Optimización de parámetros del algoritmo ACO mediante búsqueda aleatoria para encontrar soluciones parciales al problema del agente viajero (TSP) en grafos de hasta 50 nodos

1st Daniel Santiago Silva Capera
Ingeniería de Sistemas y Computación
Universidad Nacional de Colombia
Bogotá D.C, Colombia
dasilvaca@unal.edu.co

2nd Jonathan Steven Ochoa Celis
Ingeniería de Sistemas y Computación
Universidad Nacional de Colombia
Bogotá D.C, Colombia
jsochoac@unal.edu.co

3rd Cesar Esteban Diaz Medina
Ingeniería de Sistemas y Computación
Universidad Nacional de Colombia
Bogotá D.C, Colombia
cediazme@unal.edu.co

4th Victor Manuel Davila Castaneda
Ingeniería mecatronica
Universidad Nacional de Colombia
Bogotá D.C, Colombia
vdavila@unal.edu.co

5th Diego Esteban Quintero Rey
Ingeniería de Sistemas y Computación
Universidad Nacional de Colombia
Bogotá D.C, Colombia
diquintero@unal.edu.co

6th Nicolas David Contreras Ramirez
Ingeniería de Sistemas y Computación
Universidad Nacional de Colombia
Bogotá D.C, Colombia
ndcontrerasr@unal.edu.co

***Abstract*—El artículo describe el uso del algoritmo de colonia de hormigas (Ant Colony Optimization, ACO) para resolver el problema del agente viajero (Traveling Salesman Problem, TSP) en grafos de tamaño variable. El objetivo del estudio es optimizar el rendimiento del algoritmo ACO para encontrar soluciones parciales al problema del TSP en grafos de tamaño variable mediante la definición de rangos óptimos de parámetros. Los objetivos específicos incluyen la implementación del algoritmo ACO en C++, la construcción de un generador de grafos aleatorios, la optimización de los parámetros con búsqueda aleatoria, la definición de rangos óptimos de los parámetros, la construcción de un programa que utilice los rangos de parámetros definidos y la realización de experimentos con el programa implementado. Los resultados muestran mejoras en el desempeño del algoritmo tras la optimización de los parámetros y se presenta una comparación del desempeño del algoritmo con respecto a la cota inferior del algoritmo 1-Tree.**

I. Introducción

El Problema del Agente Viajero (TSP, por sus siglas en inglés) es uno de los problemas más estudiados en la teoría de la optimización combinatoria. Se trata de un problema NP-completo que consiste en encontrar la ruta más corta posible que un vendedor debe tomar para visitar todas las ciudades especificadas y regresar a su punto de partida. Este problema es aplicable en varios campos, como la logística, la planeación, la fabricación de circuitos electrónicos, entre otros [2].

Este proyecto tiene como objetivo optimizar el rendimiento del algoritmo de la colonia de hormigas (Ant Colony Optimization, ACO) para encontrar soluciones parciales al problema del agente viajero en grafos de tamaño variable, mediante la definición de rangos óptimos de parámetros. El proyecto busca implementar el algoritmo ACO en C++, construir un generador de grafos aleatorios, optimizar los parámetros con búsqueda aleatoria, definir rangos óptimos de los parámetros y construir un programa en C++ que utilice los rangos de parámetros definidos según el tamaño del grafo. El desempeño del algoritmo será evaluado mediante la comparación de las distribuciones de la métrica 1-Tree antes y después de la optimización de los parámetros. Los resultados del proyecto, incluyendo las matrices de correlación, los histogramas, los diagramas de dispersión y de distribución, pueden encontrarse en el repositorio del proyecto disponible en GitHub en [4].

II. Descripción del problema

El Problema del Agente Viajero (Traveling Salesman Problem o TSP, por sus siglas en inglés) consiste en un vendedor que necesita visitar un conjunto de n ciudades, cuyas posiciones y distancias entre ellas son conocidas. El objetivo es encontrar la ruta más corta posible que permita al vendedor visitar cada ciudad exactamente una vez y regresar al punto de partida.

