Taller 1: Inteligencia Artificial:Representación y Solución de problemas

Naigel Christopher Herrera
Maestría en MACC
Universidad del Rosario
Bogotá, Colombia
Naigel.Herrera@urosario.edu.co

***Abstract*—En este articulo se analizan 3 métodos de búsqueda para resolver el problema del vendedor viajero. Se analiza el rendimiento de cada método en función de los tiempos de CPU y el numero de ciudades a considerar.**

***Index Terms*—estados, heurística, búsqueda.**

I. Introducción y contexto

El problema del vendedor viajero, también conocido como Travelling Salesman Problem (TSP) o por sus siglas TSP, plantea la incógnita de un individuo que debe viajar entre un conjunto de ciudades. Este vendedor cuenta con la distancia entre cada par de ciudades y su objetivo es determinar la ruta más corta posible que visite cada ciudad exactamente una vez antes de regresar al punto de partida. El TSP es un desafío clásico en la optimización combinatoria y tiene aplicaciones en logística, planificación de rutas y diseño de circuitos electrónicos [1].

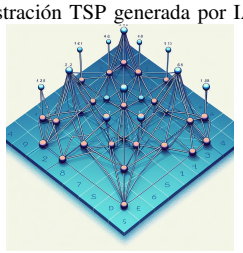
El problema del vendedor viajero fue definido en el siglo XIX por los matemáticos W.R. Hamilton y Thomas Kirkman. Sin embargo, su estudio más profundo se llevó a cabo durante la década de los años 30 por matemáticos de Harvard y la Universidad de Viena. Aunque el matemático Karl Menger fue el primero en formalizar el problema, su nombre fue popularizado por Hassler Whitney de la Universidad de Princeton. Este desafío clásico sigue siendo objeto de investigación y aplicación en diversas áreas de la optimización combinatoria [2].

En el problema del vendedor viajero, se representan N! rutas posibles, donde N es el número de ciudades. Sin embargo, podemos simplificar esto considerando que una ruta nos da el punto de partida, lo que reduce la dimensión a (N-1)!. Además, dado que no importa la dirección del viajante, el número de rutas se reduce a la mitad, es decir, (N-1)!/2 rutas posibles. Cada ciudad incrementa el número de rutas por un factor de N de manera factorial. Por ejemplo, para un problema con 5 ciudades, tendremos 12 rutas posibles, mientras que al llegar a 10 ciudades, tendremos 181,440 rutas diferentes. Esto convierte al problema en un desafío computacionalmente complejo [3].

Este problema puede ser modelado y abordado de diferentes maneras, tradicionalmente los métodos utilizados son:

- Formulación algoritmos para encontrar soluciones exactas (Dimensionalidad pequeña).
- Formular heurísticas que nos permitan hallar un solución pero no necesariamente óptima.
- Creación de heurísticas específicas para resolver casos especiales de forma exacta.

Fig. 1. Ilustración TSP generada por IA



El vendedor viajero cuenta con las siguientes propiedades del entorno:

- Completamente observable: el agente cuenta con toda la información del entorno.
- Secuencial: El recorrido se realiza mediante una secuencia de acciones.
- Agente único: Solo existe un agente en el entorno.
- Estático: El entorno solo se ve afectado por el agente.
- Discreto: El numero de estados es finito.
- Recorrido cerrado: El recorrido termina en la casilla inicial.

Como definimos anteriormente, la complejidad del problema se ve afectada por el número de ciudades a considerar, haciendo necesario simplificar más el problema a medida que crece el número de las mismas. La solución más directa, que es un algoritmo de fuerza bruta, está limitada a menos de 19 ciudades debido a los grandes recursos computacionales que necesitaría para hallar las soluciones. Por lo tanto, se aplicaron algoritmos como el de Held-Karp, que puede resolver el problema. También existen otros métodos, como los algoritmos de ramificación y acotación, que pueden utilizarse para el TSP de más de 40 ciudades. El experimento más grande incluyó 85,900 ciudades [4].

En este articulo, abordaremos el problema del Vendedor viajero mediante la creación de heurísticas, adicionalmente utilizaremos un algoritmo de búsqueda a ciegas para comparar su desempeño. Estas soluciones están dirigidas a problemas de dimensiones menores a 19 ciudades debido al costo computa-

cional que estas requieren. Consideraremos distinto numero de ciudades y estados iniciales del problema.

