Implementación de un Algoritmo Genético modificado para la solución al Problema del Agente Viajero

Marcos Alberto Moroyoqui Olan, Ulises Orozco-Rosas, Kenia Picos*

CETYS Universidad, Av. CETYS Universidad No. 4, El Lago, C.P. 22210, Tijuana B.C., México
marcos.moroyoqui@cetys.edu.mx, ulises.orozco@cetys.mx, kenia.picos@cetys.mx

Resumen

El problema del agente viajero ha representado un reto a lo largo del tiempo, debido a sus características se considera como un problema computacionalmente complejo. Debido a lo anterior, varios algoritmos estocásticos han sido desarrollados para proponer una solución factible a este problema. Entre estas soluciones disponibles, los algoritmos evolutivos siempre han sido una buena aproximación para encontrar un buen resultado para este problema, el óptimo en el mejor de los casos. En el presente trabajo se explora una alternativa de solución al problema planteado, un algoritmo genético modificado. Se presenta la implementación de un algoritmo genético modificado para la solución al problema del agente viajero mediante la modificación de la función de cruce. Se realiza una comparativa con la implementación del algoritmo de optimización por colonia de hormigas, con la finalidad de explorar las fortalezas de cada implementación, así como la naturaleza de sus soluciones. La comparativa se presenta en términos de tiempo de ejecución y distancia entregada en la solución proporcionada por cada implementación.

Palabras clave— Algoritmo Genético, Problema del Agente Viajero, Optimización por Colonia de Hormigas

Abstract

The traveling salesman problem has represented a challenge over time, due to its characteristics it is considered a computationally complex problem. Due to the above, several stochastic algorithms have been developed to propose a feasible solution to this problem. Among these available solutions, evolutionary algorithms have always been a good approach to find a good result for this problem, the optimal in the best case. In the present work, an alternative solution to the proposed problem is explored, a modified genetic algorithm. The implementation of a modified genetic algorithm for the solution of the traveling salesman problem by modifying the crossover function is presented. A comparison is made with the implementation of the ant colony optimization algorithm, to explore the strengths of each implementation, as well as the nature of their solutions. The comparison is presented in terms of execution time and distance obtained in the solution provided by each implementation.

Keywords—. *Genetic Algorithm, Travelling Salesmen Problem, Ant Colony Optimization*

1. INTRODUCCIÓN

Actualmente, debido al incremento de la capacidad computacional, se ha logrado resolver problemas cada vez más complejos y difíciles de calcular, lo que ha facilitado el desarrollo de soluciones determinísticas las cuales permiten llevar a cabo tareas tales como la asignación de parámetros, determinación de rutas, entre otros. Sin embargo, a pesar de la creciente evolución de los procesadores, es cierto que existen problemas que no tienen soluciones determinísticas, entre ellas podemos encontrar problemas como el del agente viajero.

El problema del agente viajero se vuelve complicado ya que hoy en día no se ha encontrado ningún algoritmo determinista que permita resolver el problema de manera óptima. Esto es debido a que el problema crece de manera drástica según aumenta el número de ciudades, si pensamos en el recorrido como una permutación de distintas ciudades, podemos observar que existirán n factorial posibles rutas que resuelvan el problema.

Explorar todas estas rutas sería complejo, por eso es que en la actualidad, se han explorado otras opciones las cuales permiten obtener mejores resultados de manera más rápida al incluir procesos que introducen aleatoriedad. A estos algoritmos se les conoce como algoritmos estocásticos los cuales, al incluir una parte de aleatoriedad sacrifican una parte del entendimiento y replicabilidad de los resultados, sin embargo, encontramos mejores acercamientos de manera más rápida.

Entre algunas de las ramas de aplicación de los algoritmos estocásticos, se encuentra el área de los algoritmos evolutivos. De acuerdo con Marrero [1], estos algoritmos consisten en el proceso evolutivo de una población dada, durante un periodo de tiempo prefijado, en los cuales se modelan fenómenos naturales tales como la herencia genética y la competencia Darwiniana por la supervivencia dentro de un hábitat determinado. La idea detrás de los algoritmos evolutivos es imitar lo que la naturaleza hace, incluyendo los cambios que añade mediante las mutaciones y otros procesos desarrollados por la naturaleza [2, 3].

