Algoritmos BIOINSPIRADOS parA el problema del agente viajero

­ BIOINSPIRATED ALGORITHMS FOR TRAVELLING SALESMAN PROBLEM TSP

Oraldo Jacinto Simón1,

*Marco Emanuel Casavantes Moreno2*

Universidad Autónoma de Chihuahua (UACH), México, ory10887@gmail.com

Universidad Autónoma de Chihuahua (UACH), México, marcocasavantes@gmail.com

Resumen:El problema del Viajante de Comercio, del inglés Traveling Salesman Problem o también conocido como TSP, debe su nombre a la comunidad científica (1931-1932) de la Universidad de Pincenton, donde se tomó el problema desde un punto de vista matemático. El Problema del Viajante de Comercio es uno de los problemas más complejos que se conoce de la programación matemática actual por su complejidad computacional, estando clasificado dentro de aquellos problemas considerados NP-Hard. Hasta la fecha no se ha encontrado un algoritmo exacto que encuentre la solución óptima a este problema en un tiempo polinomial. La importancia del TSP no sólo radica en la cantidad de aplicaciones que tiene, sino que, también, la investigación realizada sobre el TSP es fácilmente aplicable a otros problemas de rutas que se suelen generalizar a este último [1]. El Algoritmo Genético (AG) y el Algoritmo Colonia de Hormigas (ACH) son dos algoritmos bioinspirados que han tenido éxito obteniendo buenas soluciones aproximadas a problemas de optimización combinatoria. En el presente trabajo se aplica AG y el ACH al problema del TSP simétrico mostrando los resultados obtenidos durante la experimentación.

Palabras Clave: TSP, Algoritmo genético, TSP.

AbstracT: The Traveling Salesman Problem, also known as the TSP, owes its name to the scientific community (1931-1932) of the University of Pincenton, where the problem was taken from a mathematical point of view. The Traveling Salesman Problem is one of the most complex problems that is known about current mathematical programming due to its computational complexity, being classified within those problems considered NP-Hard. To date, no exact algorithm has been found that finds the optimal solution to this problem in a polynomial time. The importance of the TSP not only lies in the number of applications it has, but also, the research carried out on the TSP is easily applicable to other routing problems that are usually generalized to the latter. The Genetic Algorithm (AG) and the Ant Colony Optimization algorithm (ACO) are two bioinspired algorithms that have been successful in obtaining good approximate solutions to combinatorial optimization problems. In the present work the AG and the ACH are applied to the problem of the symmetric TSP showing the results obtained during the experimentation.

KeyWords:TSP, Genetic Algorithms, ACO

1. INTRODUCCIÓN

Los problemas de optimización combinatoria aparecen por la necesidad que muestran ciertos sectores empresariales, como los relacionados con procesos de fabricación, logística, problemas de localización, planificación de tareas, etc. Su fin principal es la optimización de uno o varios objetivos, bajo grandes números de posibles soluciones. Este efecto hace que sea necesaria la implementación de algoritmos de resolución, capaces de dar soluciones en tiempos computacionales razonables.

El problema del viajante de comercio puede resumirse en pocas palabras: un comerciante quiere visitar *n* ciudades empezando y terminando en su punto de origen teniendo en cuenta que el recorrido a seguir debe cumplir como requisito que su distancia total sea la mínima de todas las posibles combinaciones. La importancia del TSP no sólo radica en la cantidad de aplicaciones que tiene, sino que, también, la investigación realizada sobre el TSP es fácilmente aplicable a otros problemas de rutas que se suelen generalizar a este último. Un ejemplo de ello lo encontramos en problemas de rutas con recogida y entrega de mercancías [1].