II. Métodos de Solución

Estado Inicial: El viajero inicia su recorrido desde una ciudad asignada. Él conoce el costo de viajar entre cada una de las ciudades y solo puede visitarlas una vez hasta volver al punto de partida.

En la implementación del algoritmo, este grupo de ciudades está representado como una lista que contiene sus nombres. Cada ciudad cuenta con dos atributos: las rutas de viaje directo hacia las otras ciudades del conjunto y sus coordenadas, las cuales estan almacenadas en diccionarios que tienen como clave sus nombres. El estado inicial del problema se presenta entonces como una lista llamada “Estado”, que representa las ciudades ya visitadas. El valor de arranque es la ciudad que se introduce como punto de partida.

Las ciudades no visitadas se definen como acciones posibles y son aquellas que se tienen en cuenta para el cálculo de la heurística en cada iteración. Todas las ciudades de inicio tienen un valor de solución, esto no garantiza que este valor de solución se igualmente óptimo para todo el conjunto.

Acciones aplicables: Inicialmente como acciones aplicables tenemos una lista de ciudades, estas son las que no se encuentran incluidas en la lista ”Estado” que guarda aquellas que ya fueron visitadas.

Acciones aplicables: Inicialmente como acciones aplicables tenemos una lista de ciudades, estas son las que no se encuentran incluidas en la lista ”Estado” que guarda aquellas que ya fueron visitadas.

Función de transición: Se genera una copia del estado actual, se modifica con agregando la nueva ciudad visitada, resultando en una lista que contiene las paradas anteriores y la actual en la ultima posición.

Función de costo: Se han definido dos funciones de costo. La primera calcula el costo del viaje directo entre dos ciudades. Esta información se encuentra en un diccionario de diccionarios que contiene todas las rutas posibles por ciudad. Como segunda función de costo, tenemos la distancia euclidiana entre ciudades, la cual se calcula con sus coordenadas.

Algoritmos: Para este problema se realizaron dos algoritmos de búsqueda: búsqueda en profundidad y búsqueda ávida. Para este segundo método, se crearon dos heurísticas.

- Búsqueda en profundidad: Este algoritmo se utiliza para recorrer todos los nodos de manera ordenada, posteriormente se regresa a estados anteriores y este comienza a recorrer nuevos caminos.
- Búsqueda avara: Este algoritmo de búsqueda funciona guiado por heurísticas, intentando encontrar la ruta mas corta desde el punto inicial hasta el objetivo. Para este caso utilizamos como heurística, el costo de viaje entre dos ciudades y su distancia euclidiana.

III. Resultados

Luego de haber implementado los algoritmos en Python, obtenemos los siguientes resultados representados en un diagrama de cajas y bigotes (Figura 2). En este análisis, comparamos los tiempos de CPU de los tres métodos utilizados. Utilizamos un problema de 12 ciudades como base para nuestra evaluación.

Aquí están los hallazgos clave:

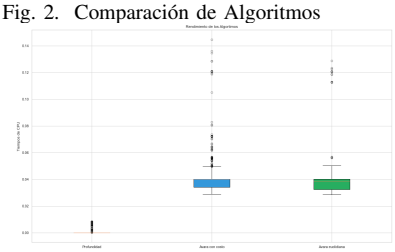
- Búsqueda en profundidad: El algoritmo de búsqueda en profundidad muestra los menores tiempos de ejecución. Esto significa que es más eficiente en términos de velocidad de procesamiento.
- Algoritmos guiados por heurística: Estos algoritmos presentan un comportamiento similar entre sí, con tiempos de ejecución considerablemente mayores que la búsqueda en profundidad. Aunque no son tan rápidos como la búsqueda en profundidad, siguen siendo comparables entre sí.

En resumen, si la prioridad es la velocidad de ejecución, elige la búsqueda en profundidad. Si se necesita un equilibrio entre eficiencia y precisión, los algoritmos guiados por heurística pueden ser una buena opción.

Por otro lado, en la búsqueda guiada por el costo, se pueden evidenciar muchos valores atípicos (outliers). Estos valores atípicos están acompañados de un rango intercuartil más corto, lo cual denota una menor variación dentro de los intervalos considerados normales.