* Autor para la correspondencia (*Corresponding author*).

Dentro de esta corriente de algoritmos evolutivos, podemos encontrar enfoques como lo son los algoritmos genéticos (GA, por sus siglas en inglés) y los algoritmos de optimización por colonias de hormigas (ACO, por sus siglas en inglés), el GA basa su funcionamiento en la evolución de genes y poblaciones imitando el proceso natural de evolución de los seres vivos [4]. Por otro lado, el ACO busca imitar a la naturaleza de las hormigas al replicar como estas mediante sus feromonas trazan caminos y acaban encontrando las mejores rutas hacia su destino.

2. FUNDAMENTOS

A continuación, se describen de manera breve algunos de los fundamentos importantes para resolver problemas complejos como lo es el problema del agente viajero.

2.1 EL PROBLEMA DEL AGENTE VIAJERO

El problema del agente viajero o TSP por sus siglas en inglés (*Travelling Salesmen Problem*) es uno de los problemas más conocidos y complejos de las ciencias computacionales y ha sido abordado por varias ramas de la ingeniería y por distintas razones, su principal aplicación es la de formar una ruta a través de un conjunto finito de nodos, esto se logra mediante un proceso que genera una secuencia específica o una distribución de carácter logístico, en dichos procesos intervienen elementos del transporte los cuales buscan la mejor ruta posible con criterios en distancia o en costo. Proveer soluciones contribuye a mejorar tareas y procesos en distintos ámbitos [5].

El problema del agente viajero se define como, dado un valor entero $n > 0$ y las distancias entre cada par de las n ciudades, donde, estas distancias se dan por medio de la matriz (d_{ij}) de dimensión $n \times n$, y un valor de d_{ij} es un entero mayor o igual a cero. Entonces, un recorrido es una trayectoria que visita todas las ciudades exactamente una vez [5].

2.2 ALGORITMOS GENÉTICOS

De acuerdo con Santos [6], el algoritmo genético (GA) es una técnica de optimización y búsqueda basada en los principios de genética y selección natural. Los GA trabajan sobre una población de individuos, cada uno de ellos representa una posible solución al problema que se desea resolver. Todo individuo tiene asociado una aptitud de acuerdo con la medida en que representa la solución potencial al problema a solucionar [7].

En el proceso de evolución una generación (de individuos) se obtiene a partir de la anterior por medio de operadores. En la reproducción existen dos tipos: cruce y copia, el primero consiste en el emparejamiento entre dos individuos denominados padres y el segundo es el traspaso total del material genético de un individuo hacia la próxima generación [8]. Otro operador muy importante en el proceso de evolución es la mutación, que suele combinarse con el operador de cruce.

2.3 OPTIMIZACION POR COLONIA DE HORMIGAS

Por otro lado, la optimización por colonia de hormigas (*Ant Colony Optimization*, ACO), fue introducida por Marco Dorio en los inicios de 1990 como herramienta para la solución de problemas complejos de optimización. La fuente de inspiración del ACO es el comportamiento natural de las hormigas. Estos insectos cuando están en búsqueda de la comida inicialmente exploran el área alrededor de su nido de forma aleatoria. Durante el regreso al nido, las hormigas depositan una sustancia química llamada feromona sobre el camino, la cual servirá de guía futura para que las demás encuentren los alimentos.

Diferentes estudios han demostrado que la comunicación de las hormigas a través de caminos con feromonas les permite encontrar las rutas más cortas entre su nido y las fuentes de alimento. Esta característica es ampliamente utilizada para la solución de problemas de optimización que necesitan mejorar sustancialmente los tiempos de cómputo para la solución de una aplicación específica.

