Análisis de Algoritmos
Proyecto Final

Arturo Márquez Flores
Maestría en Inteligencia Artificial
Universidad Veracruzana
CIIA – Centro de Investigación en Inteligencia Artificial
Sebastián Camacho No 5, Xalapa, Ver., México 91000
arturomf94@gmail.com
https://github.com/arturomf94/aa-mia

14 de Diciembre del 2018

1. Introducción

Este proyecto final consiste en el desarrollo de un programa que resuelva el problema del agente viajero mediante el método de recocido simulado. El programa desarrollado aquí toma como input un número específico de puntos en un plano bidimensional que representan las *ciudades* del problema del viajero. El algoritmo trabaja con estos puntos para aproximar una solución al problema. A continuación se describe el problema del agente viajero, así como el método de recocido simulado. Después se hace una breve descripción de la implementación en Python de este método y por último se exponen algunos resultados de simulaciones.

2. Problema del Agente Viajero

El problema en su versión más general [1] podría enunciarse de la siguiente forma:

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 regresa a la ciudad origen?

[1] Esta introducción es la misma utilizada en [MF15].

Figura 1: TSP para 15 ciudades en Alemania

Desde una primera aproximación aún bastante teórica, el problema se puede proponer de la siguiente manera en el campo de la teoría de grafos:

Dado un grafo G ponderado y no-dirigido, ¿cuál es el ciclo hamiltoniano H más corto que existe?*

Un grafo, que es un conjunto de vértices y aristas, es ponderado si cada arista tiene un número asignado y es no-dirigido si no existe dirección particular entre sus vértices. Por otro lado, un ciclo hamiltoniano es una sucesión de aristas adyacentes que visitan a todos los vértices del grafo una sola vez, y además la primera arista es adyacente a la última. De tal forma que, en este su enunciado particular, los vértices serían ciudades, las aristas caminos entre ellas, y sus ponderaciones serían las respectivas distancias entre las ciudades; un ciclo hamiltoniano, por su parte, sería un camino completo, es decir, uno que pasa por todas las ciudades correspondientes y termina en la de origen. Un ejemplo de un grafo que cumple con estas condiciones se ilustra en la Figura 2:

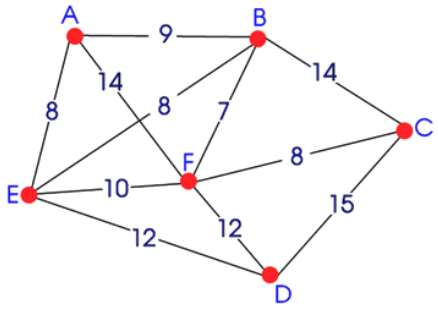

Figura 2: Un grafo ponderado y no-dirigido

En ocasiones se requiere que el grafo también sea completo [2]; es decir, que entre cada par de vértices exista una arista entre ellos. Sin embargo, esta condición no es crucial, pues a un grafo que no es completo uno puede agregarle la arista faltante con una ponderación arbitrariamente grande, de tal modo que no se afecta el ciclo hamiltoniano óptimo. El grafo de la Figura 2, por ejemplo, no es completo, pues no existe una arista entre B y D.

El problema del agente viajero es un problema computacionalmente complejo. En particular, el problema es clasificado como un problema *NP-completo*, por lo que aunque existan soluciones deterministas como aquel visto en [MF15] o métodos combinatorios, para variantes del problema con un número de nodos alto se recomienda utilizar heurísticas. Un buen método para su resolución es el de recocido simulado, que se expone en la siguiente sección.

3. Recocido Simulado

Recocido simulado es un algoritmo de búsqueda meta-heurística para problemas de optimización global; el objetivo general de este tipo de algoritmos es encontrar una buena aproximación al valor óptimo de una función en un espacio de búsqueda grande [Wik18].

El algoritmo comienza con una solución inicial al problema de optimización y en cada paso toma una solución *cercana* [3] a la solución previa. Se evalúa la bondad de esta nueva solución y, con base en eso, el algoritmo decide si quedarse o moverse con cierta probabilidad, que depende de si la nueva solución es mejor que la anterior o no. Progresivamente, la *temperatura* decrece, de manera que la probabilidad de que el algoritmo tome una peor solución decrece con el tiempo. Esta probabilidad decrece conforme a la siguiente expresión:

$$e^{\frac{-|d'-d|}{T}}$$

Donde d es la distancia de la ruta original, d' es la distancia de la nueva ruta y T es la temperatura sintética del problema.

