Reporte de la solución al Problema del Agente Viajero.

Índice

Resumen -------------------------------------------------------------2

Introducción ---------------------------------------------------------2

Planteamiento de la solución -------------------------------3

* Individuos
* Evaluación del fitness
* Operación de mutación
* Operación de cruza
* Operación de selección
* criterio de paro
* Ejemplo

Experimentos ------------------------------------------------------7

Conclusiones -----------------------------------------------------11

Bibliografía ---------------------------------------------------------11

Resumen

El presente trabajo tiene como objetivo principal presentar la solución planteada por el alumno para el problema del agente viajero (Travel Salesman Problem, TSP por sus siglas en inglés).

Se presentará una breve introducción donde se describe el problema, como el alumno modelo el problema para pensar en su solución, operaciones realizadas, corridas del algoritmo y conclusiones del mismo.

Introducción

El problema del agente viajero responde a la siguiente pregunta: dada una lista de ciudades y las distancias entre cada par de ellas, ¿cuál es la ruta más corta posible que visita cada ciudad exactamente una vez y al finalizar regresa a la ciudad origen? Este es un problema NP-Hard dentro en la optimización combinatoria, muy importante en investigación operativa y en ciencias de la computación.

El problema fue formulado por primera vez en 1930 y es uno de los problemas de optimización más estudiados. Aunque el problema es computacionalmente complejo, se conoce gran cantidad de heurísticas y métodos exactos, así que es posible resolver planteamientos concretos del problema desde cien hasta miles de ciudades.

Planteamiento de la solución

Para solucionar este problema es necesario definir cómo lo vamos a trabajar y que tipo de algoritmo utilizaremos.

En este caso el algoritmo seleccionado para trabajar este problema es el de Estrategias Evolutivas y para hacer que funcione necesitamos los siguientes objetos:

1. Una población con N individuos.
2. Una población Lambda con N individuos nuevos.
3. Evaluación del fitness (distancia a recorrer).
4. Operación de Mutación.
5. Operación de Cruza.
6. Operación de selección.
7. Criterio de paro
8. Ejemplo

Individuos

El modelado de los individuos fue planteado de la siguiente manera:

Teniendo las coordenadas de 10 ciudades y una población de 5 individuos:

Ciudades a recorrer:

[[981, 526], [917, 376], [232, 80], [177, 361], [1071, 403], [1023, 268], [809, 628], [1078, 616], [808, 718], [184, 712]]

Población = [

h0 [2, 3, 7, 5, 0, 1, 8, 4, 9, 6],

h1 [1, 5, 3, 7, 0, 8, 4, 6, 2, 9],

h2 [1, 6, 5, 9, 0, 8, 7, 4, 2, 3],

h3 [4, 8, 1, 2, 7, 0, 6, 3, 9, 5],

h4 [1, 2, 5, 3, 9, 6, 0, 8, 4, 7],

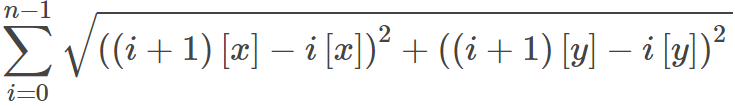
]

Cada individuo representa una combinación aleatoria de un posible recorrido de las ciudades sin repetir alguna de estas y se muestran en lo que sería nuestra población inicial para trabajar sobre esta en el algoritmo.

Evaluación del Fitness

El cálculo que se va a utilizar para evaluar el fitness de cada individuo es la fórmula de la distancia euclidiana la cual es:

, esta fórmula nos da como resultado la distancia vectorial entre 2 ciudades, así que para poder adaptarla a este problema es necesario incluir una sumatoria para que se calculen la suma de las distancias entre ciudades hasta n-1 (siendo n el número de ciudades) lo cual nos quedaría algo así:



Esta fórmula nos daría como resultado el valor total que le cuesta al viajero recorrer todas las ciudades en el orden que el individuo propone.

