Resolución de TSP mediante algoritmos ACO

Antonio Sejas[1][0000−0002−8546−2209]

[1] Sistemas Multiagente - Julio 2020
[2] Máster Universitario en Inteligencia Artificial
[3] Universidad Politécnica de Madrid (UPM)

Abstract. En este reporte consiste en un análisis detallado de los algoritmos implementados para resolver dos escenarios del problema del viajante (*Travelling Salesman Problem* [1]). Comentaré todo el procedimiento realizado, haré hincapié en la selección de parámetros y terminaré con las conclusiones y futuros trabajos.

Keywords: Sistemas Multiagentes · Ant Colony Optimization · Travelling Salesman Problem.

1 El Problema

1.1 Travelling Salesman Problem

El problema del viajante o TSP por sus siglas en inglés, *Travelling Salesman Problem* [1], es un problema *NP-Hard* muy conocido y uno de los más utilizados para evaluar algoritmos de búsquedas combinatorias [3]. De forma muy breve podemos definir este problema como la búsqueda de la ruta más corta para que un *"vendedor"* recorra todas las ciudades de sus clientes saliendo desde su casa y regresando otra vez a su propia ciudad. Con la condición de que todas las ciudades deben ser visitadas una vez. No se pueden repetir ciudades y tampoco se pueden dejar ciudades sin visitar.

1.2 Ant Colony Optimization

ACO de sus siglas en inglés, *Ant Colony Optimization* es una familia de algoritmos metaheurísticos propuesto por primera vez por Marco Dorigo [2]. Actualmente existe una gran variedad de evoluciones del algoritmo original, las cuales son interesantes para explorar alternativas. ACO es un algoritmo bioinspirado en el proceso de búsqueda de comida que tienen las hormigas. Gracias al uso de feromonas para marcar rutas ya utilizadas y a la evaporación de dichas hormonas, emerge un proceso para optimizar la ruta desde su nido hasta donde se encuentra la comida.

1.3 Descarga de los dataset

Para garantizar la reproducibilidad del experimento descrito en este reporte, se adjunta los datasets junto al código. Estos datasets han sido obtenidos de la web de la competición *World TSP Tour*. http://www.math.uwaterloo.ca/tsp/world/countries.html. Para el desarrollo de los algoritmos he utilizado un dataset pequeño *Western Sahara* que cuenta con 29 ciudades. Y para la evaluación más profunda he cogido otros datasets con cientos de ciudades. Un gran beneficio de utilizar estos datasets es que todos ellos se encuentran en formato TSPLIB95 y pueden ser cargados mediante la librería de python con el mismo nombre [5]. Por último *World TSP Tour* nos ofrece el óptimo global con el que podremos evaluar cuan buena es la solución aportada por nuestro algoritmo. Siendo un proceso de evaluación riguroso. Agradecer a Nik, el profesor de Sistemas Multiagente, por facilitarnos este recurso.

2 Desarrollo

Todo el código está bajo la licencia OpenSource MIT y se encuentra disponible en el siguiente repositorio de GitHub https://github.com/sejas/muia-multiagent-systems-tsp-aco. Los algoritmos han sido implementados siguiendo el pseudocódigo que se encuentra en las transparencias de clase, optimizando ciertas partes del código para mejorar su rendimiento y legibilidad.

Al usar python he podido utilizar librerías muy potentes que han facilitado el desarrollo de la práctica. Pudieédome centrar casi únicamente en el algoritmo de deseado. **tsplib95**: Para cargar los datasets y calcular distancias. **random**: Para implementar las partes estocásticas del algoritmo. **pandas** y numpy: Para poder simplificar las operaciones entre matrices como si de números enteros se tratasen. **matplotlib**: Para mostrar los gráficos y trazar las rutas conseguidas.

3 Resultados

Antes de empezar a comentar los resultados y búsqueda de parámetros para optimizar los resultados de la búsqueda, quiero hablar de cómo he decidido representar las ciudades.

3.1 Representación

Podemos imaginarnos un grafo donde cada nodo es una ciudad y cada arista es la conexión que une dos ciudades. El peso de las feromonas en cada una de estas aristas la he guardado en una matriz de dimensiones NxN, donde N es el número de ciudades.