4. Implementación en Python

El código puede encontrarse en este repositorio, donde además se incluyen unas imágenes que son los resultados de algunas simulaciones. Gran parte del código se basa en otro repositorio que puede encontrarse aquí Estas se encuentran en el folder llamado **result_images**. El script principal es aquel llamado **tsp.py**. Es aquí donde se especifican los parámetros del problema,

[2] Un grafo con n vértices que es completo usualmente se denota K_n.

[3] La noción de cercanía aquí puede variar, dependiendo de la naturaleza del problema. En el caso del PAV, una configuración cercana sería cambiar el orden en el camino de dos nodos adyacentes en la solución previa.

como el número de puntos que se quieren utilizar, así como las dimensiones del *mapa* y los parámetros del recocido simulado:

- **alpha** - La tasa de decrecimiento de la temperatura en cada paso.
- **temp** - La temperatura sintética inicial.
- **stopping_temp** - La temperatura límite.
- **stopping_iter** - El número de iteraciones límite.

Con los parámetros *alpha*, *temp* y *stopping_temp* se crea una retícula de parámetros de acuerdo con los rangos definidos. Para cada combinación de parámetos se realiza una corrida del programa en **simulated_annealing.py** y los resultados se guardan para futuro análisis.

La función *optimal_run* de **optimize.bf** se utiliza para obtener la mejor combinación de parámetros posible dentro de todas estas corridas. Optimiza primero la distancia obtenida, luego las iteraciones y finalmente el tiempo de ejecución.

Finalmente, para fines de comparación, se busca la solución mediante fuerza bruta con el script **brute_force.py**.

El repositorio también incluye dos scripts adicionales llamados **demo.py** y **demo2.py** hechos especialmente para replicar el ejemplo en [Lee].

5. Resultados

Los parámetros utilizados para las simulaciones fueron los siguientes:

- temp - De 100 a 5,000 en incrementos de 100.
- stopping_temp - Con valores 0.000000001, 0.00000001, 0.0000001, 0.000001, 0.00001.
- alpha de .01 a 1 en incrementos de .01.

Para este caso, se utilizaron 12 puntos aleatorios en un espacio de 200x200. Las Figuras 3, 4 y 5 muestran el rango de valores de los parámetros utilizados.

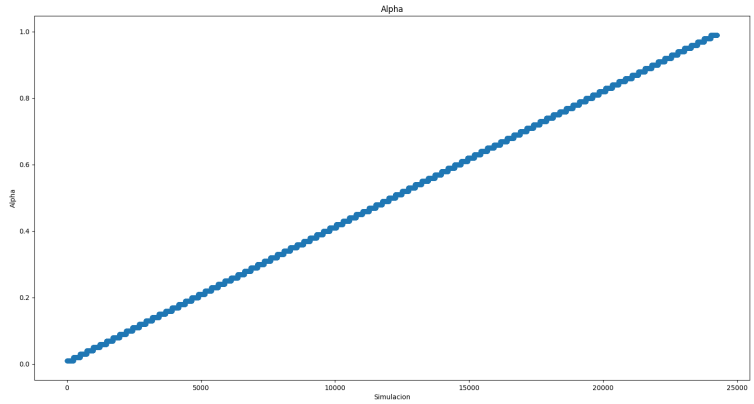

Figura 3: Alpha

Figura 4: Temperatura

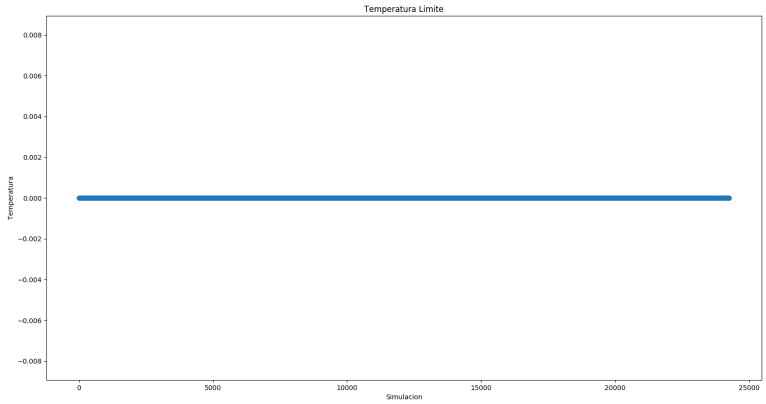

Figura 5: Temperatura Límite

De todas las combinaciones de estos parámetros, se obtuvieron las siguientes iteraciones y tiempos de ejecución, ilustrados en las Figuras 6 y 7.