En contraste, en la búsqueda en profundidad, vemos una mínima variación si la comparamos con los otros métodos. Esto sugiere que la búsqueda en profundidad es más consistente en términos de tiempos de ejecución.

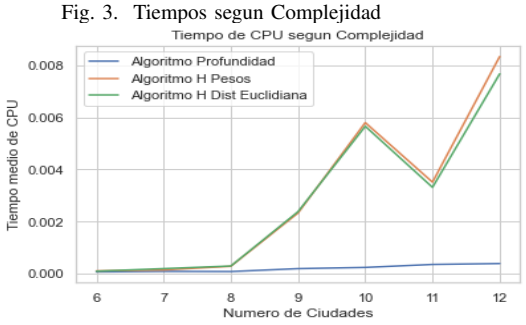
Fig. 2. Comparación de Algoritmos



A partir de la Figura 3, observamos un notable aumento en el tiempo medio de CPU para los algoritmos basados en heurísticas. Este incremento se debe al crecimiento factorial del número de soluciones y a la carga computacional que implica un aumento en la dimensionalidad del problema. En otras palabras, a medida que aumenta la complejidad del problema, los algoritmos heurísticos requieren más tiempo de procesamiento.

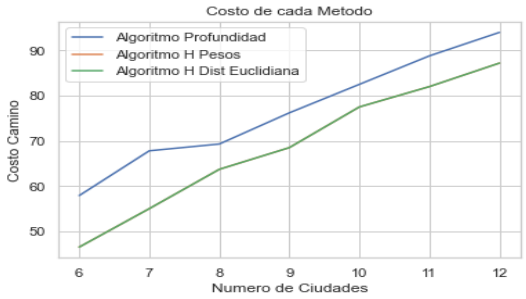
Por otro lado, la búsqueda en profundidad no experimenta un aumento significativo en sus tiempos de ejecución. Esto sugiere que la búsqueda en profundidad es más robusta frente a cambios en la dimensionalidad del problema.

Fig. 3. Tiempos segun Complejidad



De acuerdo con la Figura 4, podemos observar que la solución de menor costo es la encontrada por los algoritmos guiados por heurística. Además, notamos cierta consistencia en la diferencia de costos a medida que aumentamos el número de ciudades en cuestión. Esta consistencia podría ser un factor clave para determinar que algoritmo puede ser la mejor opción si comparamos el tiempo de CPU con el costo de la ruta.

Fig. 4. Tiempos según Complejidad



IV. Conclusiones

- Los algoritmos de búsqueda guiados por heurística presentan soluciones con un costo menor si lo comparamos con búsqueda en profundidad.
- El tiempo de CPU aumenta de manera drástica para los algoritmos guiados por heurística, mientras que para el algoritmo de búsqueda en profundidad se mantiene más constante.
- La diferencia en costo de las rutas halladas por los algoritmos de búsqueda se mantiene constante a medida que aumentamos el número de ciudades. Sin embargo, aquellos guiados por heurísticas logran mejores soluciones, aunque con un alto costo computacional.

Podemos concluir que los algoritmos guiados por heurísticas son los más adecuados para nuestro caso de estudio. Estos algoritmos encuentran la solución más óptima con tiempos razonables para la dimensionalidad que estamos estudiando en este artículo. Sin embargo, para implementaciones con una dimensionalidad mayor, podríamos prescindir de la mejor solución en términos de costo. En este caso, la búsqueda en profundidad, con sus menores tiempos de ejecución, sería la opción más viable.

Referencias

[1]Espinoza, D. (2006). EL Problema del Vendedor Viajero (TSP) y Programación Entera (IP). Universidad de Chile. Facultad de Ciencias Físicas y Matemáticas. Departamento de Ingeniería Industrial.

[2]Pérez Rave, J. I., Jaramillo Álvarez, G. P. (2013). Espacio literario relevante sobre el problema del vendedor viajero (TSP): contenido, clasificación, métodos y campos de inspiración. Production, 23, 866-876.

[3]Larranaga, P., Kuijpers, C. M. H., Murga, R. H., Inza, I., Dizdarevic, S. (1999). Genetic algorithms for the travelling salesman problem: A review of representations and operators. Artificial intelligence review, 13, 129-170.

[4]Arora, S. (1998). Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. Journal of the ACM (JACM), 45(5), 753-782.