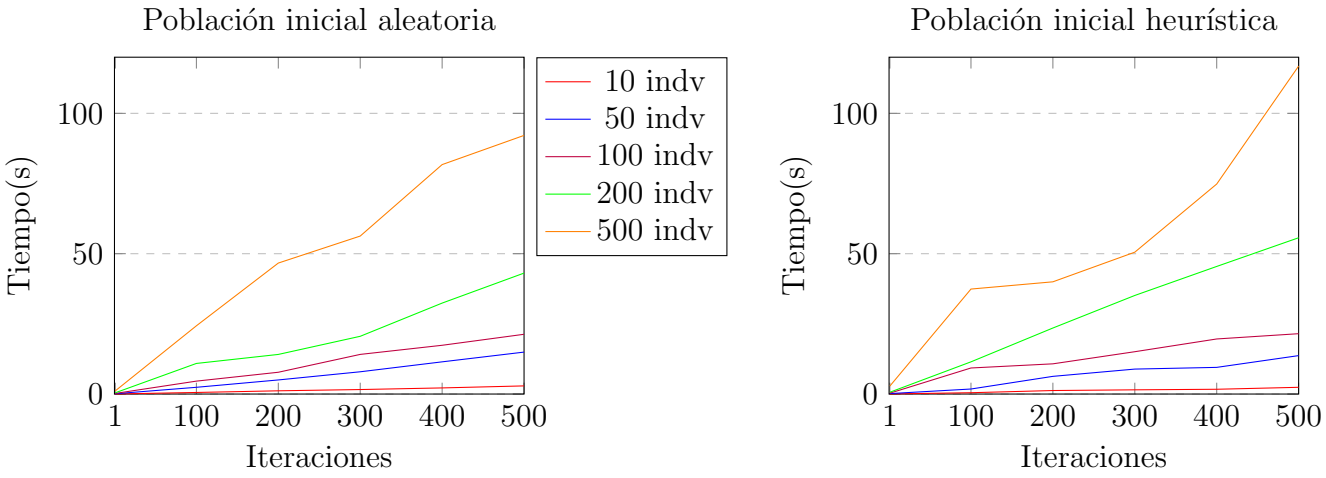
Para añadir dificultad al estudio, se han quitado 233 aristas al grafo inicial, para que este no tenga un camino euleriano trivial para el algoritmo. Este grafo vamos a denominarlo grafo reducido. La cota inferior de este se sitúa en **203714**.

Probando tanto con una generación poblacional aleatoria como con una generación poblacional heurística diferentes iteraciones y diferentes poblaciones iniciales, obtenemos las siguientes gráficas:



En estas podemos remarcar diferentes matices. Lo primero que llama la atención sobre ambas gráficas es que contra más individuos generamos en nuestra población inicial, mejor la solución que encontramos. Esto tiene bastante sentido, puesto que a más individuos generados, más posibilidades de encontrar uno mejor inicialmente. Vemos que los experimentos realizados con población aleatoria, aunque inicialmente tienen peor fitness, consiguen seguir mejorando y no estancarse hasta las 300/400 iteraciones. Sin embargo, en la población heurística tanto la generación inicial de muchos individuos (200 o 500 individuos) como la generación de muy pocos(10 individuos), se estanca a las 100 iteraciones. Esto o bien puede ser a causa de la poca diversidad genética o de la redundancia de los individuos. Para tallas intermedias (50 o 100 individuos), aunque logramos una evolución más allá de las 100 iteraciones, el algoritmo se estanca entre las 300/ 400 iteraciones igualmente.

Midiendo el correspondiente tiempo que ha tomado el cómputo de cada una de estos experimentos, en las siguientes gráficas podemos observar que aunque el manejo de 500 individuos sea el que ha obtenido un mejor individuo en ambos casos, es aquel que más ha costado en ejecutar. También se puede destacar que el algoritmo aleatorio toma un tiempo menor en ejecutarse que el heurístico (tal y como se podría esperar). Consideramos así que una población de 100 individuos sería lo ideal en relación tiempo y calidad de la solución obtenida, aunque siempre se podrían elegir más individuos en caso de tener más tiempo, o menos individuos en caso de querer rapidez.