**¿Qué es el Traveling Salesman Problem (TSP)?**

Es un problema matemático popular que solicita la trayectoria posible más eficiente dado un conjunto de puntos y distancias, donde todos los puntos deben ser visitados. El problema recibe como entrada una lista de ubicaciones físicas o nodos del sistema, junto con información de distancia, y en ciencias de la computación tiene aplicaciones para identificar métodos de optimización de red o hardware. Por ejemplo, se puede utilizar para elaborar las trayectorias más eficientes para los paquetes de datos que se enrutan a cualquier parte de una red, ya sea pública o privada [2].

En el caso concreto del TSP la entrada es el número de nodos (o ciudades) del grafo. Cuanto mayor sea el número de nodos, mayor va a ser el número de rutas posibles, y por lo tanto mayor será el esfuerzo requerido para calcular todas ellas. Así, ¡el número de rutas posibles entre N nodos es igual a N!, lo que hace que la resolución del TSP mediante la obtención de todas las rutas posibles y posterior comparación entre ellas (que es el primer algoritmo que a uno se le ocurre para dar solución) es poco factible incluso para un número de nodos no muy elevado [3].

Basta una red simple de 7 ciudades para que sea necesario calcular más de 5000 combinaciones es de *(7! =5040),* y si subimos hasta 10 el número de ciudades de la red, entonces las posibles rutas se disparan hasta más de tres millones *(10! = 3.628.800)*. Por ello, intentar resolver el TSP en redes relativamente simples mediante el método de generación y comparación de todas las rutas es absolutamente inabordable mediante los medios computacionales disponibles actualmente, lo que hace que sea necesario utilizar otros procedimientos que, aunque no obtengan la solución óptima, sí proporcionen una respuesta aproximada lo suficientemente optimizada en un tiempo razonablemente bajo.

En este trabajo se aborda el problema del viajante de comercio, resolviéndolo a través de la aplicación paralela de un algoritmo genético simple, el algoritmo de optimización con colonia de hormigas (con una modificación usando árbol de expansión mínima) y un algoritmo genético simple con heurísticas implementados en lenguaje python.

1. Metodología

En este trabajo se propone un algoritmo genético simple para resolver el problema del TSP, partiendo de los resultados de dicho algoritmo se le incluyen dos heurísticas para mejorar su rendimiento. Se le da solución al mismo problema con el algoritmo de optimización colonia de hormigas tomado de la siguiente dirección [4]. Después de aplicar dichos algoritmos se realiza un análisis estadístico a sus resultados para determinar cuál solución es mejor para el problema del TSP.

En este proyecto, se abordó un problema de 17 ciudades propuesto en el conjunto de datos GR17 (Groetschel), perteneciente a la librería TSPLIB [5], una colección de conjuntos de datos de problemas de TSP mantenidos por Gerhard Reinelt. GR17 es un problema TSP Simétrico (la distancia entre ciudades es la misma de ida que de regreso) y es posible viajar de cualquier ciudad “A” a cualquier ciudad “B”.

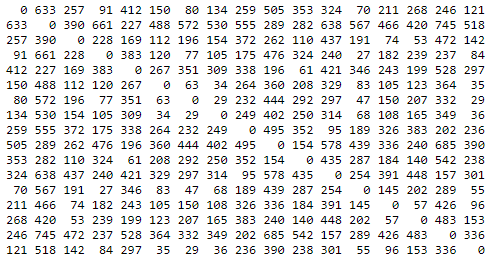


Figura 1. Matriz de distancias entre nodos en GR17

**2.1. ALGORITMO GENÉTICO (AG)**

Los algoritmos genéticos de operaciones (AG) son métodos adaptativos que pueden usarse para resolver problemas de búsqueda y optimización. Se encuentra en el proceso genético de los organismos vivos. A lo largo de generaciones, las poblaciones evolucionan en la naturaleza de acuerdo con los principios de la selección natural y la supervivencia de los más fuertes, postulados por Darwin (1859). Por imitación en este proceso, los algoritmos genéticos pueden crear soluciones para los problemas del mundo real. La evolución de estas soluciones hacia valores óptimos del problema en gran medida de una codificación adecuada de ellos. Los principios básicos de los algoritmos genéticos fueron establecidos por Holland (1975) y están bien descritos en varios textos: Goldberg (1989), Davis (1991), Michalewicz (1992), Reeves (1993) [6].