Este problema tiene aplicaciones en diversos campos, como la logística, la planificación y la fabricación de circuitos electrónicos, entre otros. Aunque existen $n!$ posibles rutas, se pueden reducir significativamente al darse cuenta de que cualquier ruta puede ser considerada como la misma si se parte de cualquier ciudad y se recorre en cualquier dirección. Por lo tanto, el número de rutas posibles se reduce a $(n-1)!/2$ [1].

Una solución para el TSP es el algoritmo de la colonia de hormigas (Ant Colony Optimization). En la siguiente sección, se describirá en detalle cómo este algoritmo genera el camino óptimo para un número dado de nodos y se mostrará una

gráfica que representa este camino y otros posibles caminos [1].

III. Objetivos

Optimizar el rendimiento del algoritmo ACO para encontrar soluciones parciales al problema del agente viajero (TSP) en grafos de tamaño variable mediante la definición de rangos óptimos de parámetros.

A. Objetivos específicos

- Implementar el algoritmo ACO en C++.
- Construir un generador de grafos aleatorios de tamaño n en el cada nodo esté definido por sus coordenadas cartesianas.
- Optimizar los parámetros con búsqueda aleatoria para los tamaños de grafo $n = 10, 20, 30, 40, 50$.
- Definir rangos óptimos de los parámetros a partir del análisis de matrices de correlación, gráficos de dispersión y de distribución.
- Construir un programa en C++ que utilice los rangos de parámetros definidos según el tamaño del grafo.
- Realizar experimentos con el programa implementado y generar gráficas para visualizar la solución y el desempeño del algoritmo.

IV. Metodología

A. Implementación del algoritmo ACO

El algoritmo ACO se implementó en C++, tomando como referencia una implementación existente en Python disponible en [3]. Se clonó este repositorio y se revisó línea por línea, verificando la correctitud y que las fórmulas coincidieran con la bibliografía consultada. Se hicieron unos ajustes a esta versión en Python para que fuera más clara y eficiente. Este código modificado está disponible en [5]. Luego se procedió a traducir el código a C++. El código en C++ se modularizó en tres clases:

- ACO: Clase que incluye métodos para actualizar la matriz de feromonas de la colonia y solucionar el problema del TSP mediante instanciación de hormigas que recorren el grafo en cada iteración, actualizando la matriz de feromonas y llevando registro de la mejor solución encontrada hasta el momento.
- ACO Graph: Clase que representa el grafo del problema mediante una matriz de distancias y una matriz de feromonas.
- Ant: Clase que representa la unidad básica del algoritmo ACO, implementando los métodos de selección del siguiente nodo y variabilidad de la intensidad de feromonas dispersadas en cada paso.

El algoritmo de ACO puede representarse en psuedocódigo de la siguiente manera [2]:

```
1:  procedure ACO(iterations, ants)
2:      best-solution ← ∞
3:      for iter in iterations do
4:          initialize-ants
5:          for ant in ants do
6:              ant.calculate-solution
7:              if ant.solution < best-solution then
8:                  best-solution ← ant.solution
9:              end if
10:             ant.put-pheromones
11:         end for
12:         update-pheromone-matrix
13:     end for
14:     return best-solution
15: end procedure
```

La ecuación 1 define la probabilidad para que una hormiga estando en un nodo x, seleccione el nodo y como su siguiente nodo. Donde τ_{xy} es la cantidad de feromona depositada para la transición del estado x al estado y, $0 \leq \alpha$ es un parámetro para controlar la influencia de τ_{xy}, η_{xy} es la deseabilidad de la transición de estado xy (conocimiento previo, típicamente $1/d_{xy}$, donde d es la distancia) y $\beta \geq 1$ es un parámetro para controlar la influencia de η_{xy}. τ_{xz} y η_{xz} representan el nivel de rastro y la atracción para otras posibles transiciones de estado [2].

$$p_{xy}^k = \frac{(\tau_{xy}^\alpha)(\eta_{xy}^\beta)}{\sum_{z \in \text{allowed}} (\tau_{xz}^\alpha)(\eta_{xz}^\beta)} \tag{1}$$