Visualización de feromonas: Dado que he utilizado la matriz completa y no solo el triángulo superior de la matriz, quiere decir que he asumido que el peso de las feromonas de i a j no es el mismo que de j a i. Con esto he conseguido un código un poco más limpio y

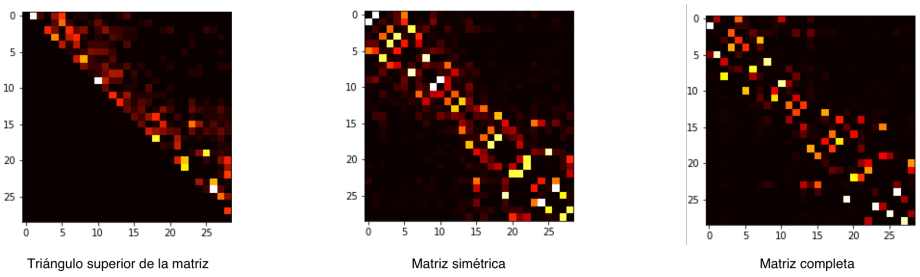

Fig. 1. Diferentes estrategias para modelizar las feromonas. Siendo el mejor el triángulo superior. En el siguiente vídeo podemos ver la evolución de las feromonas en el tiempo: https://youtu.be/Pz-RK9z26cQ

Visualización de rutas: Para visualizar las rutas he utilizado un gráfico scatter en el que he dibujado líneas uniendo los nodos de la ruta dada. A continuación podemos comprar el mejor de los resultados calculados contra el mejor de TSP World. Se aprecia que hay 3 nodos diferentes que son los que empeoran la ruta por 9 puntos.

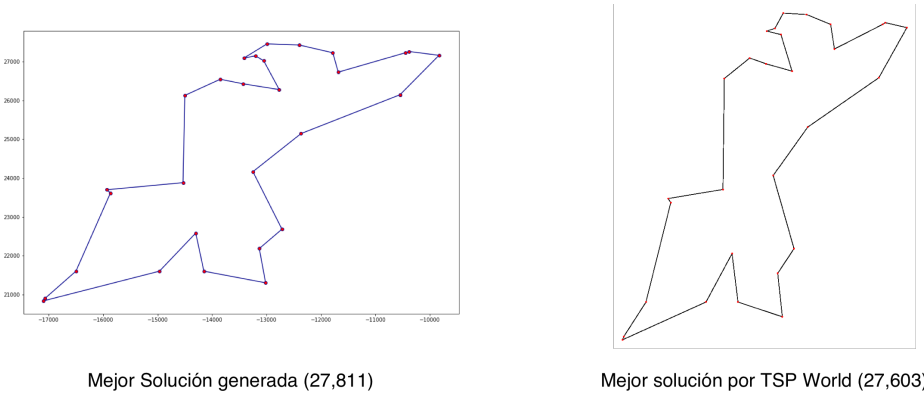

Fig. 2. Comparación de la ruta calculada de Western Sahara con la ruta de TSP World

3.2 Evaluación

Para calcular si el algoritmo está bien implementado, he decidido obtener el resultado de algoritmos básicos para tener un punto de referencia sobre la eficiencia de nuestro algoritmo. El secuencial, obtener una ruta uniendo todos los puntos por orden. El aleatorio, construir una ruta aleatoria. Y el heurístico que se basa en cada punto visitar la ciudad más cercana. Podemos apreciar que el heurístico pese a ser aparentemente una solución buena nos da un resultado el

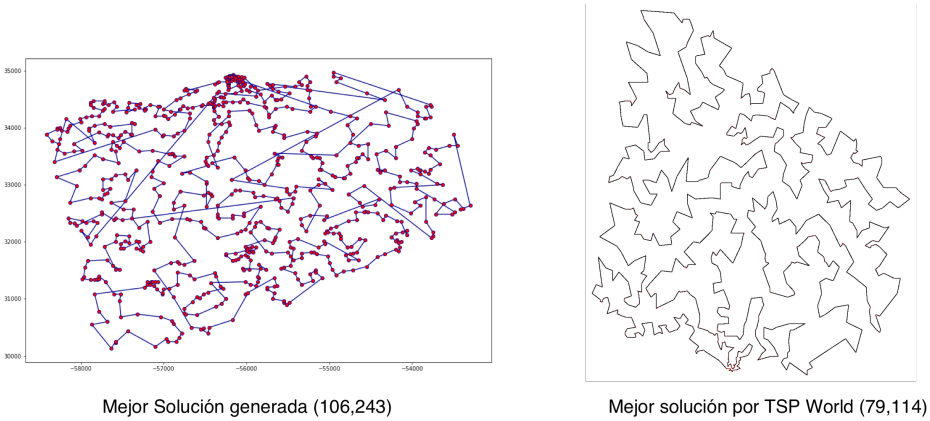

Fig. 3. Comparación de la ruta calculada de Uruguay con la ruta de TSP World

doble de ineficiente que el algoritmo más básico de ACO (Ant System). Por último podemos ver que con 100 iteraciones el algoritmo ya obtiene una solución muy próxima al óptimo global. Dato que hemos obtenido de Best World TSP.