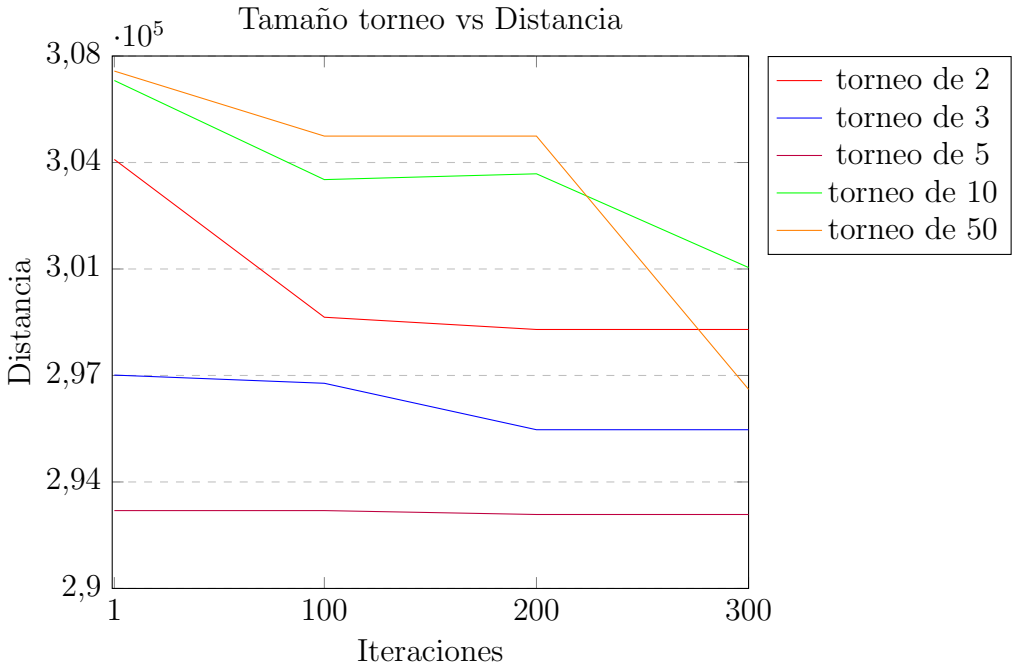


En cuanto a la diferencia entre generar la población inicial de manera aleatoria y generarla utilizando la heurística descrita previamente, consideramos que ni la diferencia temporal ni la calidad de la solución es un factor diferenciado en ningún método. Sin embargo, en caso de querer minimizar el tiempo sobre la calidad de la solución se optaría por el método aleatorio. En caso contrario, se elegiría el método heurístico.

2.2.2. Calidad de la solución vs tamaño de torneo selección

A continuación, vamos a experimentar cuál es el tamaño del torneo óptimo en la fase de selección. Para simplificar, vamos a utilizar el grafo reducido cuya cota inferior se sitúa en 203714. Utilizaremos una población inicial de cien individuos que han sido generados utilizando heurísticas. Los

resultado obtenidos para tamaños de torneo {2,3,5,10 y 50} hasta un máximo de 300 iteraciones puede observarse en la siguiente gráfica:



El comportamiento esperado a medida que la talla del torneo incrementa es uno cada vez más parecido al elitista. Sin embargo, nos topamos con una situación curiosa. Para torneos de 2, 3 y 5 participantes, el algoritmo encuentra soluciones mejores en menos iteraciones pero se estanca. Por otro lado, para tamaños de torneo de 10 y 50, el algoritmo empieza con individuos peores pero no se queda estancado. A pesar de no quedarse estancado, no consigue mejores soluciones que los anteriores.