La ecuación 2 define la cantidad de feromona que se deposita en cada enlace del grafo. Donde τ_{xy} es la cantidad de feromona depositada para una transición de estado xy, ρ es el coeficiente de evaporación de feromona, m es el número de hormigas y $\Delta\tau_{xy}^k$ es la cantidad de feromona depositada por la k-ésima hormiga, típicamente dada para un problema TSP (con movimientos correspondientes a arcos del grafo) por la ecuación 3, donde L_k es el costo del recorrido de la k-ésima hormiga (típicamente longitud) y Q es la constante de intensidad [2].

$$\tau_{xy} \leftarrow (1-\rho)\tau_{xy} + \sum_{k}^{m} \Delta\tau_{xy}^k \tag{2}$$

$$\Delta\tau_{xy}^k = \begin{cases} Q/L_k & \text{if ant } k \text{ uses link } xy \text{ in its tour} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

B. Construcción del generador de grafos

Se creó un programa que recibe el tamaño n del grafo como parámetro y genera las coordenadas cartesianas de los nodos utilizando un generador de números aleatorios con distribución uniforme. Se garantiza que los nodos estén separados por al menos 100 unidades y se calculan las distancias entre ellos para generar una matriz de distancias utilizada en ACO Graph. El programa también devuelve una lista de puntos que se utiliza para graficar el grafo con las librerías NetworkX y Matplotlib de Python al final del proceso.

C. Optimización de parámetros con búsqueda aleatoria

Se creó un programa que emplea el generador de grafos y el algoritmo ACO para optimizar parámetros por medio de búsqueda aleatoria en cinco tamaños de grafo: $n =$

$10, 20, 30, 40, 50$. Los parámetros optimizados fueron: número de hormigas, número de iteraciones, coeficientes Alpha (α), Beta (β), Rho (ρ), y la intensidad (Q) de la feromona dispersada. A continuación, una breve definición para cada uno:

- Número de hormigas: cantidad de agentes que recorren el grafo. Cada hormiga sigue una ruta diferente y en cada iteración se actualizan las feromonas de acuerdo a la calidad de la ruta.
- Iteraciones: número de veces que se ejecuta el algoritmo. Puede verse como generaciones de hormigas. En cada iteración, se instancia un nuevo conjunto de hormigas, pero la matriz de feromonas de la colonia sigue siendo la misma, por lo que con cada iteración, se converge un poco más a la solución.
- Alpha (α): representa la importancia relativa de la feromona al decidir cuál será el próximo nodo. Si es alto, la hormiga le dará más importancia a la cantidad de feromona en esa arista que a la distancia. Favorece la exploración, es decir, buscar nuevas soluciones en regiones inexplorados.
- Beta (β): representa la importancia relativa de la heurística (la distancia en este caso) al decidir cuál será el próximo nodo. Si es alto, la hormiga le dará más importancia a la distancia y que a la cantidad de feromona en esa arista. Favorece la explotación, es decir, refinar soluciones ya existentes para intentar mejorarlas.
- Rho (ρ): tasa de evaporación de las feromonas. Controla la rapidez con la que las feromonas se desvanecen con el tiempo. Un ρ mayor es una evaporación más rápida al final de cada iteración, por lo que la siguiente generación de hormigas, empieza con menos información.
- Intensidad (Q): se utiliza para actualizar la cantidad de feromonas depositadas por las hormigas en las aristas del grafo. Una vez que una hormiga ha completado su recorrido, se utiliza la intensidad para calcular cuánta feromona debe depositar en las aristas que ha recorrido. En este proyecto se utilizó la estrategia cycle-system, que consiste en incrementar el nivel de feromonas al final de cada recorrido por un factor de Q/C_{ant} donde C_{ant} es el costo del recorrido de la hormiga. A menor costo, se incrementa más el nivel de feromonas. Como Q es el mismo para todas las hormigas, no es determinante que valor se use.

