Trabajo grupal

Universidad Nacional Mayor de San Marcos
Facultad de Ingeniería de Sistemas e Informática

Curso: Investigación Operativa.
Asignación: Algoritmo basados en colonias de hormigas.
Docente: Richard Cubas Becerra.

2024

Grupo 05:

- Ortiz Herrera, Fabrizio Peter (22200208)
- Pizango Chang, Josue Alejandro (22200215)
- Melendez Blas, Jhair Roussell (22200199)
- Porras Chavez, Miguel Angel (22200036)
- Villanueva Ines, Jose Antonio (22200116)

Índice

1. Introducción y Fundamentos

¿Qué se le viene en mente al escuchar sobre la colonia de hormigas?, una red de insectos que utiliza la inteligencia colectiva para un fin?, pues es de esto que se hablará en el presente informe. Este algoritmo es una metaheurística, la cual fue inspirada en el comportamiento de colonias enteras de hormigas. El algoritmo del cual hablamos, ha sido propuesto por Marco Dorigo en 1992, esto como una solución al TSP (Problema del Viajante).

Como hemos mencionado , este enfoque utiliza la inteligencia colectiva entre hormigas las cuales se comunican indirectamente a través de feromonas, así, descubriendo soluciones óptimas en problemas de rutas complejas.

> ‘‘Las hormigas se comunican a través de sus feromonas, las cuales son sustancias que les permiten encontrar los caminos más cortos entre su nido y la fuente de alimentos. Esta característica ha sido utilizada para la solución de problemas de optimización que necesiten mejorar sustancialmente los tiempos de cómputo para la solución de una aplicación específica.’’
>
> Algarín, (2010). Optimización por colonia de hormigas: aplicaciones y tendencias, pag. 1

1.1 Fundamentos biológicos

Las hormigas son insectos fascinantes, si se puede decir que son las más organizadas, pues este es su punto, la autoorganización, en el cual logran resolver problemas de una alta complejidad como encontrar rutas cortas entre su nido a por ejemplo, un alimento.

Este comportamiento se basa en:

1.1.1 Feromonas

Las hormigas dejan feromonas mientras se mueven, creando un rastro lo que se puede describir como un trazo en rojo de un humano en una figura. Cuanto más corto sea el camino, más feromonas se acumulan ya que otras hormigas las siguen.

1.1.2 Comportamiento Estocástico

Las hormigas exploran el entorno de forma aleatoria, sin un rumbo al iniciar, lo que permite que se descubran nuevas soluciones. Hay un balance entre la explotación de rutas conocidas y la exploración de otras rutas, pues este equilibrio es la base para encontrar la futura solución óptima.

1.1.3 Refuerzo positivo

Como se ha mencionado, los rastros de feromonas se fortalecen si otras hormigas siguen el camino, volviéndose frecuente, mientras que se debilitan con el tiempo debido a la evaporación. Pues así, las hormigas descartan caminos largos.

1.2 La metaheurística como herramienta de optimización

La metaheurística es un enfoque general, el cual aplica un apartado de problemas de optimización, especialmente en los que son demasiado complejos para ser resueltos a mano en un corto tiempo. Claro, el algoritmo del cual hablamos si pertenece a esta categoría por tres razones:

Simula comportamientos naturales: Aprovecha la biología como base para resolver entornos que se hacen en computadora.

Flexibilidad: El algoritmo es aplicativo en varios problemas, no solo en un solo tipo, por ejemplo, asignación de tareas, diseños de redes de colegios, optimización de recursos, etc.

Enfoque iterativo: Ya que genera soluciones iniciales, como?, este algoritmo lo mejora mediante iteraciones sucesivas hasta converger en soluciones óptimas

2. Explicación del Algoritmo

El Algoritmo de Colonia de Hormigas (ACO) está inspirado en la observación del comportamiento de las hormigas cuando buscan las rutas más eficientes hacia las fuentes de alimentos. El algoritmo simula este comportamiento natural mediante la creación de hormigas artificiales que exploran el espacio de soluciones de un problema específico. Cada hormiga construye una solución paso a paso, eligiendo entre diferentes opciones basándose en dos tipos de información:

- **Información Heurística:** Representa el conocimiento previo sobre el problema, como distancias entre nodos en un grafo o costos asociados a ciertas decisiones. Esta información guía a las hormigas hacia opciones prometedoras desde el inicio.
- **Rastros de Feromonas Artificiales:** Simulan las feromonas reales depositadas por las hormigas, reflejando la deseabilidad de ciertos caminos basados en las soluciones previamente encontradas. A medida que las hormigas exploran y encuentran soluciones, incrementan la concentración de feromonas en las aristas de sus rutas, incentivando a futuras hormigas a seguir caminos exitosos.