3. METODOLOGÍA

Para el desarrollo de la comparativa del GA modificado con el ACO se realizó la implementación de estos dos algoritmos programados en C++. Para realizar la implementación se tomó como base los algoritmos descritos por Haupt R. & Haupt S. [9], de los cuales se emplean los parámetros iniciales para el funcionamiento.

Los parámetros iniciales del ACO se componen de una tasa de decrecimiento del 0.5 para el grado de evaporación de las feromonas, una inicialización igualitaria de 0.1 entre las feromonas inicializadas entre ciudades y finalmente la visibilidad la cual es la inversa de la distancia entre ciudades. Para conocer el funcionamiento de este algoritmo, podemos observar su implementación en el pseudocódigo descrito por el Código 1.

Código 1. Pseudocódigo para la implementación del ACO

```
distances = [[]]
for ic in cities:
    for id in cities:
        distances[ic][id] = calcDist(ic,id)

visibility = inverseMatrix(distances)
phmones = [[0.1 for _ in cities] for _ in cities]
tours = []
for  ic in ants:
    tours[ic] = randomPermut(cities)
    tours[ic][lastindex] = tours[ic][firstIndex]

for iter in numberofIters:
    for ic in numberOfAnts:
        for each city in tours:
            st = currentCity
            nxt = nextCitiesToVisit
            probs= calculateProbs(st,nxt,visibility,phmones)
               rcity=random
               for Iz in length(probs):
                   if(rcity)<sum(probs[1:iz])
                        tour[ic]=selectNextCitie(tour[ic],iz)
updatePheromones()
bonusPheromonesBestTour()
updateTrailPheromones()
```

Por otra parte, para el GA modificado, los parámetros empleados en la inicialización son una tasa de mutación del 0.2 (20%), así como una tasa de selección del 0.5 (50%), por lo cual, el algoritmo mantendría al 50% de los mejores individuos generados en una generación. Para la ejecución del algoritmo, se sigue el pseudocódigo mostrado en el Código 2.

En este trabajo se propone un operador de cruce modificado, este operador de cruce se basa en algunas recomendaciones hechas por Hussain et al. [10], el operador de cruce modificado describe en el Código 3.

Código 2. Pseudocódigo para la implementación del GA

```
for  ic in popsize:
    tours[ic] = randomPermut(cities)
    tours[ic][lastindex] = tours[ic][firstIndex]

keep=floor(selection*popsize)
nmut=ceil((popsize-1)*Nt*mutrate)
M=ceil((popsize-keep)/2);

Cost = []
for i in tours:
    Cost[i] = evaltour(i)

Cost,ind = sort(Cost)
Tour = sortWithOrder(Ind)

for I in iters:
    M = ceil((popsize-keep)/2)
    Ma=selectMates(1,M)
    Pa=selectMates(1,M)
    ix = intercalatedJoin(pa,ma)
    for id  step 2 until length(ix)
        tour[keep + ix] = crossOver(tour[ix],tour[ix]+1)
        tour[keep + ix +1] = crossOver(tour[ix]+1,tour[ix])
   mutations = random (1,nmut)
   for I in mutations:
      mutateTour(tour[mutations[I]])
   for i in tours:
      Cost[i] = evaltour(i)
      Cost,ind = sort(Cost)
       Tour = sortWithOrder(Ind)
```

Código 3. Pseudocódigo para la implementación del operador modificado de cruce en el GA

```
Def Crossover(tour1,tour2):
   min, max = Random(2,1:length(tour))
   Cross[min:max] = tour[min:max]
   actual = max+1
   for i in tour2:
        if(i in Cross):
            continue
        else:
              Cross[i] = value;
               if (actual == Cross.size() - 1)
                          actual = 1
                    else if (actual==min-1)
                          break
                   else
                          actual++
if (min != 0)
Cross[0] = Cross[Cross.size() - 1]
else
Cross[Cross.size() - 1] = Cross[0]
```

Por otro lado, en el GA modificado para el operador de mutación, se aplicaron dos mutaciones distintas para mantener cierto grado de variabilidad entre los distintos individuos, entre estos operadores se incluyen dos formas distintas de mutación, el operador espejo y el operador de intercambio, como se observa en el Código 4.