Para cada uno de los 100 grafos generados por tamaño, se ejecutó el algoritmo ACO 30 veces para registrar el mejor resultado y los parámetros utilizados. Luego se realizaron 30 ejecuciones adicionales para cada grafo utilizando los parámetros óptimos, y se calculó la media y desviación estándar de los costos obtenidos. Los resultados y parámetros se informan en un dataset de 100 filas para cada tamaño de grafo. Todos los experimentos se realizaron remotamente en una sala de física habilitada para el curso.

D. Definición de rangos óptimos de parámetros

Se llevó a cabo un análisis en Python mediante la visualización de matrices de correlación y gráficos de dispersión y distribución. El proceso consistió en lo siguiente:

- Para cada tamaño de grafo, se calculó una matriz de correlación a partir del conjunto de datos construido con los parámetros y la variable objetivo. En este caso, la variable objetivo es el costo medio obtenido de los experimentos para cada grafo. Además, se incluyó la desviación estándar para examinar la relación de los parámetros con esta métrica que puede indicar si favorecen o no la convergencia de la solución.
- Se estableció un umbral arbitrario de correlación de magnitud mayor a 0.15 para seleccionar los parámetros más importantes en la minimización del costo. A continuación, se utilizó una técnica de agrupación de histogramas (histogram binning) [7] para determinar los intervalos de valores donde el costo promedio es menor. Para ello, se dividió el dataset en 10 subintervalos en función del valor mínimo y máximo del parámetro en cuestión. Luego, se calculó el valor medio de la variable objetivo para cada subintervalo. El intervalo con la media más baja de la variable objetivo fue seleccionado como el rango óptimo.
- Para aquellos parámetros que no presentaban una correlación significativa con el costo pero sí con la desviación estándar, se recurrió a la observación visual de gráficos de dispersión y distribución para determinar qué rangos serían más adecuados.
- Finalmente, para los parámetros restantes, se establecieron los rangos presentados en la Tabla I, donde n es el tamaño del grafo.

Parámetro	Min 1	Max 2
Hormigas	5	$5n$
Iteraciones	$0.5n$	$0.5n + 100$
Alpha (α)	0.5	2.0
Beta(β)	1.0	5.0
Rho (ρ)	0.1	0.9
Intensidad (Q)	1	10

TABLE I
PARÁMETROS POR DEFECTO

E. Desarrollo del programa en C++ que utiliza los rangos definidos para los parámetros

Definidos los parámetros, se construyó se construyó un programa en C++ que contiene una tabla (matriz) con estos valores. El programa requiere dos parámetros de entrada: el tamaño n del grafo y el número de experimentos a realizar. Al recibir estos parámetros, el programa genera un grafo aleatorio de tamaño n utilizando el generador de grafos. A continuación, inicializa una instancia de ACO con los parámetros óptimos correspondientes. Para seleccionar los parámetros, se utiliza la siguiente regla: si $n <= 10$, se utilizan los valores definidos para $n = 10$; si $10 < n <= 20$, se utilizan los valores definidos para $n = 20$, y así sucesivamente hasta $n = 50$. Si $n > 50$, se utilizan los valores definidos para $n = 50$. El programa ejecuta el número de experimentos definido por el usuario, utilizando el mismo grafo y los mismos parámetros.

Se generan tres archivos de texto: uno que contiene información general como el tamaño del grafo, la cota inferior calculada con el algoritmo de 1-Tree, los parámetros utilizados y las coordenadas cartesianas de los nodos; otro archivo que contiene los costos obtenidos en cada experimento; y un tercer archivo que contiene los caminos de las soluciones encontradas, uno por cada experimento. El algoritmo de 1-Tree funciona de la siguiente manera: Se hacen n iteraciones. En cada iteración se retira un nodo distinto del grafo. Para este grafo con $n-1$ nodos, se obtiene el MST con su costo. Luego, se añade el nodo retirado y se une a sus dos nodos más cercanos. Al costo obtenido se le suma el costo de estos dos nuevos enlaces. En cada iteración se va llevando registro de este costo, y al final se entrega el mayor de todos. Esa es la cota inferior más cercana a la solución real del problema de TSP, y permite comparar el desempeño del algoritmo ACO.