En la naturaleza los individuos de una población compiten entre sí en la búsqueda de recursos tales como comida, agua y refugio. Incluso los miembros de una misma especie compiten a menudo en la búsqueda de un compañero. Aquellos individuos que tienen más éxito en sobrevivir y en atraer compañeros tienen mayor probabilidad de generar un gran número de descendientes. Por el contrario, individuos poco dotados producirán un menor número de descendientes. Esto significa que los genes de los individuos mejor adaptados se propagarán en sucesivas generaciones hacia un número de individuos creciente. La combinación de buenas características provenientes de diferentes ancestros, puede a veces producir descendientes más aptos, cuya adaptación es mucho mayor que la de cualquiera de sus ancestros. De esta manera, las especies evolucionan logrando unas características cada vez mejor adaptadas al entorno en el que viven [7].

Los **Algoritmos Genéticos** trabajan sobre una población de individuos. Cada uno de ellos representa una posible solución al problema que se desea resolver. Todo individuo tiene asociado un ajuste de acuerdo a la bondad con respecto al problema de la solución que representa (en la naturaleza el equivalente sería una medida de la eficiencia del individuo en la lucha por los recursos). El AG propuesto consta de los siguientes elementos para su funcionamiento:

* Una **población** *P* inicial de n individuos, donde cada individuo es una solución.
* Una **función objetivo** *f* conocida como función de fitness o simplemente fitness, la cual evaluará la calidad del individuo.

Una codificación de individuo, la cual permitirá adaptar las posibles soluciones al dominio del problema.

A continuación, se explican los operadores que se ejecutan en cada generación o época.

**2.1.1. SELECCIÓN**

Este operador selecciona los individuos que se van a reproducir en base a su función de fitness. La idea principal de este método consiste en realizar la selección en base a comparaciones directas entre individuos. Existen dos versiones de selección mediante torneo y en el método de selección propuesto fueron implementados los dos:

* Determinística
* Probabilística

En la versión **determinística** se selecciona al azar un número p de individuos (generalmente se escoge *p=2*). De entre los individuos seleccionados se selecciona el más apto para pasarlo a la siguiente generación.

La versión **probabilística** únicamente se diferencia en el paso de selección del ganador del torneo. En vez de escoger siempre el mejor se genera un número aleatorio del intervalo *[0...1]*, si es mayor que un parámetro p (fijado para todo el proceso evolutivo) se escoge el individuo más alto y en caso contrario el menos apto. Generalmente *p* toma valores en el rango (*0.5 < p 1)*.

**2.2.2. CRUCE**: Este operador realiza un intercambio de genes entre dos individuos seleccionados.

Una vez seleccionados los individuos, éstos son recombinados para producir la descendencia que se insertará en la siguiente generación. Tal y como se ha indicado anteriormente el cruce es una estrategia de reproducción sexual.

Existen multitud de algoritmos de cruce como cruce uniforme, cruce de un punto y cruce de dos puntos. En el algoritmo propuesto se utilizó el cruce de un punto.

**Cruce de un punto**

Es la más sencilla de las técnicas de cruce. Una vez seleccionados dos individuos se cortan sus cromosomas por un punto seleccionado aleatoriamente para generar dos segmentos diferenciados en cada uno de ellos: la cabeza y la cola. Se intercambian las colas entre los dos individuos para generar los nuevos descendientes. De esta manera ambos descendientes heredan información genética de los padres, tal y como puede verse en la Tabla 2.