Código 4. Pseudocódigo para la implementación del operador de mutación en el GA

```
Def Mutate(Tour):
    Start,end = random (2,1,length(tour))
    Tour.permute(start,end)
    Start,end = random (2,1,length(tour))
    Section = reversed(tour[start:end])
     Tour[start:end]= section
```

Las distintas ejecuciones fueron realizadas utilizando un procesador Intel Core i7-3770 junto a 8GB de memoria RAM, utilizando Visual Studio 2019, configurando el perfil del proyecto y el compilador para ejecutarse en 64 bits bajo la optimización del compilador O2, la cual le permite al programa alcanzar su mayor pico de rendimiento.

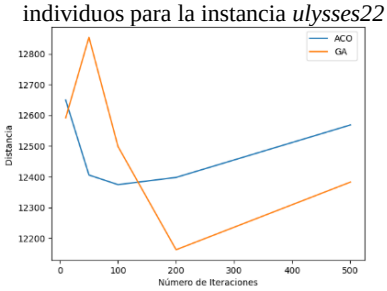
4. RESULTADOS

Para llevar a cabo la comparación entre la eficiencia de los distintos algoritmos se ha decidido aplicar ambas implementaciones, a dos diferentes instancias del problema del agente viajero, propuestas en la librería TSPLIB [11].

Para ello se seleccionarlos las instancias de prueba conocidas como *ulysses22* y *ch150*. Para cada una de las instancias del problema, se probaron tanto la propuesta del GA modificado como el ACO en las distintas configuraciones. En especial se buscó observar cómo varía el resultado de las implementaciones en función del número de iteraciones utilizadas. De la misma forma, se buscó observar el efecto del tamaño de la población en los resultados.

Para esto, se hicieron distintas configuraciones, las cuales consisten en variar el número de individuos, en 10, 50, 100, 200 y 500 individuos, así como el número de iteraciones para cada uno de los experimentos. De la misma forma, para obtener estadísticas el proceso fue repetido 10 veces.

Figura 1. Comportamiento de la optimización de distancia en función del número de iteraciones con una población de 500 individuos para la instancia *ulysses22*



Para la instancia *ulysses22*, la cual consta de 22 ciudades los resultados se observan en la Figura 1. En este

primer acercamiento, donde se compara como el tamaño de la población afecta el número de iteraciones necesarias para alcanzar el mínimo, se puede observar, que el GA rápidamente converge hacia una distancia mínima. Otra diferencia resaltable puede ser observada en el tiempo necesario para alcanzar los resultados, como se aprecia en los resultados que muestra la Tabla 1.

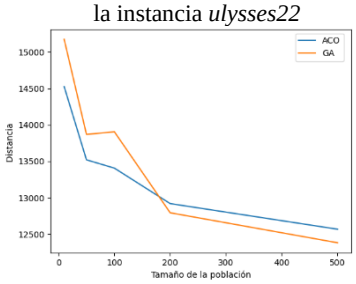
Como se observa en la Tabla 1, el GA obtuvo mejores resultados en un tiempo menor. Por otra parte, el ACO completó su ejecución en un total de 22.44 segundos para determinar la mejor distancia, mientras que el GA, lo realiza en 0.725 segundos. Esto significa que el GA, es aproximadamente 30 veces más rápido que el ACO.