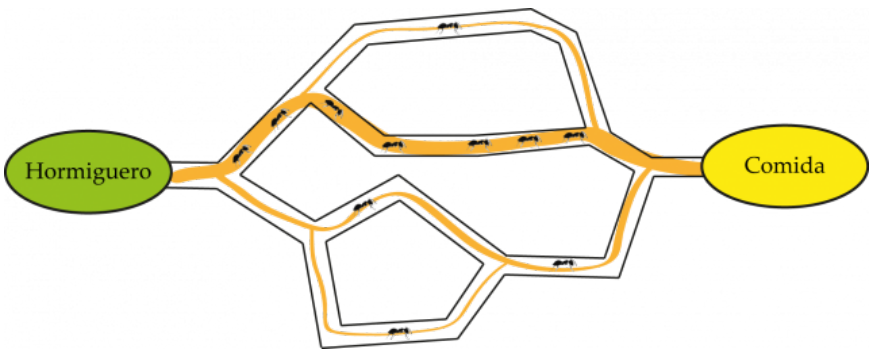


Figura 1. Representación de la ruta de las hormigas desde su hogar hacia el alimento.

2.1 Inicialización

Se inicia el proceso cuando las hormigas artificiales se colocan inicialmente en nodos predefinidos del grafo , cada hormiga comienza su recorrido desde su nodo inicial. Una vez colocadas, las hormigas inician la construcción de sus soluciones paso a paso. En cada paso, una hormiga debe decidir a qué nodo moverse a continuación. Esta decisión se basa en la fórmula de probabilidad que combina la influencia de la feromona y la información heurística.

Fórmula de Probabilidad :

$$P_{ij} = \frac{(\tau_{ij})^{\alpha} \cdot (n_{ij})^{\beta}}{\sum_{k \in vecinos} (\tau_{ik})^{\alpha} \cdot (n_{ik})^{\beta}}$$

Donde :

P_{ij} : $es\ la\ probabilidad\ de\ seleccionar\ la\ arista\ (i, j).$

τ_{ij} : $es\ el\ valor\ del\ rastro\ de\ feromonas\ en\ la\ arista\ del\ nodo\ i\ al\ nodo\ j.$

$n_{ij} = \frac{1}{d_{ij}}\ es\ la\ información\ heurística,\ con\ d_{ij}\ representando\ la\ distancia\ o\ coste\ entre\ los\ nodos\ i\ y\ j.$

α : $parámetro\ que\ controla\ la\ influencia\ de\ la\ feromona.$

β : $parámetro\ que\ controla\ la\ influencia\ de\ la\ heurística.$

La hormiga identifica todos los nodos vecinos al nodo actual *i* que aún no han sido visitados o que son candidatos válidos según las restricciones del problema y se aplica la fórmula de probabilidad para determinar P_{ij} para cada nodo vecino *j* .

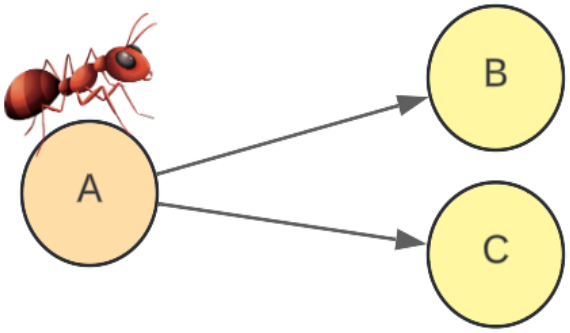


Figura 2. Una hormiga estando en el nodo i, a punto de elegir un nodo j.

2.2 Actualización de Feromonas

Después de que todas las hormigas han construido sus soluciones, es necesario actualizar las cantidades de feromona en las aristas para reflejar la calidad de las soluciones encontradas. Esta actualización se realiza en dos pasos: evaporación y depósito de feromonas.

Evaporación de Feromonas: Se reduce la cantidad de feromona en todas las aristas para evitar la convergencia prematura hacia soluciones subóptimas. La evaporación se calcula mediante:

$$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij}$$

donde $0 < \rho \leq 1$ es la velocidad de evaporación de feromona. El parámetro ρ se utiliza para evitar la acumulación ilimitada de los rastros de feromona y permite que el algoritmo elimine las malas decisiones tomadas previamente.