**Padre1** **Hijo1**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | A | B | C | D | E | |  | |  |  |  |  |  | | --- | --- | --- | --- | --- | | A | B | 3 | 4 | 5 | |
| **Padre2**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 1 | 2 | 3 | 4 | 5 | |  | **Hijo2**   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 1 | 2 | C | D | E | |

Tabla 1. Cruce de un punto

Al realizar el cruzamiento de un punto sin tener en cuenta donde se insertan los nuevos genes en el nuevo estamos insertando la posibilidad que el nuevo hijo no sea lo suficientemente apto. Con el objetivo de mejorar la aptitud de los nuevos hijos producto del cruzamiento, se inserta la siguiente heurística.

**Heurística agregada al cruzamiento**

Como entrada al cruzamiento contamos con dos individuos: Padre1, Padre2

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Padre1  0 1 2 3 4   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 5 | 3 | 2 | 4 | 1 | | Padre2   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 5 | 3 | 1 | 2 | 4 | |

El primer paso es generar una posición aleatoria desde la posición 1 hasta la penúltima posición de del Padre 1. Se forma un subarrreglo del Padre 1 desde el cero hasta el valor generado.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Padre1  0 1 2 3 4   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 5 | 3 | 2 | 4 | 1 | | Valor aleatorio: 2  Subarreglo de 0 a 2   |  |  |  | | --- | --- | --- | | 5 | 3 | 2 | |

Luego tratamos de insertar cada elemento del Padre 2 que no esté en el Padre 1, pero al insertar el elemento se valoran todas las posibles posiciones y se inserta en la **posición más favorable**.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Subarreglo  0 1   |  |  |  | | --- | --- | --- | | 5 | 3 | 2 | | Se intenta insertar el elemento 4 en la posición más favorable y se tienen en cuenta las siguientes combinaciones |
| Posibles opciones     |  |  |  |  | | --- | --- | --- | --- | | 4 | 5 | 3 | 2 |      |  |  |  |  | | --- | --- | --- | --- | | 5 | 4 | 3 | 2 |  |  |  |  |  | | --- | --- | --- | --- | | 5 | 3 | 4 | 2 |  |  |  |  |  | | --- | --- | --- | --- | | 5 | 3 | 2 | 4 | | Comparación de distancias  Distancia de 4 a 5 = 6  Distancia de 4 a 2 = 3  Distancia total = 9    Distancia de 5 a 4 = 6  Distancia de 4 a 3 = 4  Distancia total = 10  Distancia de 3 a 4 = 4  Distancia de 4 a 2 = 3  **Distancia total = 7**  Distancia de 2 a 4 = 3  Distancia de 4 a 5 = 6  Distancia total = 9 |

Tabla 2. Heurística en cruzamiento

Luego de analizar las posibles soluciones se toma la posición de menor distancia por lo tanto el valor se inserta entre los valores de menor distancia total en este caso entre 3 y 2. Esto se ejecuta para todos los genes que se necesiten cruzar.

**2.2.3. MUTACIÓN**: Modifica el material genético de un individuo de manera aleatoria.

La mutación de un individuo provoca que alguno de sus genes, generalmente uno sólo, varíe su valor de forma aleatoria.

Aunque se pueden seleccionar los individuos directamente de la población actual y mutarlos antes de introducirlos en la nueva población, la mutación se suele utilizar de manera conjunta con el operador de cruce. Primeramente, se seleccionan dos individuos de la población para realizar el cruce. Si el cruce tiene éxito entonces uno de los descendientes, o ambos, se muta con cierta probabilidad. Se imita de esta manera el comportamiento que se da en la naturaleza, pues cuando se genera la descendencia siempre se produce algún tipo de error, por lo general sin mayor trascendencia, en el paso de la carga genética de padres a hijos.

0 1 2 3 4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |

Valor aleatorio: **2**

**Resultado:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 4 | 3 | 5 |

Tabla 2. Mutación aleatoria de un hijo

**Heurística agregada a la mutación**

En la mutación simple los genes eran intercambiados aleatoriamente. Con esta nueva heurística el gen es intercambiado a la posición más favorable de todo el individuo siguiendo el mismo principio que se empleó a la heurística insertada en el cruzamiento.