F. Generación de gráficas y análisis del desempeño

Al final de todo el proceso, un script de Python lee los archivos generados, y genera las siguientes gráficas:

- Distribución y dispersión de costos obtenidos comparada con la cota inferior
- Distribución de la tasa de desempeño
- Grafo con el camino de la mejor solución resaltado

V. Resultados

En las Figuras 1 Y 2 se muestran las matrices de correlación obtenidas para $n = 40, 50$. Las matrices de correlación para los demás valores de n, así como las demás gráficas generadas que no se incluyen en este reporte, pueden encontrarse en el repositorio del proyecto disponible en [4] . En estas matrices, solo se prestó atención a las filas y columnas correspondientes a la variable objetivo (costo medio) y a la desviación estándar del costo.

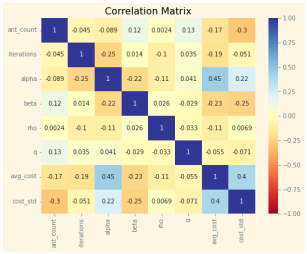

Fig. 1. Matriz de Correlación para $n = 40$

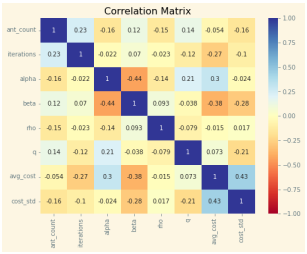

Fig. 2. Matriz de Correlación para $n = 50$

En la Tabla II se presentan los resultados de la correlación de cada parámetro con respecto al costo medio (variable objetivo). Solo se muestran los valores cuya magnitud es mayor o igual a 0.15. Para aquellos parámetros que presentaron una correlación sobre este umbral, se aplicó la técnica de histogram binning [7], y los resultados se resumen en la Tabla III.

En el caso de los parámetros que no mostraron una correlación sobre este umbral con respecto al costo medio, pero sí con la desviación estándar, se realizó un análisis visual mediante la observación de los diagramas de distribución y

Parámetro	$n = 10$	$n = 20$	$n = 30$	$n = 40$	$n = 50$
Hormigas	–	–	–	-0.17	–
Iteraciones	0.18	–	–	-0.19	-0.27
Alpha (α)	–	0.29	0.46	0.45	0.30
Beta (β)	-0.38	-0.17	-0.25	-0.23	-0.38

TABLE II
CORRELACIÓN ENTRE LOS PARÁMETROS Y EL COSTO MEDIO PARA DISTINTOS VALORES DE n

Parámetro	$n = 10$	$n = 20$	$n = 30$	$n = 40$	$n = 50$
Hormigas	–	–	–	$(167.8, 182.9)$	–
Iter.	$(7.0, 16.7)$	–	–	$(89.6, 99.4)$	$(84.4, 94.3)$
Alpha (α)	–	$(0.51, 0.65)$	$(0.50, 0.65)$	$(0.51, 0.65)$	$(0.65, 0.80)$
Beta (β)	$(4.59, 4.99)$	$(3.79, 4.19)$	$(3.48, 3.85)$	$(4.16, 4.54)$	$(4.58, 4.96)$

TABLE III
RANGOS ÓPTIMOS DE PARÁMETROS DEFINIDOS CON HISTOGRAM BINNING

dispersión. Los valores de correlación obtenidos para estos parámetros se resumen en la Tabla IV y los rangos definidos por observación están en la Tabla V.

Parámetro	$n = 10$	$n = 30$	$n = 50$
Hormigas	–	-0.20	-0.16
Rho (ρ)	0.19	-0.21	–
Intensidad (Q)	0.19	–	-0.21

TABLE IV
CORRELACIONES DE PARÁMETROS CON LA DESVIACIÓN ESTÁNDAR DEL COSTO

En las Figuras 3 y 4 hay un ejemplo de los diagramas de dispersión y de distribución que se usaron para definir el rango de hormigas para $n = 30$ por simple observación.