Depósito de Feromonas: Cada hormiga que ha recorrido una arista (i,j) deposita una cantidad de feromona proporcional a la calidad de su solución. Esto se expresa como:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k}$$

Donde :

$$\Delta\tau_{ij}^{k} = \frac{Q}{L_k} \; si\ la\ hormiga\ k\ usa\ la\ arista\ (i,j) \quad \vee \quad \Delta\tau_{ij}^{k} = 0\ en\ otro\ caso$$

$$Q : Constante\ que\ determina\ la\ cantidad\ total\ de\ feromona\ depositada.$$

L_k : $Longitud\ de\ la\ solución\ de\ la\ hormiga\ k.$

m : $Número\ total\ de\ hormigas.$

Al concluir el algoritmo, se retorna la mejor ruta encontrada como la solución óptima o cercana a óptima para el problema planteado. Esta ruta representa la combinación más eficiente de decisiones tomadas por las hormigas durante todo el proceso de optimización.

3. Aplicaciones Prácticas

A lo largo del tiempo se ha podido apreciar que el algoritmo de la colonia de hormigas (ACO) ha ganado un gran valor impresionante, dado que ha experimentado múltiples enfoques hasta transformarse en un instrumento para solucionar diversos problemas complicados de optimización combinatoria.

A modo de recordatorio, podemos mencionar que se destaca su eficiencia al explorar grandes espacios de soluciones y encontrar rutas efectivas mediante el depósito de feromonas, en aquellos problemas donde la solución exacta es difícil de obtener debido a la alta complejidad o el gran tamaño del problema.

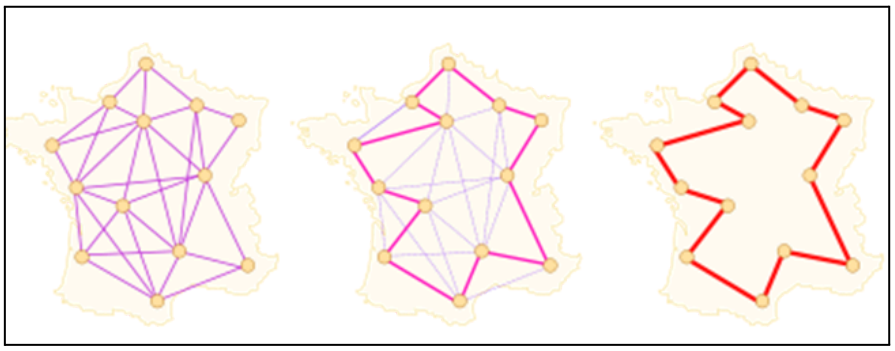
Gracias a su comportamiento muy semejante a la naturaleza, dicho algoritmo es oportuno para diversas tareas que serán ejemplificadas en esta sección. En ese sentido, exploramos diferentes ejemplos prácticos, tales como, el Problema del Viajante de Comercio (TSP), la optimización de rutas en Logística y Transporte, la optimización de redes de telecomunicaciones, el problema de la Mochila y la optimización de Circuitos Electrónicos, con la intención de mostrar su capacidad de adaptación para solucionar problemas reales en diversos campos.

3.1. Problema del Viajante de Comercio (TSP):

El Problema del Viajante TSP consiste en encontrar la ruta más corta para un viajante con la condición de que debe visitar varias ciudades pasando por cada una de ellas exactamente una vez y regresando al punto inicial de partida. Por ejemplo nos dan un conjunto de ciudades con sus respectivas distancias entre ellas, nuestra tarea será encontrar la mejor combinación de rutas entre ciudades cumpliendo con los parámetros establecidos. El presente problema se considera NP-completo, ya que no existe una solución eficiente para un gran número de ciudades usando métodos exactos.

Figura 3.

Representación gráfica del Problema del Viajante de Comercio.



Nota. Esta figura representa la ruta más óptima en un grafo en específico que simula el problema de TSP. Tomado de: https://www.cs.us.es/~fsancho/Blog/posts/img/tsp1.png

3.1.1. ¿Por qué ACO es adecuado?

El algoritmo de la colonia de hormigas es muy adecuado en estos casos porque tiene la capacidad para explorar eficientemente grandes espacios de

solución. La razón se basa en que ACO simula el comportamiento de las hormigas que dejan feromonas en las rutas que eligen. Haciendo una analogía con el presente problema, podemos apreciar que el objetivo tanto de las hormigas como del viajante es encontrar la ruta más corta. Si el camino es más corto, entonces se depositan más feromonas por esta, aumentando la probabilidad de que otras hormigas (el viajante) sigan el mismo camino. Durante el proceso interactivo, las hormigas convergen hacia la mejor solución posible, encontrando efectivamente el camino o la ruta más cortas.