2.2.3. Calidad de la solución vs probabilidad de mutación

Seguidamente, vamos a descubrir cómo influye la probabilidad de mutación en la solución. Para ello, utilizando el algoritmo genético con una generación inicial de población heurística y con un máximo de 300 iteraciones, iremos variando la probabilidad de mutación para observar qué cambios podemos ver en el fitness. Los resultados son mostrados en la tabla a continuación:

Probabilidad de mutación	**Fitness**
0	293011
0.05	301929
0.2	305811
0.5	309208
0.9	304779
1.0	306883

Los resultado obtenidos siguen una tendencia: a más probabilidad de mutación, peor el mejor individuo de la población. No es un hecho sorprendente puesto que nuestra mutación se basa

en un intercambio recíproco, y nuestro problema no sólo es de optimización sino también de satisfactibilidad, por lo que un cambio menor puede hacer que el individuo no sea correcto e invalidarlo.

2.3. Conclusiones

Los resultados obtenidos para resolver el problema del cartero chino a través de algoritmos genéticos son claros: aunque conseguimos obtener soluciones factibles, el algoritmo se estanca (converge en muy pocas iteraciones, no dando lugar a más mejoras). Se considera que para obtener una solución rápida, este tipo de algoritmo puede ser apropiado. Sin embargo, a medida que cruza y muta individuos, poco a poco va perdiendo material genético necesario hasta que no puede encontrar ninguna solución mejor. Esto también puede deberse a la dificultad del problema que se está utilizando, pues cada individuo factible consta de al menos 418 elementos, y muchos de ellos pueden repetirse.

3. Enfriamiento simulado

Dejando de lado los algoritmos genéticos, otra manera de abordar el problema del cartero chino es utilizando un algoritmo de enfriamiento simulado. Estos algoritmos pretenden llegar a una solución óptima del problema partiendo de una solución inicial y generando vecinos, intentando evitar estancarse en un espacio de soluciones.

3.1. Diseño algoritmo

Se procede a detallar de qué manera se ha diseñado cada una de las fases del algoritmo de enfriamiento simulado. Simplemente destacar que el algoritmo ha sido programado en *Python*, y la correspondiente implementación puede ser encontrada en el código adjunto.

3.1.1. Soluciones vecinas. Solución inicial.

La solución inicial se obtiene a partir de la ejecución del algoritmo genético. Para la obtención de soluciones vecinas, se han seguido dos propuestas diferentes:

- **Obtención de vecinos heurística**: Cogemos un índice aleatorio de la solución inicial. El individuo entonces se acorta hasta ese índice y se procede como en la generación de población heurística comentada en el apartado de algoritmos genéticos, con la diferencia de que si no se llega a una solución factible en una longitud máxima fijada por el usuario, esta no se añade a la población y se suma una nueva iteración. Finaliza al construir un número determinado de individuos factibles o al cabo de un máximo de intentos, si no se consigue ninguno.
- **Obtención de vecinos aleatoria**: Cogemos un índice aleatorio de la solución inicial. El individuo entonces se acorta hasta ese índice y se extiende iterativamente utilizando la arista adyacente no visitada aleatoria. Si no encuentra solución, utiliza la heurística para poder apoyarse. Finaliza al cabo de un máximo de intentos, si no se consiguen individuos factibles.

En el apartado de algoritmos genéticos, se ha consentido el manejo de individuos no factibles. Esto ha sido así debido a que se consideraba fundamental que no todos los individuos fueran perfectos para conseguir una buena solución. Sin embargo, en enfriamiento simulado no se considera trabajar bajo ninguna circunstancia con soluciones no factibles, pues estas no nos aportan absolutamente nada.

3.2. Evaluación

Una vez se ha explicado cómo se han diseñado la vecindad en el algoritmo de enfriamiento simulado, en este apartado vamos a estudiar su comportamiento siguiendo diversos criterios.