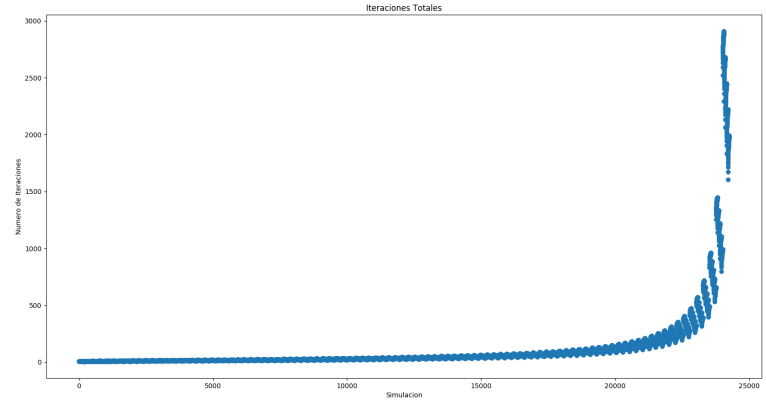

Figura 6: Iteraciones

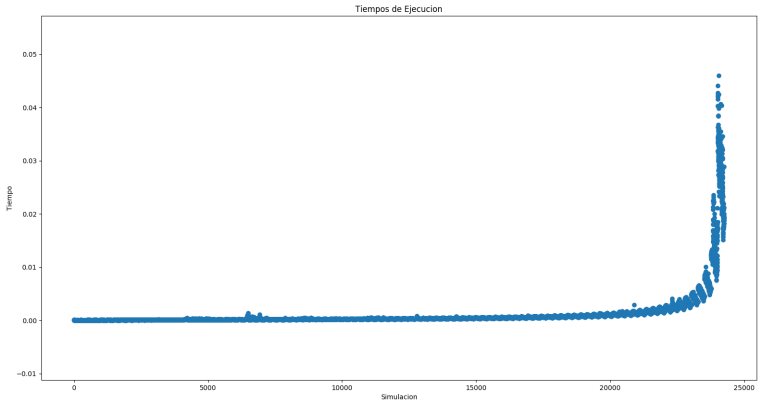

Figura 7: Tiempos de Ejecución

Por último, las distancias que se obtuvieron en cada una de las simulaciones se muestran en la Figura 8

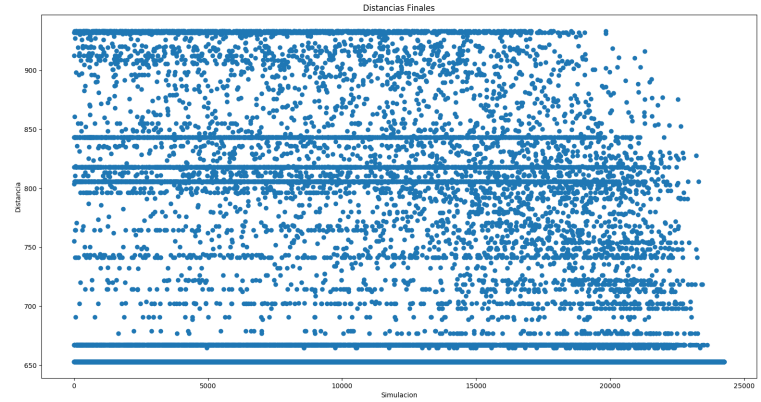

Figura 8: Distancias Finales

En cambio, en las Figuras 9, 10 y 11 se muestran las iteraciones, tiempos de ejecución y distancias finales en comparación con los de fuerza bruta:

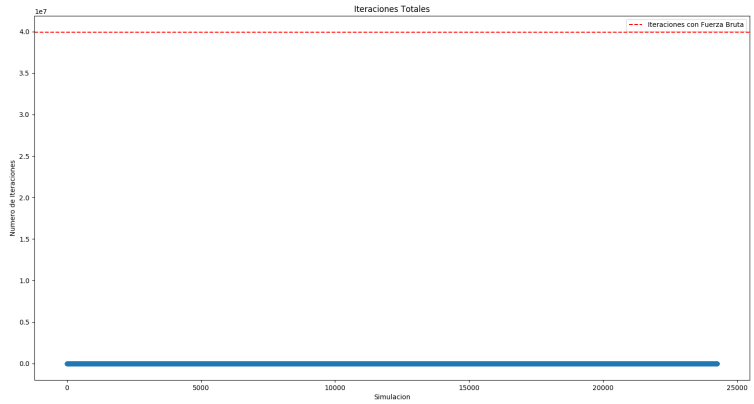

Figura 9: Iteraciones

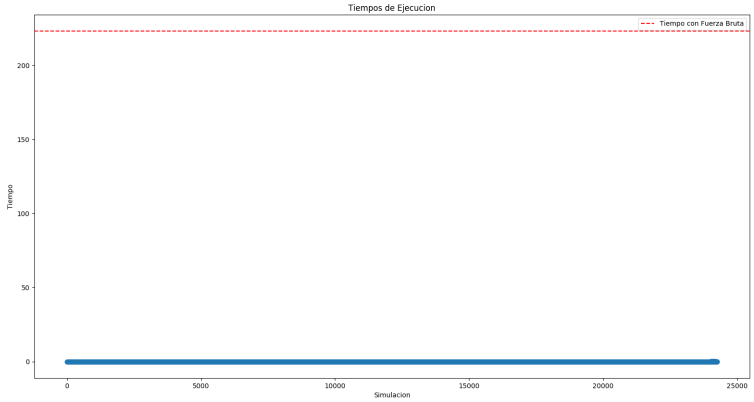

Figura 10: Tiempos de Ejecución

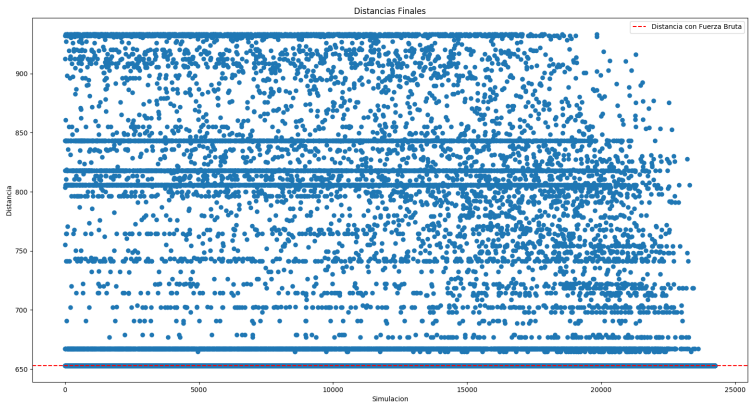

Figura 11: Distancias Finales

Evidentemente el método de recocido simulado tiene una ventaja en cuanto a la eficiencia del cómputo de una solución en este caso, y aunque las simulaciones en muchas ocasiones no alcanzan el óptimo global, los parámetros se pueden calibrar de tal manera que se minimize la probabilidad de obtener una ruta subóptima.

El método de optimización en el código es fácilmente engañado. Como podemos observar en las Figuras 6 y 7, los mejores valores para alpha son aquellos cercanos a 1. Sin embargo, utilizando el método de optimización en **optimize.py** obtenemos que $\alpha = 0{,}03$ es el óptimo. Si utlizamos esos parámetros en **demo.py** obtenemos los siguientes resultados:

- Distancia mínima: 930.66
- Iteraciones: 6
- Tiempo de Ejecución: 0.00038

Evidentemente, con un alpha tan pequeña, el cambio en temperatura es demasiado drástico y no se alcanza el óptimo global, pues no se explora lo suficiente el espacio de soluciones. Si, en cambio, optamos por los parámetros de [Lee], que especifican un alpha de .997 y temperatura de 10,000 entonces obtenemos los siguientes resultados y la ruta en la Figura 12.