Table 1. Evaluación de algoritmos.

Algoritmo	Peso (Western Sahara)	Tiempo (ms)	Peso (Uruguay)	Tiempo (ms)
Secuencial	52,284	1.37	844,742	7.81
Aleatorio	109,417	1.44	1,628,600	8.73
Heurístico	56,645	4.32	942,720	2430
Ant System	27,612	9190	106,243	22min 4s
Best World TSP	27,603	-	79,114	-

3.3 Optimización de parámetros

Para Western Sahara he utilizo una gran variedad de parámetros hasta dar con los adecuados. Las dimensiones del dataset han permitido una rápida ejecución que ha resultado fundamental para poder iterar y afinar el algoritmo. Algunas variaciones que he implementado han sido, distintas formas de guardar las feromonas. Ejecuciones con 1 y

M_ANTS : Número de hormigas, normalmente coincide con el número de nodos (N); **ALPHA** : Normalmente 1 . Es el coeficiente de feromonas.; **BETA** : Coeficiente heurístico, normalmente un valor entre [2,5]. He probado con 0 a 6.; **RO** : Tasa de evaporación. Es como el enfriamiento en el recocido simulado. Un valor alto es similar a disminuir la temperatura drásticamente y quedarse atascado en un óptimo local.; **Q** :Factor de cambio de la feromona; **TAU_INITIAL** : Feromona inicial; **STEPS** : Número de iteraciones;

Cada uno de estos parámetros es fundamental para jugar con el equilibrio entre exploración y explotación. Además de que el número de hormigas y número iteraciones tiene una consecuencia drástica en el tiempo de cómputo. En el código se puede ver las diferentes estrategias adoptadas.

4 Conclusiones

Implementar desde cero los algoritmos de ACO ha sido una experiencia didáctica muy enriquecedora. Realizar algo por uno mismo siempre aporta una visión más profunda sobre la materia que utilizar código ya desarrollado. La optimización de parámetros es como siempre una de los puntos más delicados y como siempre dependiente del problema. Es por ello que esta optimización supone un área por si misma dentro de la IA. Por último hasta el último momento no he sido consciente del tiempo de ejecución del algoritmo en datasets muy grandes. He probado a ejecutar el algoritmo con 4.000 ciudades y tras más de una hora no ha completado el primer step. Ni en mi máquina ni en Google Colab. De este modo, soy aún más consciente de lo que significa la dificultad polinómica.

4.1 Futuras investigaciones

Tras leer varios artículos de clasificación no supervisada bioinspirados en el comportamiento social de las hormigas. Me he dado cuenta que ninguno de ellos toma las fortalezas del algoritmo clásico de ACO, y por el contrario utilizan otros tipos de comportamientos como el apilamiento de cuerpos o la aceptación por el olor de la colonia y su propio genoma [4]. Por este motivo considero interesante explorar un algoritmo clásico que una todos los puntos del dataset y posteriormente romper las conexiones que superen un determinado valor, como puede ser la media de las distancias o 2 veces la desviacion típica. Con esto conseguiremos islas de putnos conectados y al mismo tiempo outliers.

References

[1] David L Applegate et al. *The traveling salesman problem: a computational study*. Princeton university press, 2006.

[2] Marco Dorigo and Gianni Di Caro. "Ant colony optimization: a new meta-heuristic". In: *Proceedings of the 1999 congress on evolutionary computation-CEC99 (Cat. No. 99TH8406)*. Vol. 2. IEEE. 1999, pp. 1470–1477.

[3] David S Johnson and Lyle A McGeoch. "The traveling salesman problem: A case study in local optimization". In: *Local search in combinatorial optimization* 1.1 (1997), pp. 215–310.

[4] Nicolas Labroche, Nicolas Monmarché, and Gilles Venturini. "Visual clustering with artificial ants colonies". In: *International Conference on Knowledge-Based and Intelligent Information and Engineering Systems*. Springer. 2003, pp. 332–338.

[5] Gerhard Reinelt. *TSPLIB. University of Heidelberg*. 1996.