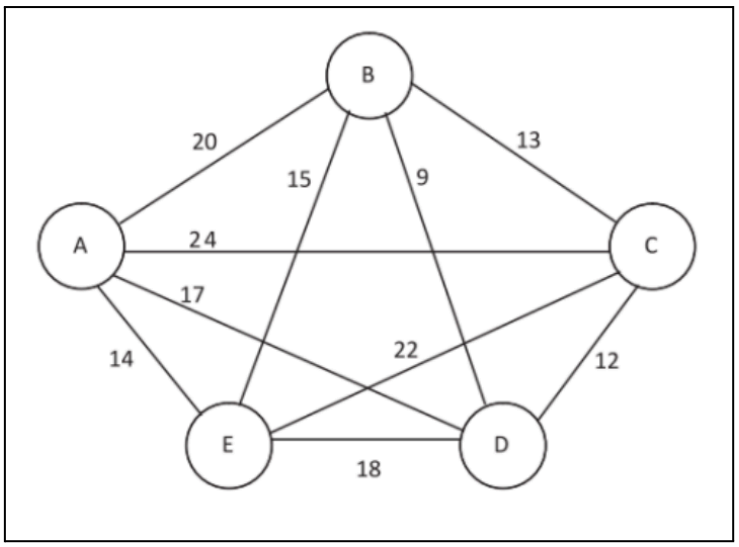
3.1.2. Caso relacionado

Ejercicio 3 - Práctica Calificada 02 del curso de Investigación Operativa G02

Un repartidor de Mercado Libre es responsable de entregar paquetes a 5 clientes (A, B, C, D y E) en una determinada zona. El repartidor siempre comienza su ruta desde el cliente A, realiza una entrega a cada uno de los demás clientes una sola vez y luego regresa al punto de partida. La red de rutas con las respectivas distancias entre los clientes (en km) se muestra en la siguiente Figura.

Figura 4.

Imagen del ejercicio 3 de la práctica calificada 02 del curso de Investigación operativa.



Nota. La presente figura muestra un grafo en donde los nodos simulan ser los clientes a los que el repartidor debe llegar y las aristas representan las distancias que hay entre estos, todo lo mencionado representa lo necesario para resolver el ejercicio 02 del curso de investigación operativa. ESCUELA DE INGENIERÍA DE SISTEMAS. (2024). *Práctica calificada: Investigación operativa* [Documento de clase]. Facultad de Ingeniería de sistemas e Informática (FISI).

Determinar la ruta óptima que debe seguir el repartidor para minimizar la distancia total recorrida, utilizando técnicas de programación dinámica.

Resolución:

Dado que el recorrido debe comenzar y terminar en el cliente A, pasando por todos los demás clientes y cubriendo la menor distancia posible, este problema se puede asociar con el concepto de caminos hamiltonianos, buscando aquel que minimice la ruta total.

Para calcular el número de ciclos hamiltonianos posibles, utilizamos la fórmula:

$$\text{Número de ciclos hamiltonianos} = \frac{(n-1)!}{2}$$

Aplicamos al problema:

$$\frac{(4)!}{2} = \ 12\ ciclos.$$

Tabla 1.

Tabla de todas las rutas hamiltonianas del ejercicio 3.

n°	V1	V2	V3	V4	V5	Volvemos al mismo	Distancia entre el nodo V_n y V_{n+1}					Suma de las rutas
1	a	b	c	d	e	a	20	13	12	18	14	77
2	a	b	c	e	d	a	20	13	22	18	17	90
3	a	b	d	c	e	a	20	9	12	22	14	77
4	a	b	d	e	c	a	20	9	18	22	24	93
5	a	b	e	d	c	a	20	15	18	12	24	89
6	a	c	b	d	e	a	24	13	9	18	14	78
7	a	c	b	e	d	a	24	13	15	18	17	87
8	a	c	d	b	e	a	24	12	9	15	14	74
9	a	d	c	b	e	a	17	12	13	15	14	71
10	a	d	c	e	b	a	17	12	22	15	20	86
11	a	d	b	e	c	a	17	9	15	22	24	87
12	a	d	b	c	e	a	17	9	13	22	14	75

Nota. La presente tabla muestra todas las rutas hamiltonianas del ejercicio 3, además de las distancias de cada una de estas para su posterior comparación. Ayudará a identificar rápidamente la distancia más corta y más larga del ejercicio en cuestión. Elaboración propia.