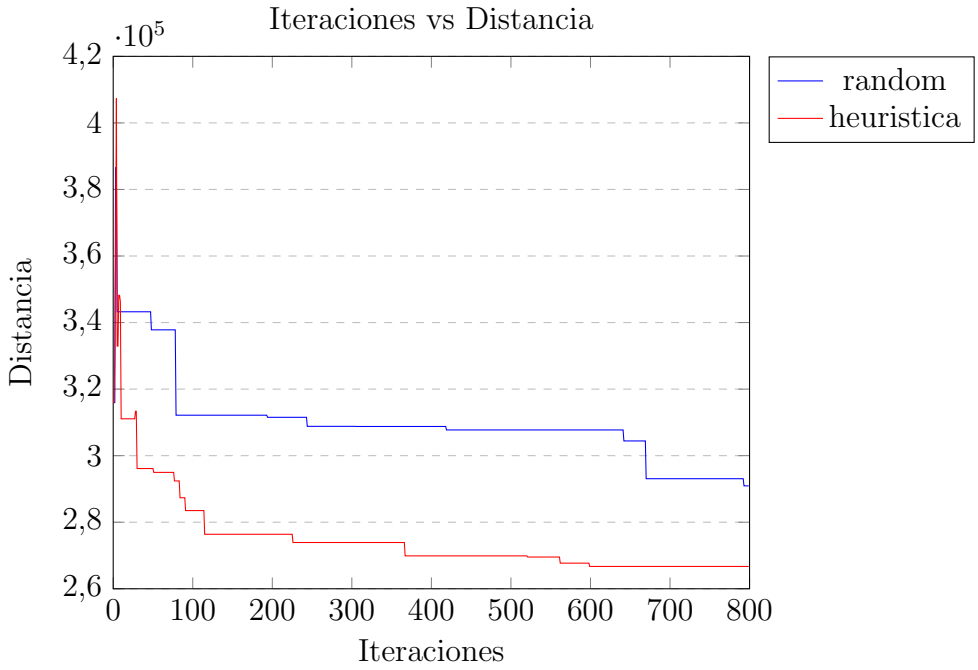
Antes de proceder, simplemente se quiere destacar que nuevamente, vamos a utilizar el grafo reducido de cota inferior que nos proporciona el hipotético camino euleriano. También debemos destacar que, si no se indica lo contrario, los parámetros por defecto que se van a utilizar son:

- tipo de vecindad = heurística
- temperatura inicial = 40000

- k = 0.9
- actualización temperatura = k x temperatura
- máximo número de iteraciones = 800
- máxima longitud del individuo = 2 x longitud del individuo inicial. Este parámetro es crítico y conviene tenerlo al mínimo posible, puesto que de este parámetro depende la talla de los mayores individuos.

3.2.1. Calidad de la solución vs iteraciones

Para comparar la calidad de la solución respecto a las iteraciones y al tiempo de cómputo, como ya se ha comentado se ha empleado el grafo reducido. Estos son los resultados obtenidos:

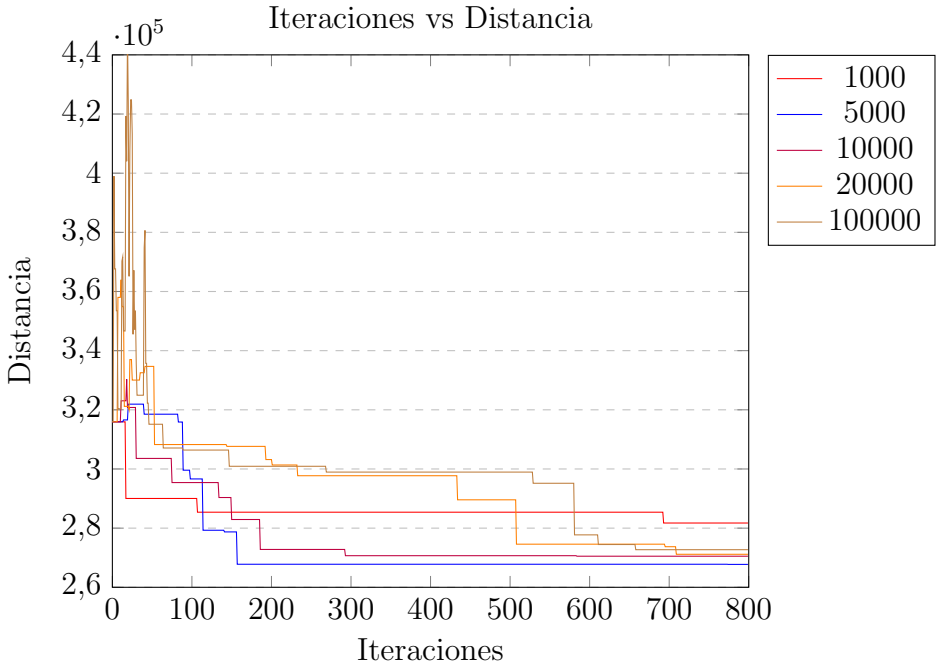


El tiempo empleado para obtener las 800 iteraciones utilizando el método aleatorio ha sido de 167.48 segundos, mientras que el tiempo empleando en heurísticas ha sido de 125.57 segundos. De la gráfica, podemos extraer diferentes conclusiones. Podemos ver como inicialmente la temperatura permite individuos peores y a medida que avanzan las iteraciones cada vez es más restrictivo. También podemos ver que generalmente no existen muchos despuntes. Esto quizá lo podemos atribuir a la inteligencia a la hora de crear vecinos, pues quizá por la forma en la que los generamos, es difícil obtener uno peor a partir del anterior.

Otro aspecto importante a remarcar es el método de generación de vecinos. Los resultados obtenidos muestran que el método heurístico obtiene mejores individuos que el método aleatorio. Es decir, a partir de una solución inicial, el método heurístico consigue obtener mejores vecinos.

3.2.2. Temperatura inicial vs Distancia

A continuación, vamos a realizar experimentos acerca de la relación entre la temperatura inicial y la distancia recorrida. Para ello se va a utilizar un valor fijo de k = 0.95 con el método de actualizar la temperatura por defecto. Los resultados utilizando como temperaturas iniciales 1000,5000,10000,20000,100000 son:



Como podemos observar, con una temperatura de 1000 el algoritmo apenas permite individuos peores, impidiendo así explorar diferentes caminos y estancándose en mínimos locales. Para temperaturas superiores, el algoritmo permite individuos peores, por lo que se exploran diferentes posibilidades y se consigue una mejor solución.

3.2.3. Formas de actualizar vs Distancia

Una vez descubierta la importancia de la temperatura inicial, se procede a comprobar el impacto del método de actualización de la temperatura. Los resultados obtenidos pueden verse en la siguiente tabla (x indica que k era demasiado pequeña y en este método no se podía aplicar por la drástica reducción de la temperatura):

Actualización temperatura	**k**	**Fitness**
$temperatura_{inicial} - iteracion * k$	0.05	287020
	0.1	280499
	0.8	273503
	0.99	271581
$k * temperatura$	0.05	x
	0.1	x
	0.8	273858
	0.99	284725
$temperatura_{inicial}/(1 + k * temperatura)$	0.05	279168
	0.1	279908
	0.8	273830
	0.99	275455

Para el primer método de descuento (método de descuento lineal), vemos que a mayor k, mejores los resultados obtenidos. Por otro lado, para el resto de métodos la mejor k es una moderadamente alta. Tampoco debemos poner una altísima, pues la temperatura bajará rapidísimo y permitirá muy pocos individuos peores. No podemos destacar ninguno de los tres métodos como el mejor, pues en general, el último método obtiene mejores resultados, pero el primero ha obtenido el mejor resultado encontrado en este experimento y el segundo no tiene suficientes experimentos (al no ser k suficientemente grande).