Se genera un valor aleatorio para seleccionar el gen a mutar y si es mayor que el rate de mutación se inserta el gen seleccionado en la posición más favorable o de menor costo.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Nuevo hijo  0 1 2 3 4   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 5 | 3 | 2 | 4 | 1 | |  |

Posición más favorable

Hijo mutado

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 3 | 1 | 2 | 4 |

**REINSERCIÓN:** Inserta en la población inicial los hijos más aptos sustituyéndolos por los padres que menos aptitudes tienen en la población, esto ocurre de manera aleatoria. Cada uno de los hijos es comparado con un padre y si es más apto es sustituido.

**Población inicial**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Individuo 1   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 1 | 2 | 4 | 3 | 5 |   Costo: 23  Individuo 3   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 1 | 2 | 4 | 3 | 5 |   Costo: 22  Valor aleatorio: Individuo 4  Nuevo hijo   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 1 | 2 | 4 | 3 | 5 |   Costo: 20 | Individuo 2   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 1 | 2 | 4 | 3 | 5 |   Costo: 17  Individuo 4   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 1 | 2 | 4 | 3 | 5 |   Costo: 21    **Resultado**: Como el costo del nuevo hijo es menor se inserta en la población |

**Población actual**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Individuo 1   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 1 | 2 | 4 | 3 | 5 |   Costo: 23  Individuo 3   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 1 | 2 | 4 | 3 | 5 |   Costo: 22 | Individuo 2   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 1 | 2 | 4 | 3 | 5 |   Costo: 17  Nuevo hijo   |  |  |  |  |  | | --- | --- | --- | --- | --- | | 1 | 2 | 4 | 3 | 5 |   Costo: 20 |

Tabla 3. Reinserción de un hijo

Una vez que los hijos han sido reinsertados ha concluido una generación del algoritmo y el individuo más apto o de menor costo entre sus distancias es retornado como solución más optima de su generación. Cada vez que se ejecuta una generación es comparada contra la anterior prevaleciendo siempre el individuo más óptimo.

Seudocódigo:

**Algoritmo genético ()**:

Asignar e inicializar parámetros

***Mientras*** *(La condición de paro no se satisfaga) hacer*

*Aplicar elitismo opcional*

*Aplicar Selección*

*Aplicar Cruce (heurística: opcional)*

*Aplicar Mutación (heurística: opcional)*

*Aplicar Reinserción*

***End***

**2.2. Ant Colony Optimization (ACO)**

El algoritmo ACO fue creado por Marco Dorigo en 1992, y está inspirado por una experiencia real en la cual una colonia de hormigas tiene acceso a una fuente de alimento en una zona conectada por un puente dividido en dos caminos de diferente longitud a su nido. Ellas encuentran la ruta más corta entre estos puntos usando feromonas [8]

Las feromonas son una sustancia química que las hormigas depositan al ir del nido a la fuente de comida y viceversa. Cuando se encuentran en un punto de decisión (intersección), toman una opción probabilística sesgada por la cantidad de feromona que detectan.

Las hormigas que elijan el camino más corto llegarán más rápido a la fuente de comida y de manera subsecuente volverán primero a la colonia, a diferencia de aquellas hormigas que tomen rutas de mayor distancia. Mientras este proceso itera, se deposita con mayor frecuencia feromona en el camino más corto, mientras que en los otros esta sustancia comienza a evaporarse conforme transcurre el tiempo (así se evita la convergencia en un óptimo local).