Operación de mutación

La mutación es una operación bastante sencilla a aplicar en este problema ya que es tan simple como intercambiar la posición de 2 ciudades dándonos como resultado el primer individuo de nuestros lambda nuevos, de esta forma ayudamos al algoritmo a que pueda encontrar un mejor camino y validamos que las ciudades a recorrer no se repitan, aquí se muestra con un ejemplo:

Seleccionamos aleatoriamente un individuo de nuestra población inicial

h2 [1, 6, 5, **9**, 0, **8**, 7, 4, 2, 3]

Se generan 2 números aleatorios entre 0 y 9: num1 = **5**, num2 = **3**, se intercambian las ciudades de estas posiciones y listo.

h5 [1, 6, 5, **8**, 0,**9**, 7, 4, 2, 3]

Operación de Cruza

La cruza es una operación mucho más complicada que la mutación ya que se nos dificulta la validación de que no existan ciudades repetidas en nuestro nuevo individuo.

La cruza que fue utilizada es la cruza cíclica:

En esta cruza se construyen dos descendientes en los cuales cada valor y su posición se toman de uno de los padres. Por ejemplo:

h1 = (1, 5, 3, 7, 0, 8, 4, 6, 2, 9) y h3 = (4, 8, 1, 2, 7, 0, 6, 3, 9, 5)

Podemos producir el primer descendiente tomando el primer valor del primer padre

h1: d1 = (1 x x x x x x x x x)

Dado que el valor 4 en h3 se encuentra justo en la misma posición del valor seleccionado 1, seleccionamos el valor 4.

En h1 el valor 4 está en la posición 6 por lo que se tiene

d1 = (1 x x x x x 4 x x x)

Siguiendo esta regla, ahora el valor 6 en h3 se encuentra en la misma posición que el valor 4 en h1 por lo que se elige 6

d1 = (1 x x x x x 4 6 x x)

Ahora el valor 3 en h3 se encuentra en la misma posición que el valor 6 en h1 por lo que se elige 3

d1 = (1 x 3 x x x 4 6 x x)

Ahora el valor que se encuentra en la misma posición en h2 que el valor 3 en h1 es el valor 1, que ya se encuentra en el descendiente, por lo que se cerró un ciclo, ahora los valores faltantes se toman del otro padre

d1 = (1 8 3 2 7 0 4 6 9 5)

Similarmente d2 = (4 5 1 7 0 8 6 3 2 9)

Como se está trabajando con un algoritmo evolutivo no podemos meter a 2 hijos nuevos en la nueva población, así que los ponemos a competir; se calculan los fitness respectivos de los dos hijos y se hace una selección completamente elitista, el hijo que tenga el menor fitness será el que sobreviva y pase a los lambda nuevos.

Operación de Selección

La selección que se utiliza es completamente elitista, teniendo nuestra lista de fitness, se selecciona el mejor individuo que pasa directamente a la siguiente generación, después se selecciona al segundo mejor y así sucesivamente hasta que se llene la población de la siguiente generación.

Criterio de paro

Para que estos tipos de algoritmos funcionan de una mejor manera es necesario implementar un criterio de paro ya que no lo podemos dejar corriendo infinitamente y tampoco le podemos decir que corra un número de veces ya que lo estaríamos limitando a un resultado el cual podría no ser el mejor.

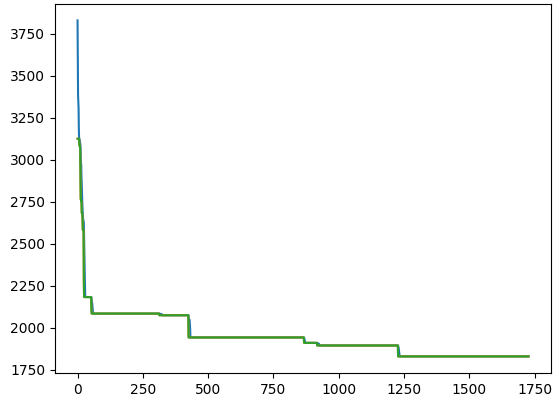
Para este criterio de paro le diremos al algoritmo que corra n veces (siendo n, establecido por el usuario). La primera iteración almacenaremos en una variable el mejor fitness encontrado, haremos todas las operaciones listadas anteriormente y para la segunda operación guardaremos en otra variable el mejor fitness encontrado en esa iteración, si el mejor fitness encontrado es mejor que el mejor fitness de la iteración, el algoritmo continúa incrementando iteraciones, pero si el mejor fitness de la iteración en mejor que el mejor fitness encontrado, esta variable se actualiza con el valor del mejor fitness de la iteración y las iteraciones se reinician, dando así más oportunidad de que se encuentre un mejor resultado.