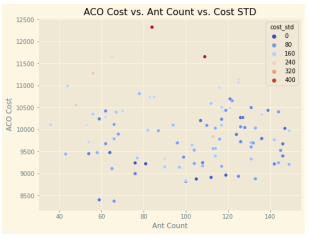

Fig. 3. Diagrama de dispersión del parámetro hormigas para $n = 30$

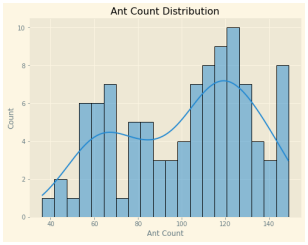

Fig. 4. Diagrama de distribución del parámetro hormigas para $n = 30$

Para los parámetros restantes, se usaron los rangos por defecto presentados en la Tabla I.

El desempeño del algoritmo con respecto a la cota inferior del algoritmo 1-Tree mostró mejoras tras la optimización de los parámetros. En las Figuras 5, 6, 7, y 8 se presentan las distribuciones de esta métrica antes y después de la optimización para los casos $n = 40, 50$. Los datos de antes de la optimización fueron los recolectados durante la búsqueda aleatoria (100 grafos diferentes, cada uno 30 experimentos), mientras que los datos de después de la optimización (1 grafo, 100 experimentos) corresponden a la ejecución de los experimentos con los parámetros ya definidos. Se observa que tras la optimización, no se obtienen tasas mayores a 1.20 y

Parámetro	n=10	n=30	n=50
Hormigas	–	(140, 150)	(150, 225)
Rho	(0.3, 0.8)	(0.5, 0.8)	–
Intensidad	(1, 7)	–	(4, 9)

TABLE V
RANGOS ÓPTIMOS DE PARÁMETROS DEFINIDOS VISUALMENTE

para el caso de 40 nodos, todos los resultados estuvieron por debajo de 1.10.

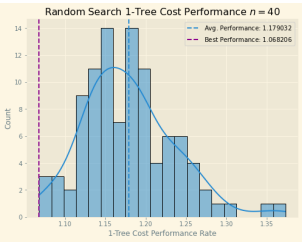

Fig. 5. Distribución de tasas de desempeño para $n = 40$ antes de la optimización

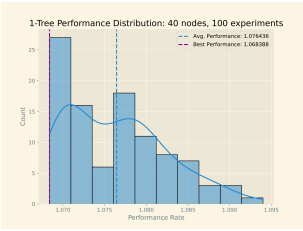

Fig. 6. Distribución de tasas de desempeño para $n = 40$ después de la optimización

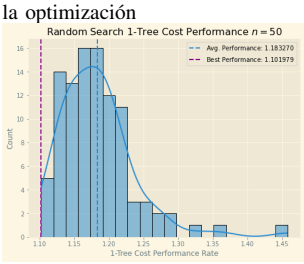

Fig. 7. Distribución de tasas de desempeño para $n = 50$ antes de la optimización

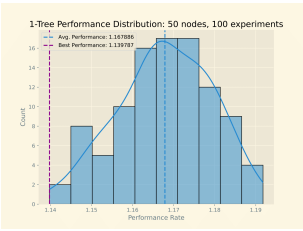

Fig. 8. Distribución de tasas de desempeño para $n = 50$ después de la optimización

La tasa de desempeño se calcula con la fórmula:

$$\text{Tasa de Desempeño} = \frac{\text{costo de ACO}}{\text{cota inferior de 1-Tree}} \tag{4}$$

Un resultado aceptable para nosotros es entre 1.05 y 1.20, teniendo en cuenta que el promedio para otras heurísticas como las voraces es de 1.17 y para el algoritmo de Christofides es de 1.1 y no superior a 1.5 [1] [6].

Las Figuras 9 y 10 muestran las mejores soluciones obtenidas para $n = 40, 50$ en una ejecución de 100 experimentos. Los parámetros utilizados se encuentran en cada figura.

VI. CONCLUSIONES

En general, se encontró que el parámetro de número de hormigas no fue determinante y funcionó bien en el rango de n a 5n para todas las instancias. Solo se observó una pequeña correlación en la instancia de 40 nodos, lo que sugiere que a medida que aumenta el número de hormigas, el costo se reduce.