Tabla 1. Tiempo de ejecución (en milisegundos) para el número de iteraciones dado en la instancia de prueba *ulysses22*

Número de iteraciones	ACO	GA
10	424.3	15.9
50	2288.5	78.9
100	4570.3	157.4
200	8708.2	283.6
500	22440	724.9

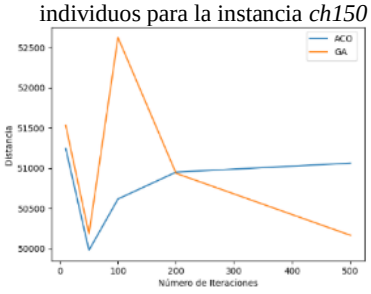
Al analizar, estos resultados en la misma instancia de prueba, pero ahora, comparando el tamaño de la población, en contra de la distancia obtenida, podemos observar una tendencia similar, en la cual el GA vuelve a tener mejores resultados que los obtenidos por el ACO, ver Figura 2. Para esta instancia, TSPLIB registra el mínimo en una distancia de 7,013. El mejor resultado obtenido del GA es 11,966 y de 12,803 para el ACO. En proporción con el resultado registrado en TSPLIB, se obtiene una diferencia de 1.71 para el GA y 1.83 para el ACO, obteniendo un mejor resultado por el GA.

Figura 2. Comportamiento de la optimización de distancia en función del tamaño de la población utilizando 500 iteraciones para la instancia *ulysses22*



Ahora, al realizar el mismo análisis sobre una instancia la cual contiene un número mayor de ciudades, los resultados comenzaron a ofrecer resultados ligeramente distintos, como lo podremos ver en la Figura 3 y 5, donde los resultados obtenidos después de aplicar el GA y el ACO a la instancia *ch150* reducen sus diferencias al aumentar el número de individuos involucrados en las ejecuciones.

Figura 3. Comportamiento de la optimización de distancia en función del número de iteraciones con una población de 10 individuos para la instancia *ch150*



Como se puede observar en la Figura 4 ambos algoritmos muestran una tendencia claramente a la baja cuando el análisis se realiza variando el número de individuos involucrados. Por otro lado, también es posible observar en la Figura 5 que, al aumentar el número de ciudades, los resultados obtenidos por el ACO comenzaron a mejorar drásticamente, llegando al punto en el cual el ACO, comenzó a generar resultados ligeramente mejores que el GA.

Figura 4. Comportamiento de la optimización de distancia en función del tamaño de la población utilizando 10 iteraciones para la instancia *ch150*

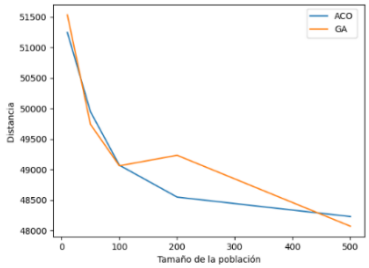


Figura 5. Comportamiento de la optimización de distancia en función del tamaño del número de iteraciones utilizando 500 individuos para la instancia *ch150*

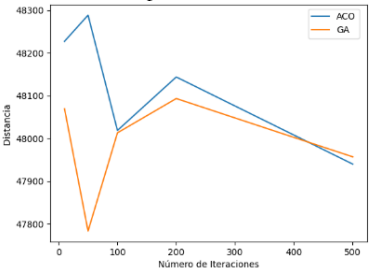


Figura 6. Comportamiento de la optimización de distancia en función del tamaño de la población utilizando 500 iteraciones para la instancia *ch150*

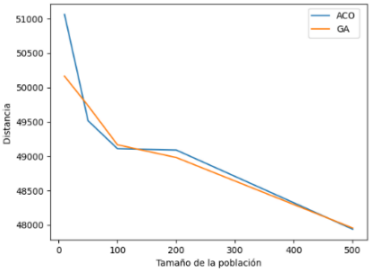


Tabla 2. Tiempo de ejecución (en milisegundos) para el número de iteraciones dado en la instancia de prueba *ch150*

Número de iteraciones	ACO	GA
10	13232.0	85.9
50	66516.6	413.3
100	133174.7	832.1
200	271490.4	1697.7
500	687148.6	4192.6