Esto hace que el camino más corto sea seleccionado cada vez más, hasta que al último todas las hormigas terminan usándolo. Este algoritmo fue concebido para encontrar la ruta más corta en el TSP, y las fórmulas usadas para implementar esta técnica se muestran en la tabla 4 [9].

|  |  |
| --- | --- |
| Probabilidad de elección de camino | Pxy = |
| Feromona | xy = (1-*𝜌*) \* xy + |
| Variación de feromona | si se usa xy ,  0 si no se usa |
| Parámetro de aprendizaje | Q = 1 |
| Tasa de evaporación | *𝜌* = 0.01 |
| Costo de camino por hormiga | LK |

Tabla 4. Fórmulas de algoritmo ACO

Basándose en la metodología propuesta en [10], para tener un estudio correcto de la influencia del número de hormigas frente al número de iteraciones, fijamos el producto del número de hormigas y las iteraciones en 60. Por lo tanto, todas las ejecuciones tomarán el mismo tiempo. Variamos el número de hormigas para que tengan los siguientes valores:

{1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, 60}

y el número de iteraciones para que sean respectivamente {60, 30, 20, 15, 12, 10, 6, 5, 4, 3, 2, 1}

Se ejecutó el algoritmo ACO 30 veces con cada una de las combinaciones Núm.-Hormigas \* Núm.-iteraciones = 60, y los resultados se muestran en la Tabla 5.



Tabla 5. Resultados de ejecución donde la diagonal (Hormigas \* Iteraciones) = 60

Al encontrar que los valores más bajos de rutas fueron encontrados por (1 hormiga \* 60 iteraciones) y (3 hormigas \* 20 iteraciones), se exploraron las combinaciones superiores e inferiores a estas celdas, dando el valor 2094 como mínimo local de ACO.

El camino más corto de acuerdo a TSPLIB es el siguiente y fue el mismo:

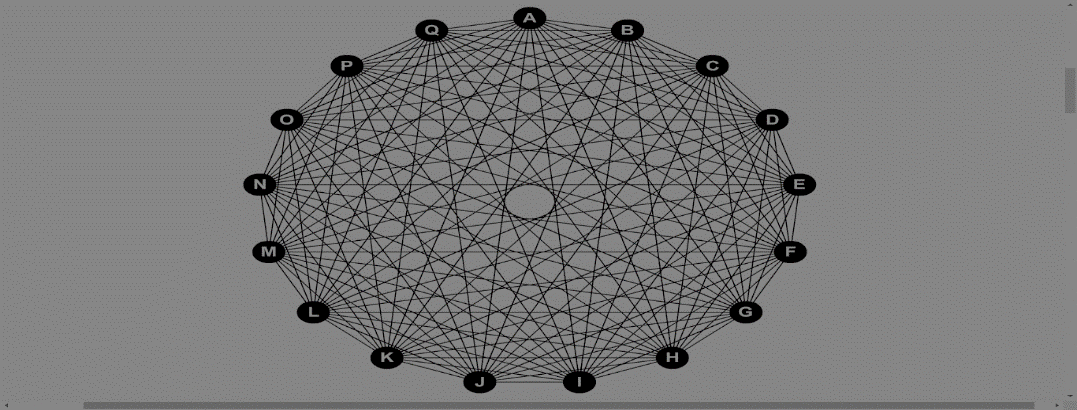


Figura 2. Camino mínimo en TSPLIB y en nuestro algoritmo genético simple con heurísticas creado con [11]

(1, 4, 13, 7, 8, 6, 17, 14, 15, 3, 11, 10, 2, 5, 9, 12, 16) con un costo de **2085** ó

(A, D, M, G, H, F, Q, N, O, C, K, J, B, E, I, L, P) si seguimos la equivalencia de la tabla 6.