Por otro lado, el número de iteraciones parece ser más determinante a medida que el grafo incrementa su tamaño, indicando que para instancias mayores, más iteraciones reducirían el costo.

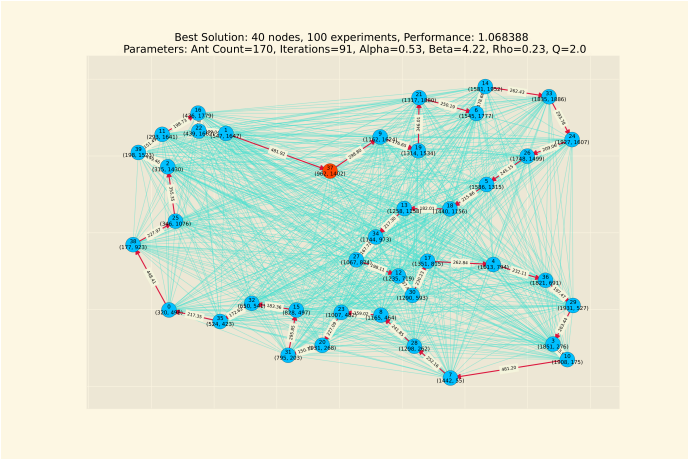

Fig. 9. Mejor solución para una ejecución de 100 experimentos para un grafo de 40 nodos con los parámetros optimizados.

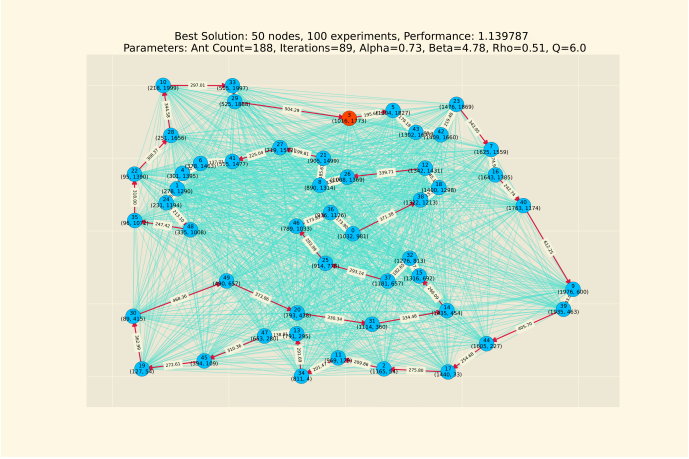

Fig. 10. Mejor solución para una ejecución de 100 experimentos para un grafo de 50 nodos con los parámetros optimizados.

El rango óptimo de α se mantuvo consistente entre 0.5 y 0.65 para instancias de 30, 40 y 50 nodos, lo que sugiere un sesgo por la explotación antes que la exploración. El rango de beta se mantuvo entre 3.5 y 5 para los cinco escenarios, lo que ratifica el sesgo por la explotación.

Es necesario investigar a qué se debe esta configuración entre α y β, y determinar si podrían tomar valores en la misma magnitud. Es probable que el sesgo por la explotación se deba a que el rango de iteraciones utilizado estuvo entre $0.5n$ y $0.5n+100$, lo que podría haber resultado en que al usar valores altos de α, el número de iteraciones no era suficiente para encontrar una solución que compitiera con las mejores encontradas hasta el momento durante la búsqueda aleatoria.

No se encontró correlación significativa para la tasa de evaporación ni la intensidad de feromonas. Esto se debe a que la intensidad es igual para todas las hormigas, y para todas las iteraciones. Sin embargo, se encontró cierta relación con la desviación estándar. Por ejemplo, para la instancia de 50 nodos, un valor de intensidad pequeño resultó en una mayor

desviación, lo que podría deberse a que a medida que el grafo aumenta su tamaño, los recorridos también lo hacen, por lo que el incremento de la feromona en la matriz es cada vez menos significativo. Puede ser que al obtener números tan pequeños, las hormigas no escogieran los mejores caminos siempre y al final se obtuvieran soluciones muy diferentes entre sí. En cuanto a la tasa de olvido, se encontró cierta correlación con la desviación estándar, pero solo para las instancias de 10 y 30 nodos, por lo que es difícil hacer una apreciación. En términos generales, una tasa de olvido más alta hace que la colonia pierda más información en cada generación, lo que en teoría debería favorecer la exploración. Se sospecha que la exploración no tuvo buenos resultados en este experimento debido al bajo número de iteraciones utilizado y la diferencia de magnitud en los rangos de α y β.