De esta forma garantizamos no volver a recorrer el mismo nodo (cliente) tomando en cuenta las demás restricciones mencionadas en el problema. Podemos apreciar que la distancia óptima (mínima) es de 71 Km.

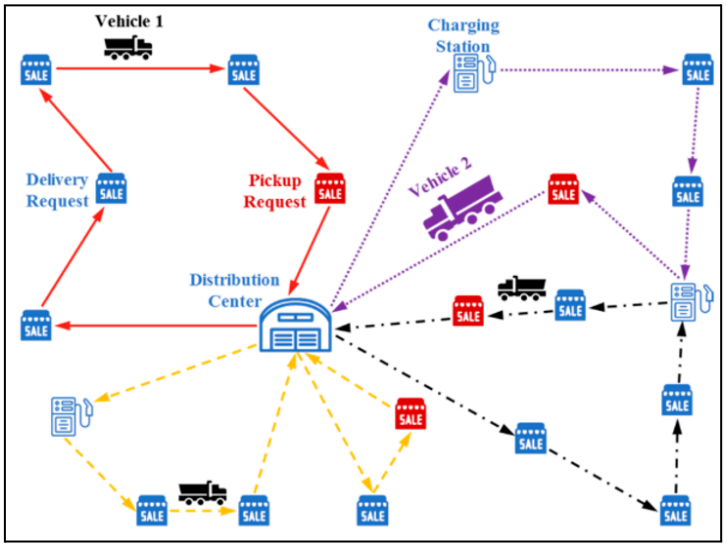
La aplicación del código relacionado a este problema y derivado del algoritmo de colonia de las hormigas se explica en la última sección del documento.

3.2. Optimización de Rutas en Logística y Transporte:

El propósito de la optimización de rutas en logística y transporte es diseñar las rutas más eficaces para para entregar cargas a través de una flota de vehículos. Este problema es conocido comúnmente como el problema de generación de rutas de vehículos (VRP) que abarca varios elementos como la capacidad del vehículo, el tiempo de entrega, las restricciones geográficas y la optimización en el uso de recursos. La intención es reducir los gastos de transporte como el tiempo o el combustible y, simultáneamente, cumplir con todas las restricciones del problema.

Figura 5.

Representación gráfica del problema de Optimización de Rutas en Logística y Transporte.



Nota. Esta figura representa la ruta de diferentes vehículos con paradas múltiples que simula el problema de Optimización de Rutas en Logística y Transporte. Tomado de: https://es.linkedin.com/pulse/modelos-vrp-fabian-cerda

3.2.1. ¿Por qué ACO es adecuado?

En este caso, ACO es adecuado para VRP porque tiene la capacidad para resolver problemas con múltiples variables y restricciones como es el presente caso. Al igual que el TSP, las hormigas depositan feromonas a través de las vías más eficientes, esto ayuda a encontrar mejores soluciones (rutas). Además, la ventaja de este algoritmo es que permite que las decisiones de enrutamiento se ajusten dinámicamente en función de las restricciones y condiciones cambiantes, lo que lo convierte en una herramienta poderosa para la optimización de rutas en escenarios logísticos complejos.

3.2.2. Estudio de caso relacionado

Algoritmo basado en la optimización mediante colonias de hormigas para la resolución del problema del transporte de carga desde varios orígenes a varios destinos

Este artículo escrito por Barcos (2015) trata sobre cómo mejorar las rutas de transporte de carga entre diferentes puntos de origen y destino, este es un problema muy común en empresas de paquetería y logística. El objetivo es encontrar las rutas más eficientes para mover sus mercancías, pero debe considerar distintos tipos de caminos y la posibilidad de usar hubs (lugares donde se transbordan cargas). Además de ello, Barcos (2015) analiza rutas en las que un vehículo puede hacer varias paradas para recoger o entregar mercancías antes de llegar a su destino final.

Podemos ver el problema desde la perspectiva de cómo un camión o vehículo consigue optimizar su recorrido eficientemente, a través de la

elección entre rutas directas, rutas que pasan a través de uno o más hubs o también rutas con paradas intermedias. El sistema busca minimizar los costos de transporte, además de cumplir con los tiempos de entrega, ya que en algunos casos se impone como límite 24 horas o 48 horas, pero esta depende de la carga y los destinos involucrados.