Las ilustraciones 2 y 3 fueron generadas por medio de [6].

|  |  |
| --- | --- |
| A | 1 |
| B | 2 |
| C | 3 |
| D | 4 |
| E | 5 |
| F | 6 |
| G | 7 |
| H | 8 |
| I | 9 |
| J | 10 |
| K | 11 |
| L | 12 |
| M | 13 |
| N | 14 |
| O | 15 |
| P | 16 |
| Q | 17 |

Tabla 6. Equivalencia en la etiqueta de nodos entre números y letras

El camino más corto encontrado por el algoritmo ACO fue:

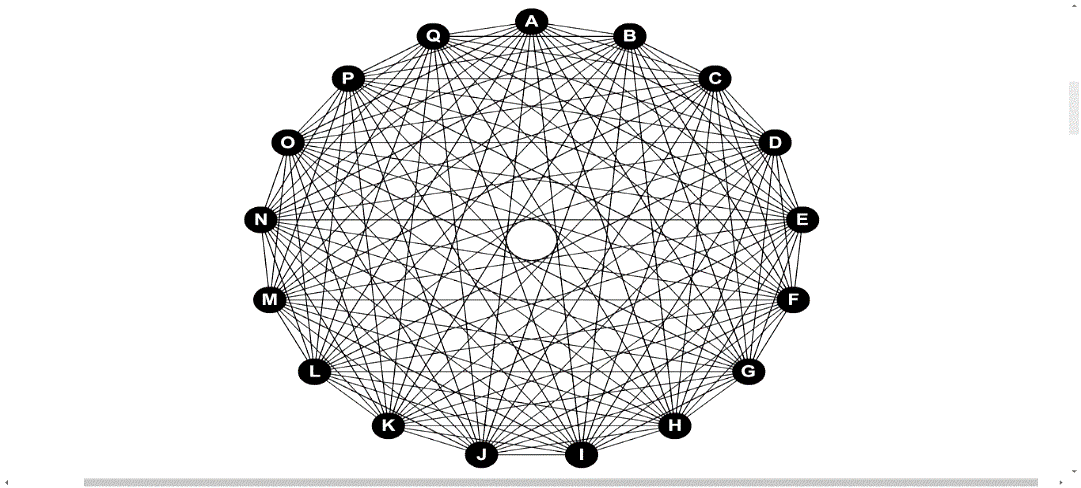


Figura 3. Camino mínimo en ACO creado con [11]

(8, 7, 4, 13, 1, 16, 12, 9, 5, 2, 10, 11 3 ,15, 14 ,17 ,6) con un costo de **2094** ó

(H, G, D, M, A, P, L, I, E, B, J, K, C, O, N, Q, F)

Como intento de mejora, se obtuvo el árbol de expansión mínima del grafo generado por el dataset GR17 y se aumentaron las feromonas a los nodos por los que pasa el árbol, después se pasó la nueva matriz de feromonas (Figura 5) como parámetro al algoritmo ACO en lugar de iniciar en ceros, sin embargo esta modificación no representó una mejora significativa al desempeñarse de forma muy similar al algoritmo ACO original.

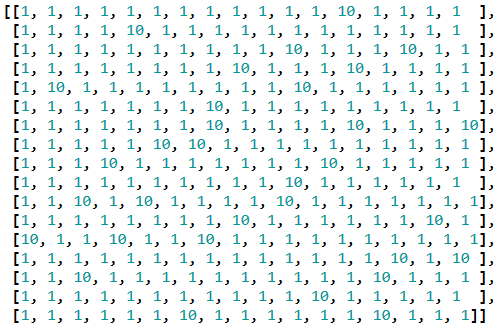


Figura 5: Existen 11 coincidencias entre el árbol de expansión mínima y la ruta más corta

**Análisis estadístico**

Los cuatro algoritmos propuestos fueron comparados por medio de un análisis de Friedman con 30 repeticiones cada uno (tabla 6), en la Figura 6 se puede apreciar que el método colocado en el extremo derecho (Gen-Heurístico) es considerado el mejor.