A pesar del sesgo por la explotación, el desempeño del algoritmo con los parámetros definidos tuvo resultados aceptables, reduciendo la tasa de desempeño máxima de 1.35 a 1.095 para una instancia de 40 nodos, y de 1.45 a 1.19 para una instancia de 50 nodos, lo que satisface el objetivo propuesto de optimización de parámetros con búsqueda aleatoria.

Se sugiere mejorar la metodología de optimización de parámetros mediante la utilización de la técnica de búsqueda aleatoria con un mayor número de iteraciones y con rangos de parámetros más precisos, que tengan en cuenta la distancia promedio entre nodos y el tamaño del grafo para evitar problemas de precisión. Además, se recomienda aumentar la cantidad de iteraciones y variar la estrategia utilizada por ACO para evitar el sesgo de explotación y favorecer también la exploración. Es importante tener en cuenta que estos ajustes implican un mayor tiempo computacional, por lo que es conveniente analizar la complejidad del algoritmo para estimar el tiempo aproximado requerido. Además, se sugiere explorar otros métodos de optimización de parámetros y establecer restricciones en los valores de α y β, considerando que ambos no pueden ser altos o bajos al mismo tiempo si tienen la misma magnitud.

VII. Próximos pasos

- Paralelizar el algoritmo ACO para que diferentes grupos de hormigas usen diferentes procesos concurrentes. Esto requiere incluir un mecanismo que permita a los diferentes grupos comunicarse entre sí al final de cada generación para comparar soluciones.
- Probar los programas construidos con grafos de más de 50 nodos.
- Analizar resultados de la complejidad en tiempo del algoritmo ACO para estimar cuánto le tomaría al algoritmo ejecutarse bajo ciertos parámetros: tamaño del grafo, número de hormigas y número de iteraciones (generaciones).
- Utilizar una estrategia diferente a cycle-system para actualizar la matriz de feromonas, para comparar si hay mejoras en el desempeño.
- Utilizar el conocimiento adquirido en la primera fase del proyecto, para definir los parámetros para grafos más grandes y para otras estrategias diferentes a cycle-system.
- Identificar potenciales problemas de precisión al usar un valor de Q pequeño comparado con la distancia entre los nodos y el tamaño del grafo, y explorar la posibilidad de definirlo en base a estos valores.
- Investigar la diferencia de magnitud entre α y β en los valores optimizados, y analizar si es posible usar valores para estos parámetros en la misma magnitud, o si depende de la distancia promedio entre los nodos.
- Evaluar el impacto de evaporación de feromonas en cada iteración con el parámetro ρ.
- Comparar el tiempo de ejecución de la versión paralelizada y no paralelizada.
- Estudiar otras combinaciones de parámetros y con rangos diferentes a las optimizadas con búsqueda aleatoria, por ejemplo favoreciendo la exploración en lugar de la explotación, o variando la tasa de olvido y a intensidad.

References

[1] Reducible. (2018, October 16). Ant Colony Optimization Algorithm - Explanation & Implementation [Video]. YouTube.
[2] Ant colony optimization algorithms - Wikipedia.
[3] ppoffice/ant-colony-tsp: Ant Colony Optimization for TSP problems. https://github.com/ppoffice/ant-colony-tsp
[4] IHPC-G6/Ant-Colony-Optimization: Implementation of Ant Colony Optimization for the Traveling Salesman Problem. https://github.com/IHPC-G6/Ant-Colony-Optimization
[5] IHPC-G6/ant-colony-tsp-py: Improved Python Implementation for ACO to solve TSP problems https://github.com/IHPC-G6/ant-colony-tsp-py
[6] Christofides algorithm - Wikipedia.
[7] Data binning - Wikipedia