En este caso, el algoritmo ACO ayuda a los vehículos a encontrar las rutas más económicas y eficientes, a medida que las "hormigas" (simulaciones del algoritmo) exploran diversas alternativas de rutas.

El autor menciona que el estudio se aplica a una parte de España, donde se analizan los flujos de carga entre diferentes delegaciones y terminales de ruptura de carga (hubs). Se busca obtener soluciones para una empresa de paquetería como SEUR, que necesita asegurar que las mercancías lleguen a su destino en un plazo determinado y con el menor coste posible. En la práctica, se evalúan rutas entre ciudades y poblaciones en el territorio español, donde los vehículos deben ser asignados a diferentes rutas dependiendo de la carga y las prioridades de entrega.

3.2.2.1. Objetivo principal

El objetivo principal del trabajo es desarrollar una solución eficiente para el transporte de carga entre delegaciones, optimizando el costo del sistema sin sacrificar el nivel de servicio. El problema se resuelve mediante un algoritmo ACO que primero optimiza las rutas directas y con uno o dos hubs, y luego mejora las rutas utilizando el enfoque de peddling.

3.2.2.2. Conclusión

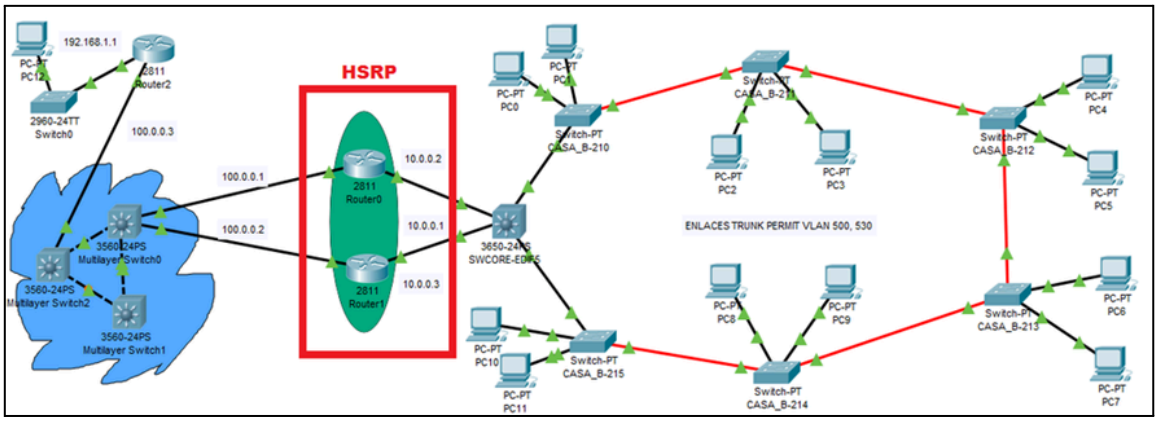
Finalmente concluye que el algoritmo ACO mostró buenos resultados al comparar las soluciones obtenidas con las de la programación entera y fue capaz de resolver un caso real de manera efectiva, pues logró reducir los costos y mejorar la eficiencia en el transporte de carga entre delegaciones.

3.3. Optimización de Redes de Telecomunicaciones:

Con respecto a la optimización de redes de telecomunicaciones, podemos mencionar que el objetivo principal es optimizar el tráfico de datos y la distribución de paquetes a través de la red. Esto consiste en poder encontrar las mejores rutas entre los diferentes nodos de la red, con el objetivo de minimizar la congestión y mejorar la eficiencia de la trasmisión de los datos. El problema se complica ya que existen múltiples rutas posibles que un paquete pueda tomar para llegar a su destino, sumado a ello, existen restricciones relacionadas con la capacidad de los enlaces y el tráfico de datos.

Figura 6.

Representación gráfica de la topología física de una red cualquiera.



Nota. Esta figura representa la topología física de una red cualquiera en donde la comunicación entre computadoras puede tomar diferentes caminos para llegar a su destino. Tomado de: https://community.cisco.com/t5/image/serverpage/image-id/41322i35A3D9CACC347E6B/image-size/large?v=v2&px=999

3.3.1. ¿Por qué ACO es adecuado?

En este problema, ACO es una excelente alternativa ya que posee la habilidad de explorar una extensa variedad de rutas potenciales y ajustarse de manera dinámica a las condiciones de la red. El algoritmo tiene la capacidad de modificar las rutas en tiempo real, promoviendo las rutas menos saturadas, debido a las feromonas que orientan las decisiones de las hormigas. Esto facilita la identificación de las soluciones más eficaces, incluso en redes de gran complejidad y tráfico en constante cambio.