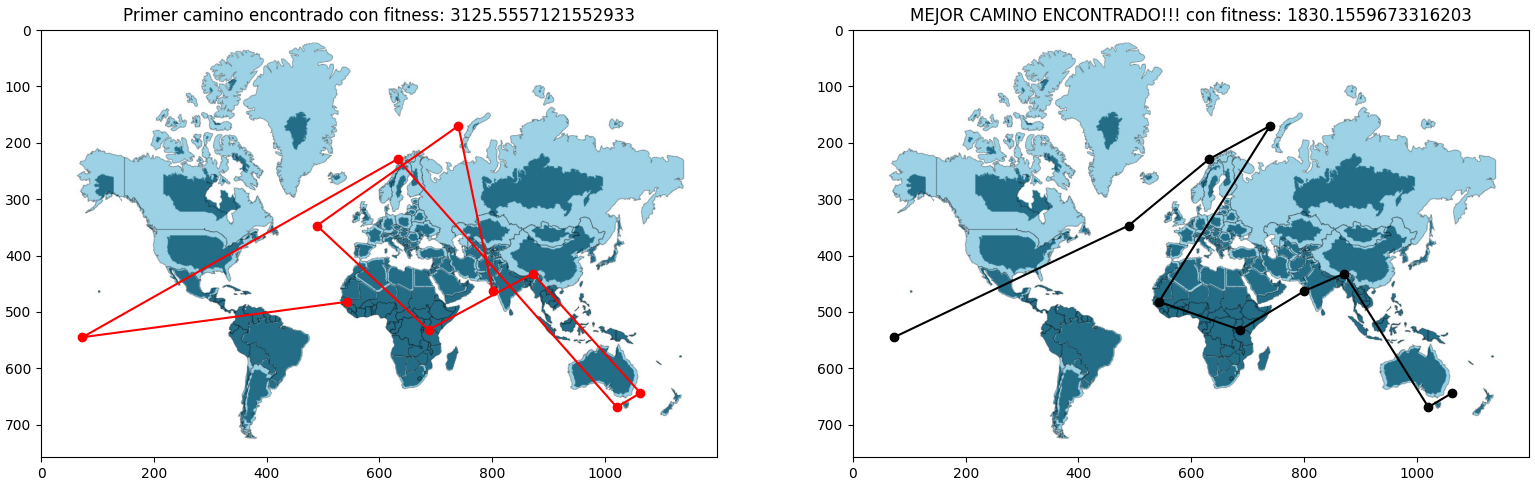
En este tipo de problemas es muy posible que lleguemos a un resultado óptimo pero no al mejor, de modo que se estaría atorando en un máximo local, es por eso que debemos ayudar al algoritmo a que no suceda esto; para eso le diremos que si pasaron n/2 iteraciones sin encontrar un mejor resultado, duplicaremos nuestro valor de mutación, haciendo que cuando mutemos a un individuo, se intercambiaran 4 ciudades en lugar de 2, dándole chance de que encuentre un mejor resultado.

Cuando se completen las n iteraciones el algoritmo se detendrá arrojando el mejor resultado encontrado.

Ejemplo de corrida

Gráfica de los promedios de los fitness





El mejor fitness se encontró en 1227 Iteraciones

El mejor fitness encontrado fue:1830.1559673316203

La mejor combinación es:

[3, 9, 4, 5, 1, 8, 2, 7, 0, 6]

Tiempo transcurrido: 3.3624801635742188 segundos

Experimentos

Se realizaran 3 experimentos con diferentes valores:

**Primer experimento:**

Ciudades: 50

Población: 20

Lambda nuevos: 15

Cruza: 0.7

Mutación: 0.3

10,000 iteraciones

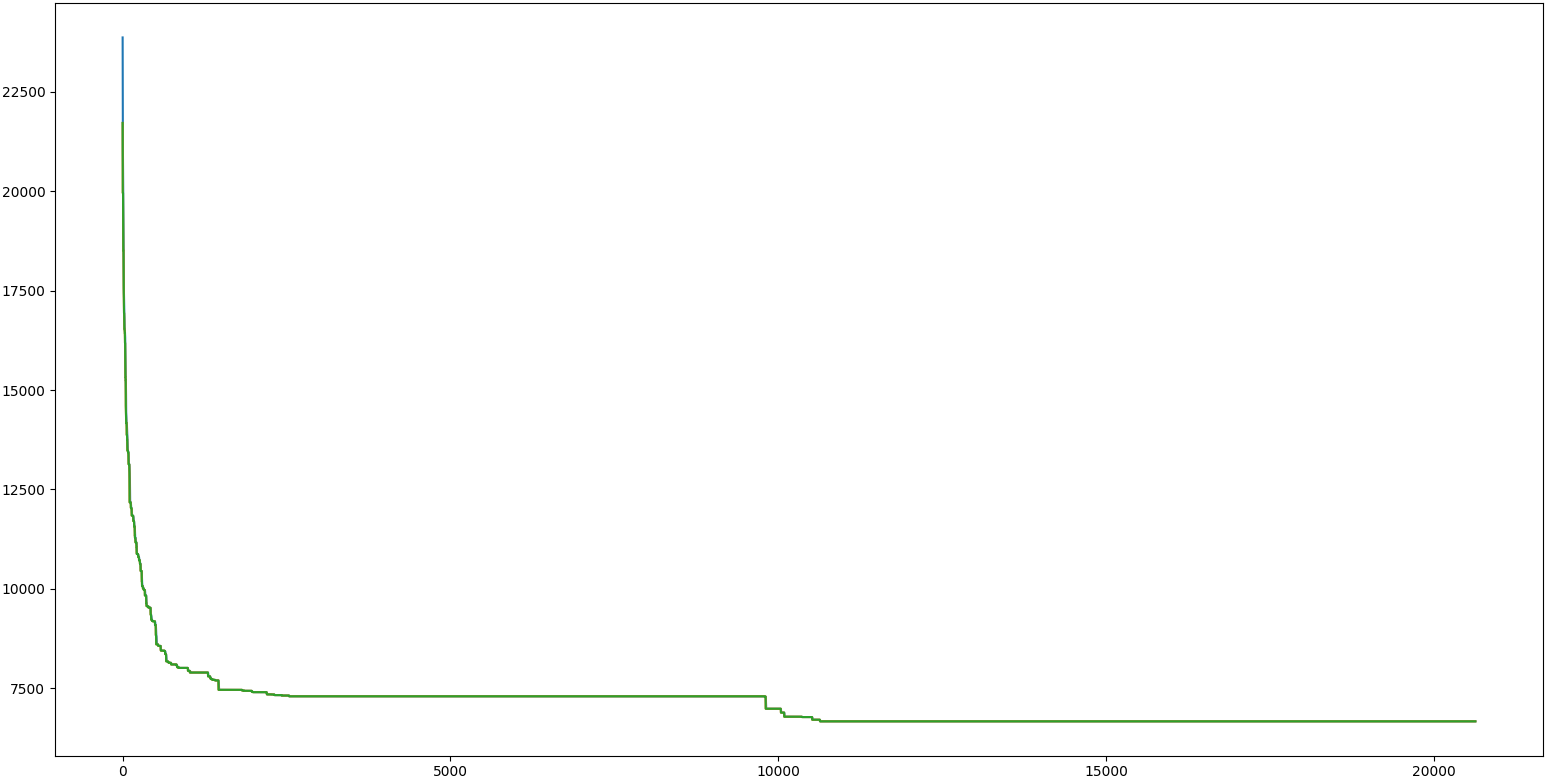
RESULTADOS:

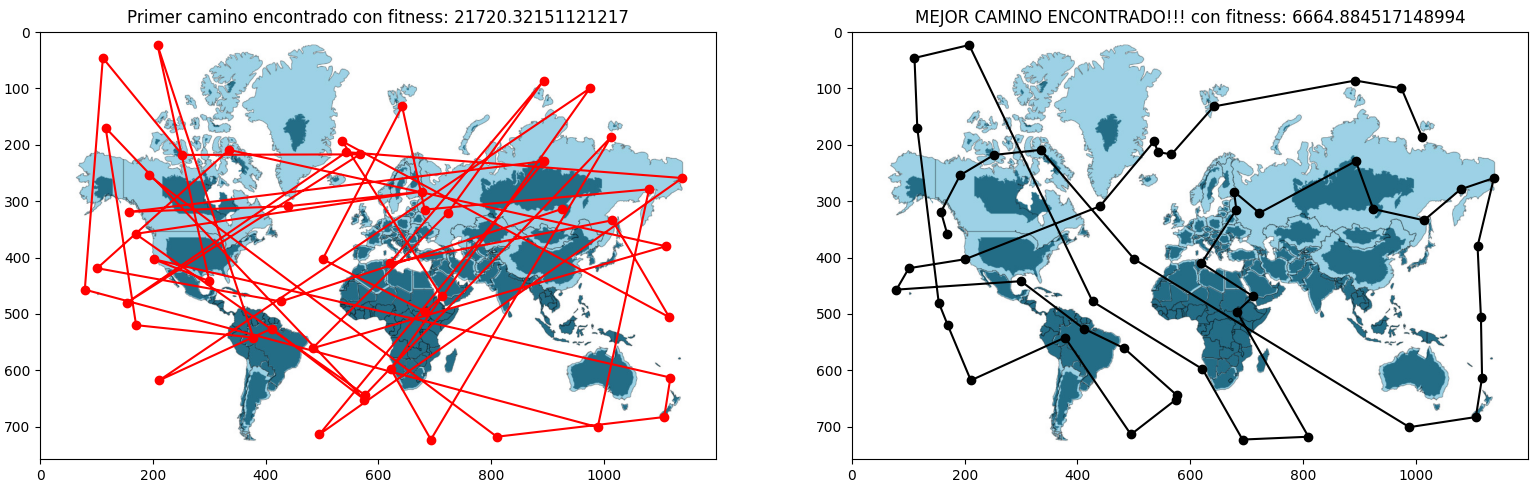
Tiempo transcurrido: 82.077 segundos

Se encontró en 10,637 Iteraciones

El mejor fitness es de: 6664.8845

Promedio de los fitnes:





**Segundo experimento:**

Ciudades: 100

Población: 20

Lambda nuevos: 15

Cruza: 0.7

Mutación: 0.3

10,000 iteraciones

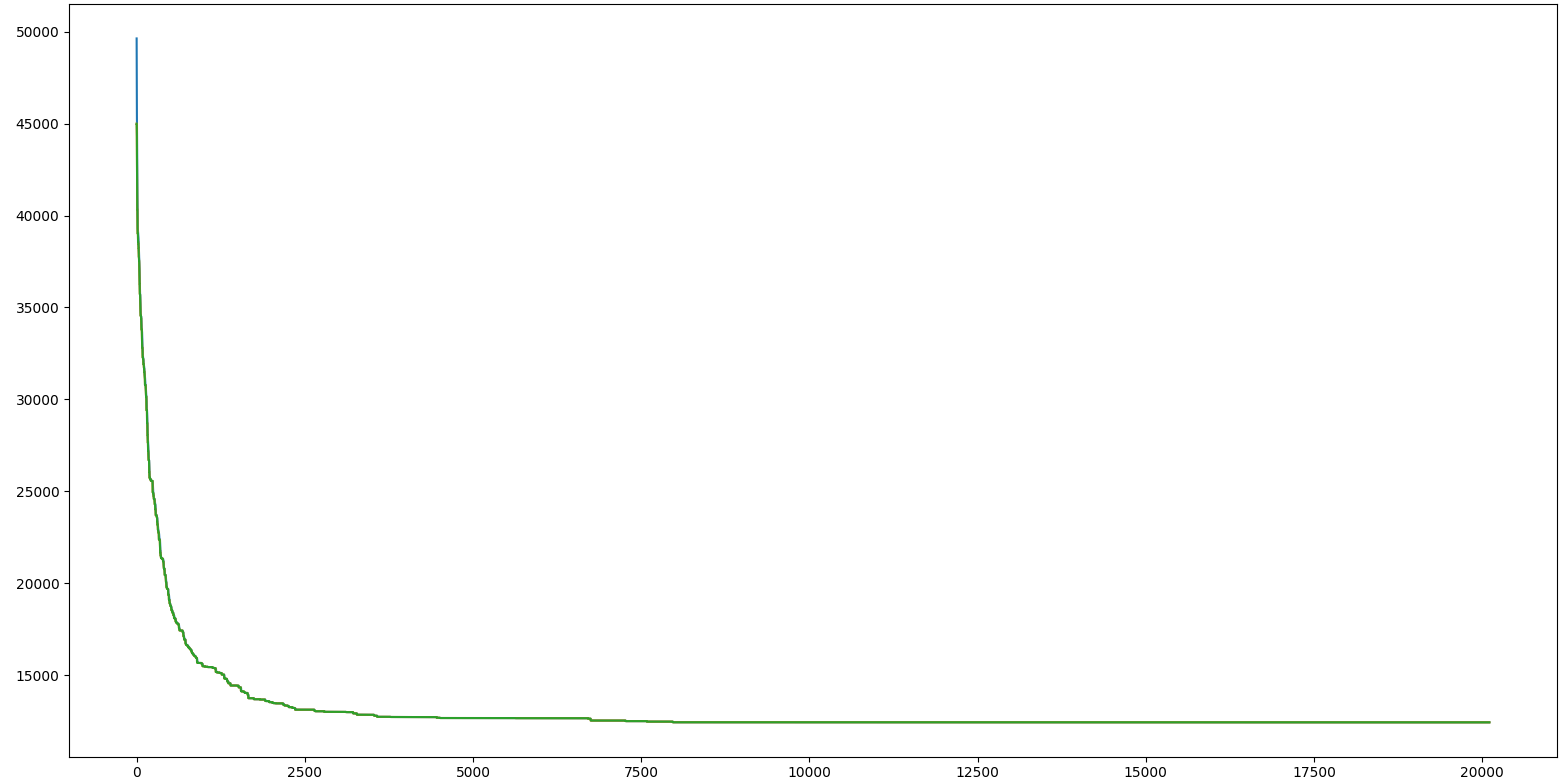
RESULTADOS:

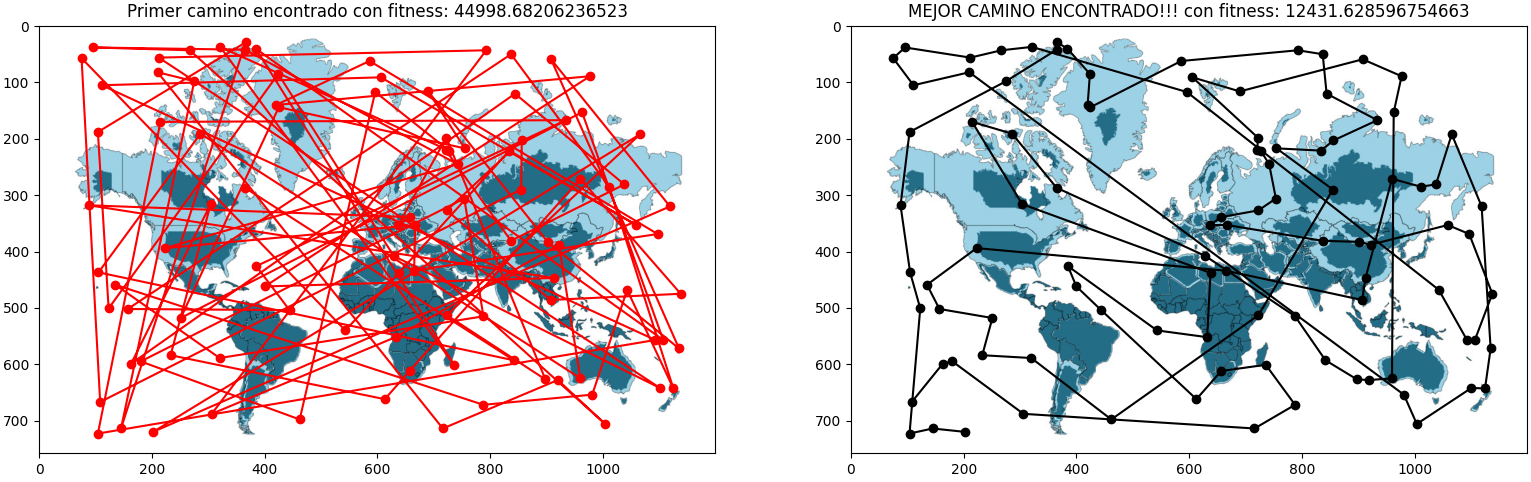
Tiempo transcurrido: 138.641 segundos

Se encontró en 10,107 Iteraciones

El mejor fitness es de: 12431.628

Promedio de los fitness:





**Tercer experimento:**

Ciudades: 1000

Población: 50

Lambda nuevos: 25

Cruza: 0.8

Mutación: 0.2

10,000 iteraciones

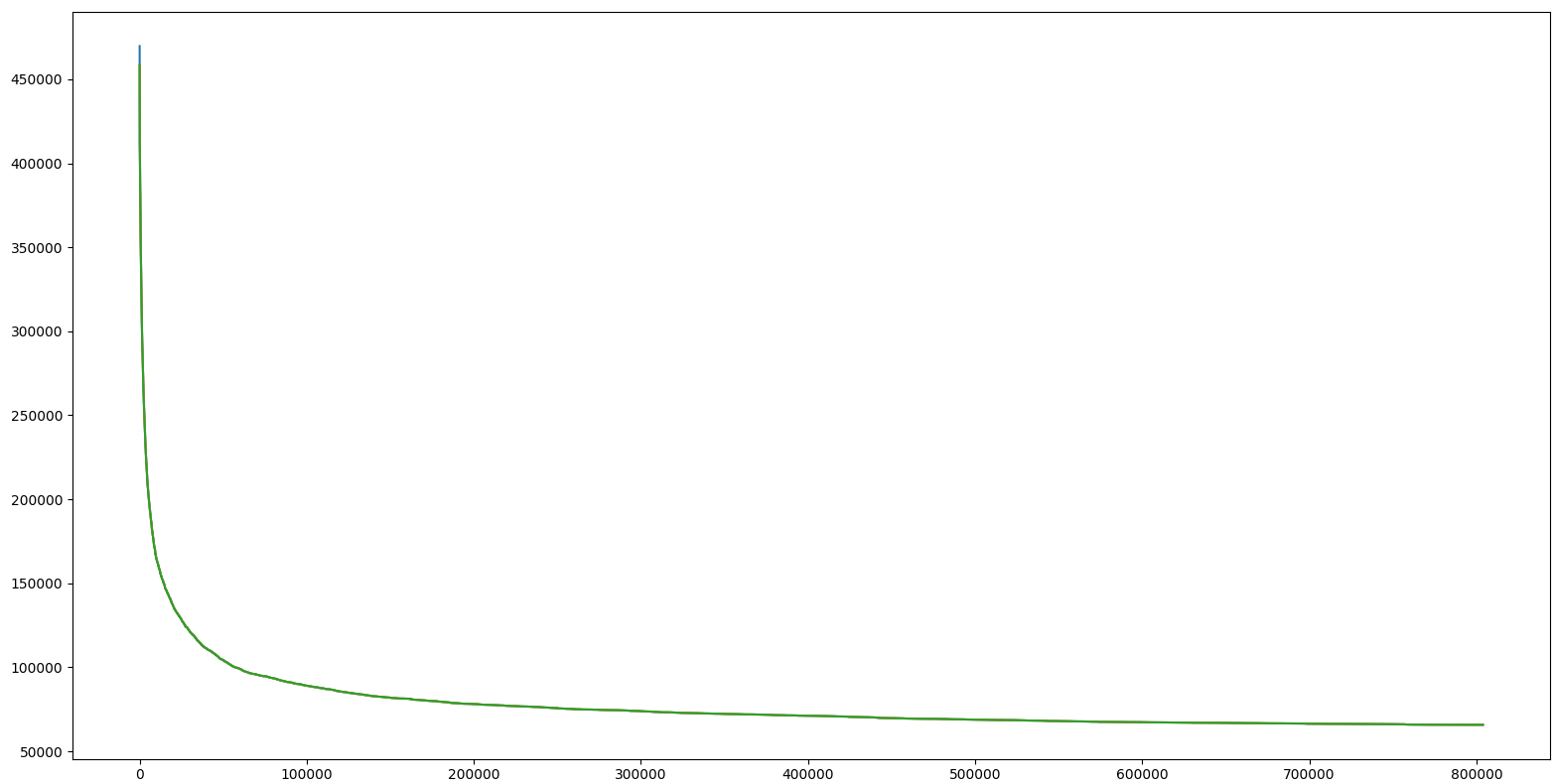
RESULTADOS:

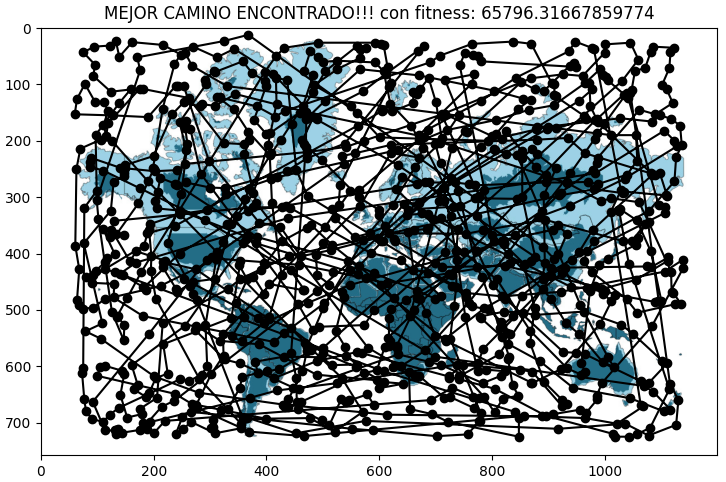
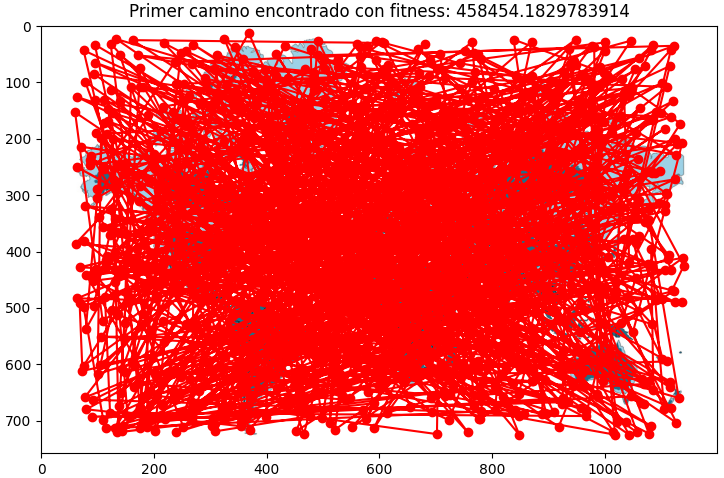
Tiempo transcurrido: 135,421.641 segundos (37.6169444)

Se encontró en 793,861 Iteraciones

El mejor fitness es de: 65,796.628

Promedio de los fitness:





Conclusiones

Los resultados obtenidos fueron bastante buenos y después de haber realizado más experimentos podemos concluir que la operación de la cruza en este algoritmo es muy importante, ya que al ser mutación solo cambiar la posición de 2 ciudades no ayuda mucho, se podría incrementar el porcentaje de veces que se mutan ciudades para que le ayude mucho más.

La verdad me sorprendió bastante lo que duró la corrida del algoritmo con 1000 ciudades, previamente ya lo había corrido con 1000 pero con un porcentaje de cruza de 20% y de mutación de 80%, con estos valores había corrido un total de 6 horas lo cual lo veía normal, para este experimento se usó un porcentaje de cruza del 80% y 20% de mutación, y aquí se puede ver la gran diferencia con los valores de cruza ya que siempre mejoraba y mejoraba lo cual hacía que el algoritmo no se detuviera a no ser que ya no pudiera mejorar.

Cabe recalcar que este algoritmo no encuentra la MEJOR ruta que se puede encontrar ya que es muy difícil encontrarla, pero analizando los resultados obtenidos en los experimentos podemos ver la diferencia del primer fitness al mejor encontrado, solo observando pasamos de 458,454.4156 a 65,796, decreció 392,658 de fitness lo cual es increíblemente bueno.

Estoy bastante satisfecho con el resultado de este algoritmo, pero siento que puede mejorar bastante, trabajare en un futuro para que logre encontrar un mejor juego de resultados que los actuales.

Bibliografía

(Fuentes Penna, 2015)

(Wikipedia, 2018)