En la Figura 6, se observa que, desde la perspectiva del tamaño de la población, los resultados se mantienen similares. Para esta instancia, TSPLIB registra el mínimo en una distancia de 6,528. El mejor resultado obtenido del GA es 46,937 y de 47,167 para el ACO. En proporción con el resultado registrado en TSPLIB, se obtiene una diferencia de 7.19 para el GA y 7.22 para el ACO. Donde ambos algoritmos obtienen resultados muy cercanos entre sí, a pesar de esto, la tendencia en el tiempo empleado para converger sigue siendo superior en el GA, ver Tabla 2. De acuerdo con los resultados de la Tabla 2, podemos observar que el GA es 163 veces más rápido que el ACO. Al pasar a observar las rutas generadas por estos algoritmos, podemos observar algunas peculiaridades en las mismas, en la Figura 7 y 8 podemos observar las distintas rutas generadas por el GA y el ACO respectivamente, para una configuración de 10 iteraciones y 10 individuos.

Figura 7. Recorrido generado por el GA para la instancia *ulysses22*, con una configuración de 10 individuos y 10 iteraciones

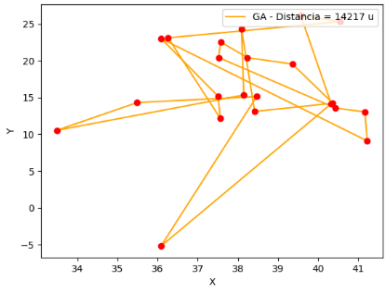
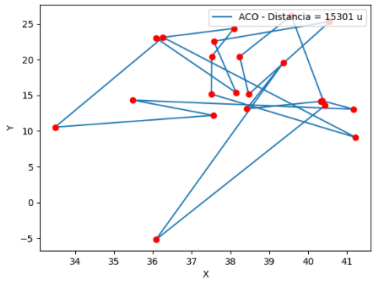


Figura 8. Recorrido generado por el ACO para la instancia *ulysses22*, con una configuración de 10 individuos y 10 iteraciones



Como podemos observar en la Figura 7 y 8, ambos algoritmos generan distintas rutas de las cuales, el GA genera un mejor primer acercamiento a la solución del problema, sin embargo, para observar su evolución a través de las diferentes configuraciones podremos observar su comportamiento cuando en la misma instancia se cuenta con una configuración de 500 individuos y 500 iteraciones (ver Figura 9 y 10).

Figura 9. Recorrido generado por el GA para la instancia *ulysses22*, con una configuración de 500 individuos y 500 iteraciones

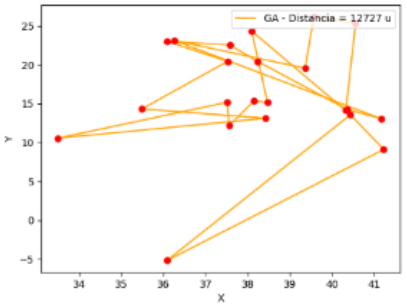
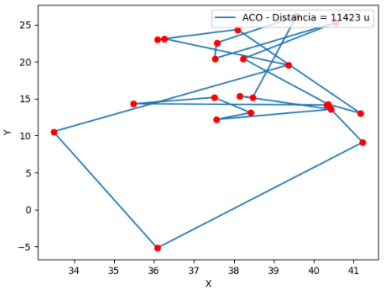


Figura 10. Recorrido generado por el ACO para la instancia *ulysses22*, con una configuración de 500 individuos y 500 iteraciones



Para estas ejecuciones en particular podemos observar como el ACO, genera una mejor aproximación al resultado que la generada por el GA por alrededor de 1,300 unidades de distancia. De la misma manera es factible observar que el GA, mantiene una ejecución similar a la de su configuración de 10 iteraciones y 10 individuos.

Si analizamos este mismo comportamiento en la instancia la cual contiene más ciudades, como lo es la *ch150*, podremos observar resultados distintos, estos resultados se pueden observar en la Figura 11, 12, 13 y 14.