- Distancia mínima: 890.22
- Iteraciones: 6,899
- Tiempo de Ejecución: 0.098

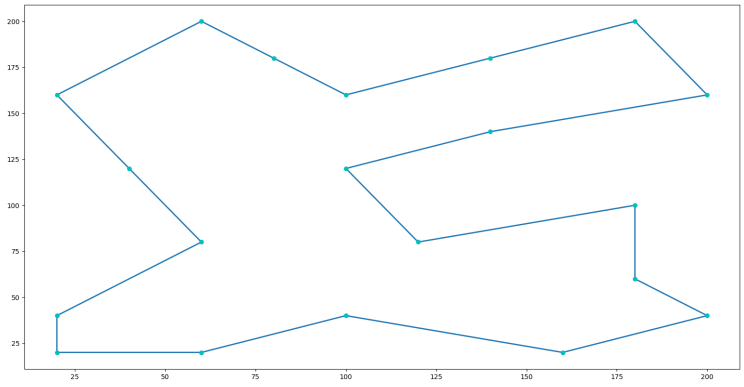

Figura 12: Ruta Final

Este resultado es incluso mejor que el que se indica en [Lee], que es de 911. Por último, en búsqueda de un resultado aún mejor, utilizamos un alpha de .999 y una temperatura de 50,000. Se obtuvieron los siguientes resultados y la ruta en la Figura 13.

- Distancia mínima: 871.12
- Iteraciones: 31,529
- Tiempo de Ejecución: .37

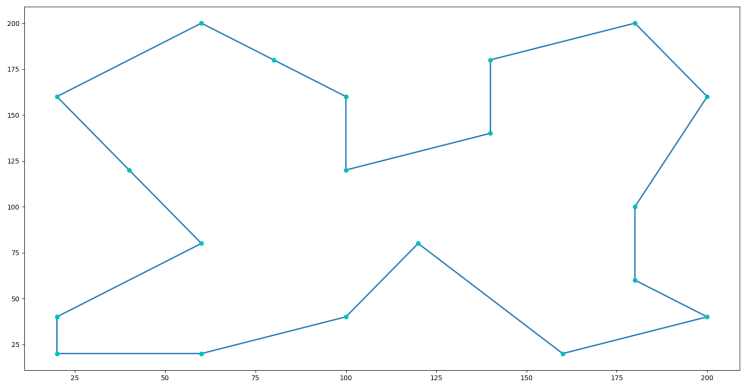

Figura 13: Ruta Final

Si este es el óptimo global entonces, juzgando por el número de iteraciones, el método de recocido simulado es mucho mejor que el de fuerza bruta, donde se habría que explorar el espacio de soluciones de tamaño de 121,645,100,400,000,000. La Figura 14 muestra el proceso de optimización de esta última simulación.

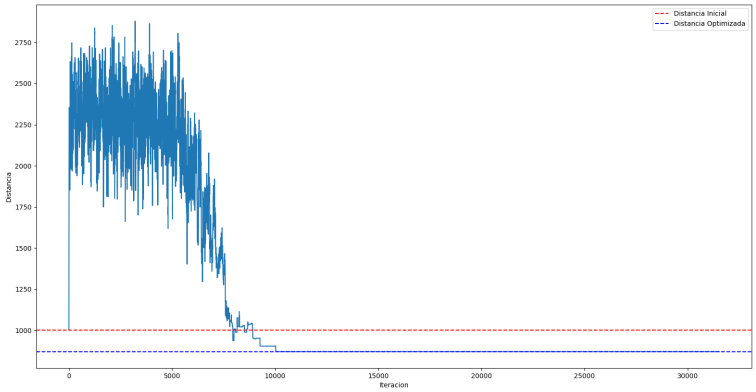

Figura 14: Optimización

Referencias

[Lee] Lee. Simulated annealing for beginners. http://www.theprojectspot.com/tutorial-post/simulated-annealing-algorithm-for-beginners/6. [Online; accessed 12-December-2018].

[MF15] Arturo Marquez-Flores. Problema del agente viajero: un enfoque de programación dinámica. *Laberintos & Infinitos*, (38):45–50, 2015.

[Wik18] Wikipedia. Algoritmo de recocido simulado — Wikipedia, the free encyclopedia. http://es.wikipedia.org/w/index.php?title=Algoritmo%20de%20recocido%20simulado&oldid=108084090, 2018. [Online; accessed 12-December-2018].