Tabla 6. Datos para análisis Friedman

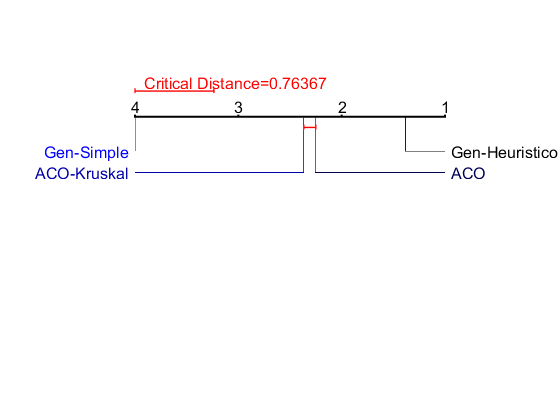


Figura 6. Resultado de análisis Friedman

1. CONCLUSIONES

Como resultado del presente trabajo podemos concluir que resolvimos un problema NP-Hard con tres versiones de algoritmos bioinspirados, un algoritmo genético simple (Gen-simple), un algoritmo genético con heurística ambos propuestos en esta investigación. Personalizamos además el algoritmo ACO-Kruskal para resolver el presente problema. El algoritmo Gen-Heurístico resultó ser el de mejores resultados, obteniendo un costo mínimo de 2085 comparable con la mejor de las soluciones según la literatura.

Los problemas de optimización combinatoria aparecen por la necesidad que muestran ciertos sectores empresariales, como los relacionados con procesos de fabricación, logística, problemas de localización, planificación de tareas, etc. Su fin principal es la optimización de uno o varios objetivos, bajo grandes números de posibles soluciones. Este efecto hace que sea necesaria la implementación de algoritmos de resolución, capaces de dar soluciones en tiempos computacionales razonables. El presente trabajo es un caso de éxito para resolver el problema del comerciante viajero y se puede aplicar a los problemas que estén relacionados con minimizar costos de viajes. Aunque no siempre dará la solución más optima, promete dar una solución cerca del óptimo.

El código fuente de los métodos propuestos en este trabajo se encuentran en la siguiente dirección

https://github.com/ojacinto/genetic\_algorithm

1. REFERENCIAS BIBLIOGRÁFICAS

[1] M. Victoria and G. Travieso, “Problema del viajante de comercio (TSP) Métodos exactos de resolución Traveling Salesman Problem (TSP) Exact solving methods,” 2014.

[2] D. from Techopedia., “What is Traveling Salesman Problem (TSP)?,” *Definition from Techopedia*, 2018. [Online]. Available: https://www.techopedia.com/definition/22635/traveling-salesman-problem-tsp. [Accessed: 29-Nov-2018].

[3] F. S. Caparrini, “Algoritmos de hormigas y el problema del viajante,” 2018. [Online]. Available: http://www.cs.us.es/~fsancho/?e=71.

[4] trevlovett, “Python Ant Colony TSP Solver,” 2018. [Online]. Available: https://github.com/trevlovett/Python-Ant-Colony-TSP-Solver/blob/master/ant.py.

[5] “TSP - Data for the Traveling Salesperson Problem.” [Online]. Available: https://people.sc.fsu.edu/~jburkardt/datasets/tsp/tsp.html. [Accessed: 29-Nov-2018].

[6] J.-Y. Gendreau, Michel; Potvin, *Handbook of Metaheuristics*. 2010.

[7] J. H. Holland, *Adaptation in Natural and Artificial Systems*. 1975.

[8] G. A. Di Caro, “The Ant Colony Optimization Meta-Heuristic” no. February 2014, 1999.

[9] “Colonia Hormigas 1 - YouTube.,” *Youtube*. [Online]. Available: https://www.youtube.com/watch?v=7adfoRX1JFU&list=PLASi3Cszr4qeYi9Y\_X6m0wWqpd4CmrRk7. [Accessed: 29-Nov-2018].

[10] \s Fidanova and P. Marinov, “Number of Ants Versus Number of Iterations on Ant Colony Optimization Algorithm for Wireless Sensor Layout.”

[11] “Graph Creator.” [Online]. Available: https://www.nctm.org/Classroom-Resources/Illuminations/Interactives/Graph-Creator/. [Accessed: 29-Nov-2018].