3.3.2. Estudio de caso relacionado:

Optimización de la transmisión de datos en una red de routers mediante el algoritmo de la colonia de hormigas

En este trabajo de investigación fue escrito por Cruz (2015) y se está diseñando e implementando un protocolo de enrutamiento para una red de routers, basado en el algoritmo de la colonia de hormigas (ACO), con el objetivo de mejorar la transmisión de datos en redes de datos.

El algoritmo ACO se utiliza para encontrar el mejor camino en la red imitando el comportamiento de las colonias de hormigas que, mediante la exploración y el uso de feromonas, marcan las rutas óptimas. El autor, para

comprobar la eficiencia del algoritmo realiza pruebas comparativas con los protocolos tradicionales de enrutamiento RIP y OSPF. El algoritmo fue implementado en MATLAB y se probaron diferentes escenarios, como el envío de archivos comprimidos entre un servidor y un cliente Linux.

3.3.2.1. Objetivo principal

El objetivo principal de la investigación fue estudiar y determinar si el algoritmo de la colonia de hormigas (AntNet) podría ser utilizado para elegir el mejor camino para el enrutamiento de datos en una red de routers, comparando su rendimiento con los protocolos RIP y OSPF, los cuales son ampliamente utilizados en la actualidad.

3.3.2.2. Conclusión

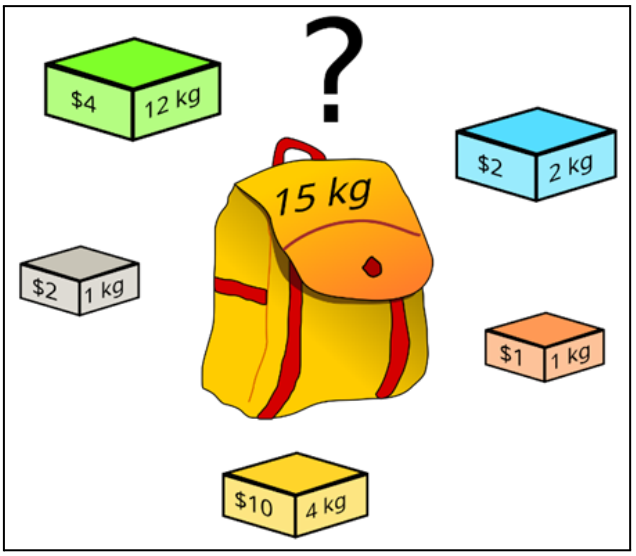
Finalmente el autor Barcos (2015) llega a la conclusión de que el algoritmo de la colonia de hormigas (AntNet) demostró ser una herramienta útil para el enrutamiento de datos en redes de routers. En las pruebas realizadas, el algoritmo ACO mostró un rendimiento superior en términos de velocidad de transmisión de datos en comparación con OSPF y RIP, siendo un 2.16% más rápido que OSPF y un 0.45% más rápido que RIP. A nivel estadístico, se comprobó que existía una diferencia significativa entre el rendimiento de ACO y OSPF, pero no entre ACO y RIP. En general, el algoritmo ACO puede ser una opción confiable para mejorar la eficiencia de la transmisión de datos en redes reales, lo que lo convierte en una base prometedora para futuras investigaciones en protocolos de capa de Internet.

3.4. Problema de la Mochila:

Ahora les presentamos nuevamente el Problema de la mochila que muchas veces lo hemos visto a lo largo de nuestros ciclos académicos, esta implica seleccionar un conjunto de objetos con diferentes pesos y valores para maximizar el valor total, pero sin exceder la capacidad de la mochila. Es un problema de optimización combinatoria que se aplica en situaciones como la selección de inversiones, asignación de recursos o diseño de productos, donde se deben tomar decisiones de inclusión o exclusión de elementos teniendo presente la limitación de capacidad.

Figura 7.

Representación gráfica común del problema de la mochila.



Nota. Esta figura representa el problema de la mochila, en donde se tiene objetos de diferentes pesos con valor monetario distinto, además de una mochila con capacidad máxima, la intención en maximizar el aspecto económico. Tomado de:

https://upload.wikimedia.org/wikipedia/commons/f/fd/Knapsack.svg

3.4.1. ¿Por qué ACO es adecuado?