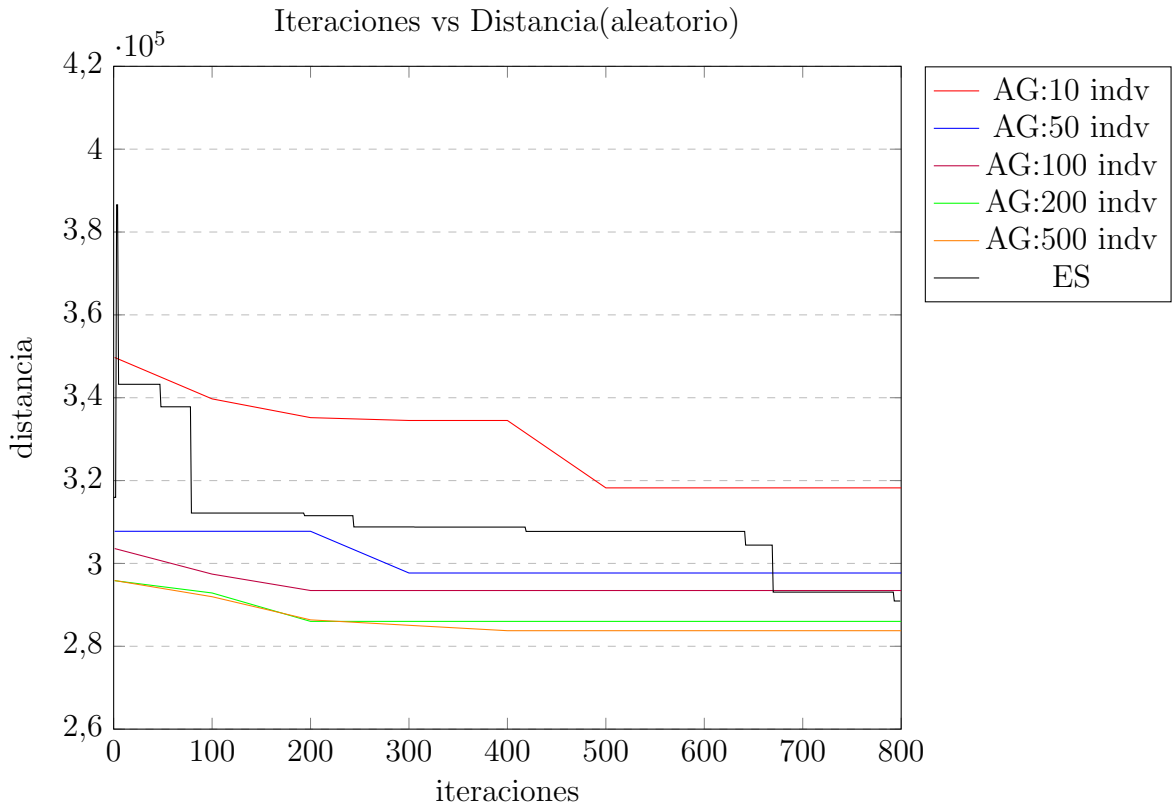
3.3. Conclusiones

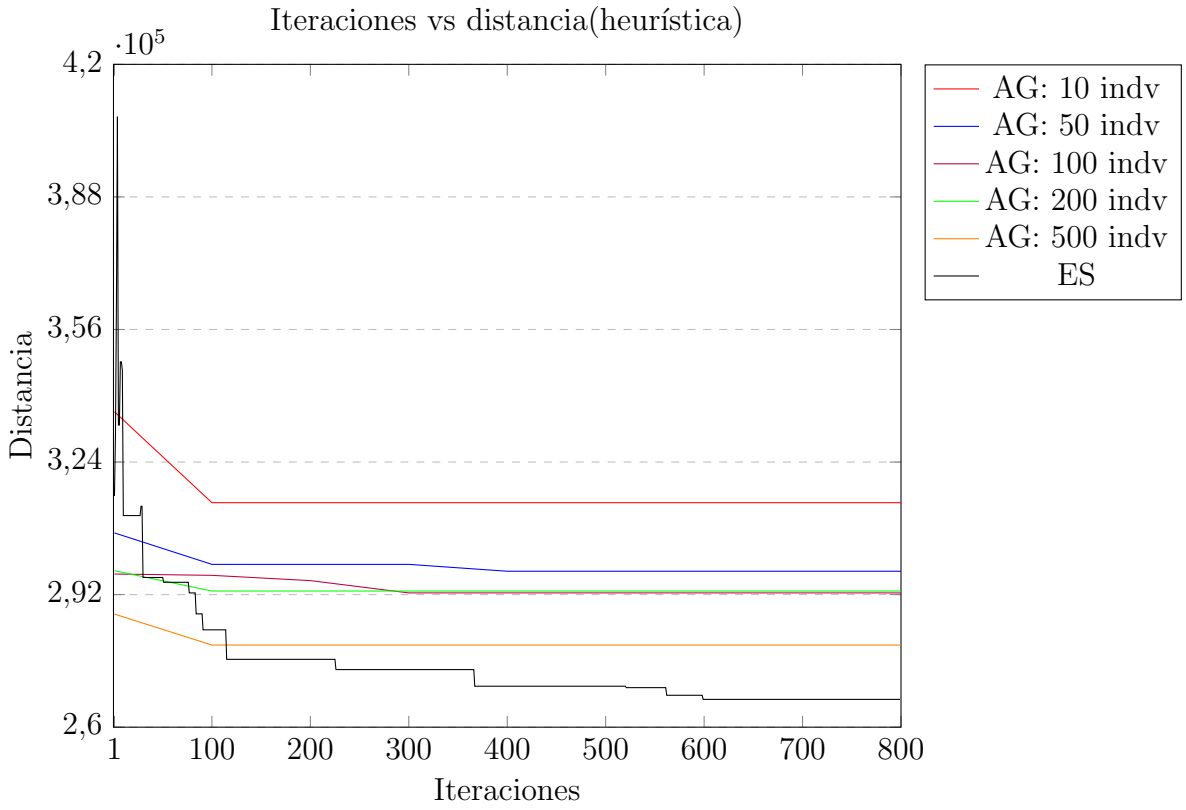
Los resultados obtenidos para resolver el problema del cartero chino a través de algoritmos de enfriamiento simulado funcionan bastante bien. Este consigue obtener soluciones factibles, minimizarlas y evitar estancamientos. Sin embargo, para maximizar el funcionamiento de este algoritmo, deberemos conseguir un equilibro entre el método de descuento de la temperatura, la temperatura inicial y la k. Con ello, nuestro objetivo será así evitar que nuestro algoritmo se estanque y pueda explorar diferentes individuos. Utilizando los parámetros apropiados, se considera que este algoritmo funciona genial, y que es un buen algoritmo en relación calidad de la mejor solución obtenida y tiempo de cómputo.

4. Conclusiones

Como conclusiones generales de este proyecto, se considera que para el problema del cartero chino es mucho más eficiente utilizar enfriamiento simulado que algoritmos genéticos. Mientras que el enfriamiento simulado consigue salir de óptimos locales y seguir mejorando, algoritmos genéticos se queda estancado en las primeras 300 iteraciones.

A continuación, se muestran dos gráficas con los resultados de los dos métodos para diferentes métodos: aleatorio y heurístico.





Como podemos ver, utilizando el método aleatorio para los individuos o vecinos, ambos algoritmos consiguen más o menos la misma calidad. Sin embargo, utilizando heurísticas en un sólo individuo conseguimos valores óptimos que no habían explorados con anterioridad.

Cabe destacar, que aunque sabemos que para este problema la cota inferior es de 203714, no sabemos a cuanto se ha quedado de la solución óptima.

Para finalizar, destacar que en caso de realizar una recomendación, obtendríamos una primera solución inicial con el algoritmo genético (con una población inicial aleatoria de 50 a 100 individuos) y 800 iteraciones y a continuación aplicaríamos enfriamiento simulado con esa solución obtenida para conseguir minimizarla. Gran cuidado con los parámetros relacionados con la actualización de la temperatura, pues deben ser permisivos hasta cierto punto.

Se considera que en general este será el mejor procedimiento para el problema del cartero chino.