Figura 11. Recorrido generado por el GA para la instancia *ch150*, con una configuración de 10 individuos y 10 iteraciones

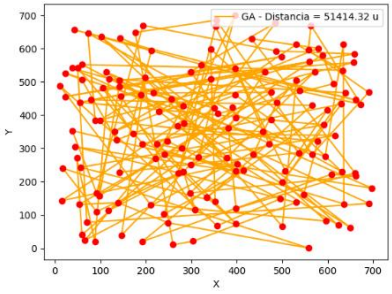


Figura 12. Recorrido generado por el ACO para la instancia *ch150*, con una configuración de 10 individuos y 10 iteraciones

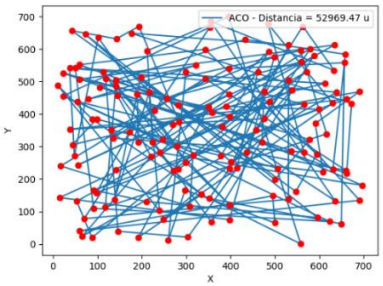


Figura 13. Recorrido generado por el GA para la instancia *ch150*, con una configuración de 500 individuos y 500 iteraciones

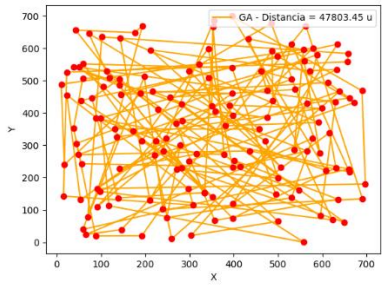
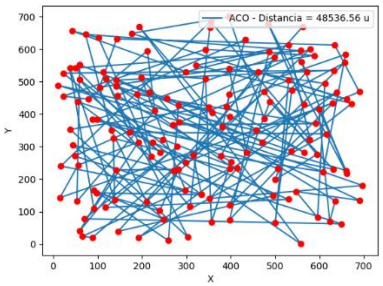


Figura 14. Recorrido generado por el ACO para la instancia *ch150*, con una configuración de 500 individuos y 500 iteraciones



Como se puede observar desde la Figura 11 a la 14, el GA, en ambas configuraciones obtiene mejores resultados que su contraparte ACO, sin embargo, tomando en cuenta el tamaño del recorrido podemos ver que esta diferencia entre las distancias es relativamente baja, haciendo que sus resultados se separen entre si alrededor de 1,000 unidades de distancia. A pesar de esto podemos ver como aún queda un amplio margen de mejora para ambos algoritmos.

5. CONCLUSIONES Y RECOMENDACIONES

Las pruebas realizadas anteriormente dejan observar algunos comportamientos interesantes de los algoritmos comparados. Entre uno de los hallazgos más importante es el efecto del número de iteraciones, comparado contra el efecto del tamaño de la población. En ambos casos, tanto el GA modificado como el ACO, los resultados tienden a mejorar de manera drástica cuando el número de individuos aumenta.

Por otro lado, cuando el número de iteraciones aumenta, aunque existe una tendencia a la baja, la realidad es que para ambos algoritmos el comportamiento depende de la instancia de prueba.

Otro de los aspectos que resaltan, son las diferencias entre los resultados obtenidos entre ambos algoritmos, el GA en la mayoría de los casos obtuvo mejores resultados cuando el número de individuos y ciudades visitadas era menor. Por otro lado, conforme el número de ciudades, individuos e iteraciones ascienden el ACO empieza a tener mejor rendimiento frente al GA hasta producir resultados los cuales difieren ligeramente entre sí.

Una de las razones por las que podríamos afirmar que el GA funciona mejor en las configuraciones anteriormente descritas, es debido a que las mutaciones y el cruce entre individuos que componen al GA le dotan de una mayor capacidad de exploración en las primeras etapas, lo que propician que de un rendimiento mejor al ACO.

Sin embargo, según el número de ciudades aumenta y el número de posibles recorridos aumenta en la misma manera, el espacio muestral que ambos algoritmos tienen que recorrer se amplifica exponencialmente, propiciando que el ACO pueda generar rutas con mejores aproximaciones que el GA, ya que el mismo no basa su funcionamiento completamente en la aleatoriedad y exploración de distintas rutas.

Otro de los aspectos importantes a mencionar son las concesiones que ambos algoritmos hacen. Por un lado, el GA basa su funcionamiento en procesos mayormente estocásticos, lo que le dan una mayor diversidad al número de rutas que se pueden generar, esto le permite no realizar cálculos complejos, lo que se ve reflejado en su tiempo de ejecución, siendo más rápido que el ACO.

Finalmente, podemos observar que ambos algoritmos generan buenos acercamientos para resolver el problema del agente viajero, sin embargo, el GA, nos da un mejor acercamiento inicial al poder explorar una mayor cantidad de opciones en un menor tiempo. De manera general se puede concluir que el GA, es mejor opción debido a que suele producir mejores acercamientos en un periodo más corto de tiempo debido a su capacidad de exploración. Como trabajo futuro, se plantea una combinación entre ambos acercamientos, la cual genere una población inicial mediante el GA y desarrolle sus etapas finales mediante el ACO. Este acercamiento tiene el potencial para generar recorridos con una mayor diversidad y mejores acercamientos a la solución óptima.

REFERENCIAS

[1] A. Marrero Severo, L. M. Pedroso Rodriguez y J. Barrios Ginart, «Algoritmos Evolutivos en la solucion de problemas de estimacion de parametros,» *Revista Matematica : Teoría y Aplicaciones*, vol. 13, nº 2, pp. 139-150, 2006.

[2] O. Montiel, R. Sepúlveda y U. Orozco-Rosas, «Optimal Path Planning Generation for Mobile Robots using Parallel Evolutionary Artificial Potential Field,» *Journal of Intelligent & Robotic Systems*, vol. 79, nº 2, pp. 237-257, 2015.

[3] U. Orozco-Rosas, K. Picos y O. Montiel, «Hybrid path planning algorithm based on membrane pseudo-bacterial potential field for autonomous mobile robots,» *IEEE Access*, vol. 7, nº 1, pp. 156787-156803, 2019.

[4] U. Orozco-Rosas, O. Montiel y R. Sepúlveda, «Parallel Evolutionary Artificial Potential Field for Path Planning—An Implementation on

GPU,» de *Design of Intelligent Systems Based on Fuzzy Logic, Neural Networks and Nature-Inspired Optimization*, Studies in Computational Intelligence, vol 601. Springer, Cham, 2015, pp. 319-332.

[5] E. López, Ó. Salas y Á. Murillo, «El problema del agente viajero: Un algoritmo determinístico utilizando búsqueda tabú,» *Revista de Matemática: Teoría y Aplicaciones*, vol. 21, nº 1, pp. 127-144, 2014.

[6] C. Santos Rodriguez y A. T. Aday Martinez, «Algoritmos Genéticos: una solución para la optimización del Reflector Parabólico,» *Ingeniería Electrónica, Automática y Comunicaciones*, vol. 35, nº 1, pp. 1-15, 2014.

[7] K. Picos, U. Orozco-Rosas, V. Díaz-Ramírez y O. Montiel, «Pose Estimation in Noncontinuous Video Sequences Using Evolutionary Correlation Filtering,» *Mathematical Problems in Engineering*, vol. 2018, pp. 1-13, 2018.

[8] U. Orozco-Rosas, O. Montiel y R. Sepúlveda, «Mobile robot path planning using membrane evolutionary artificial potential field,» *Applied Soft Computing*, vol. 77, pp. 236-251, 2019.

[9] R. L. Haupt y S. E. Haupt, Practical Genetic Algorithms, Wiley-Interscience, 2004.

[10] M. Yousaf Shad , H. Ijaz , S. M. Nauman, H. Abid, . S. Alaa Mohamd y G. Showkat, «Genetic Algorithm for Traveling Salesman Problem with Modified Cycle Crossover Operator,» *Computational Intelligence and Neuroscience*, 2017.

[11] Heidelberg University, *TSPLIB- Discrete and Combinatorial Optimization*, 2013.