En el presente problema (Problema de la Mochila) ACO es adecuado porque permite explorar eficazmente las combinaciones de elementos a incluir, utilizando el proceso de feromonas para guiar la búsqueda hacia las mejores soluciones. Dado que el algoritmo puede manejar múltiples restricciones, como el peso máximo de la mochila, y encontrar soluciones cercanas a la óptima en espacios grandes, es una excelente opción para abordar este tipo de problemas de optimización.

3.4.2. Estudio de caso relacionado:

ACHPM: ALGORITMO DE OPTIMIZACIÓN CON COLONIA DE HORMIGAS PARA EL PROBLEMA DE LA MOCHILA

El artículo escrito por Ponce (2006) presenta un algoritmo de optimización basado en colonias de hormigas para resolver el Problema de la Mochila, un problema NP-duro comúnmente utilizado en diversas aplicaciones como asignación de procesos en sistemas distribuidos y planificación de presupuestos de capital. El problema de la mochila implica seleccionar un subconjunto de objetos con beneficios y pesos determinados, con el fin de maximizar la utilidad total sin exceder la capacidad de la mochila (contenedor).

El algoritmo implementado se basa en que cada "hormiga" busca una solución candidata al problema, guiada tanto por la información heurística relacionada con el problema como por las feromonas dejadas por otras hormigas.

El algoritmo fue implementado en el lenguaje de programación C y se evaluó mediante varios archivos de datos de prueba. El formato para representar las instancias del problema fue desarrollado utilizando archivos de texto. Los resultados de la implementación se analizaron para verificar la efectividad del algoritmo.

3.4.2.1. Objetivo Principal

El objetivo principal del artículo fue desarrollar un algoritmo de optimización con colonia de hormigas para resolver el Problema de la Mochila, maximizando la utilidad total de los objetos seleccionados sin exceder la capacidad de la mochila. Además, se buscó explorar si este algoritmo puede ofrecer soluciones óptimas o aproximadas para este tipo de problemas NP-duros.

3.4.2.2. Conclusiones

Finalmente el autor Ponce (2006) concluyó que el algoritmo de Colonia de Hormigas implementado proporciona soluciones óptimas para el Problema de la Mochila en las pruebas realizadas. Sin embargo, dado que el problema de la mochila es NP-duro, el algoritmo se vuelve más demandante en términos de tiempo de ejecución conforme aumenta el número de objetos a considerar. Por lo tanto, se recomienda explorar variantes o mejoras del algoritmo de Colonia de Hormigas, o combinarlo con otros métodos heurísticos, para abordar problemas de mayor escala de manera más eficiente. El uso de algoritmos heurísticos ha demostrado ser una opción efectiva para obtener buenos resultados

en una amplia gama de problemas de optimización, como en el caso del problema de la mochila.

4. Ventajas, Limitaciones y Comparaciones

En este apartado se van a examinar las ventajas que posicionan al ACO como una herramienta clave en diversas aplicaciones, las restricciones que pueden influir en su rendimiento y su contraste con otros algoritmos.

4.1 Ventajas del algoritmo de Colonia de Hormigas

El Algoritmo de Colonia de Hormigas (ACO), inspirado en el comportamiento colaborativo de las hormigas al encontrar rutas óptimas, se ha posicionado como una técnica de optimización altamente eficaz y adaptable. A continuación, se destacan sus principales ventajas, que lo hacen sobresalir en distintos ámbitos, junto con otras características que refuerzan su capacidad y versatilidad.

- **Adaptabilidad y Escalabilidad**

  El ACO destaca por su capacidad de adaptarse a problemas dinámicos, donde las condiciones pueden variar mientras el algoritmo está en funcionamiento. Por ejemplo, en el ámbito logístico, puede responder en tiempo real a cambios en las demandas o en las restricciones de las rutas. Asimismo, su habilidad para manejar problemas de mayor complejidad lo posiciona como una solución eficaz en el diseño de redes de comunicación y sistemas de transporte.

- **Evitar mínimos locales**

  Gracias a su enfoque probabilístico, el ACO le permite explorar varias soluciones de forma simultánea, disminuyendo considerablemente la probabilidad de quedar atrapado en mínimos locales, un desafío frecuente en algoritmos deterministas como el descenso de gradiente. Esta característica lo

hace particularmente valioso para problemas de optimización combinatoria con espacios de búsqueda amplios y complejos.

- **Paralelismo**

  Gracias a su diseño intrínseco, facilita la implementación en paralelo, permitiendo que varias "hormigas" examinen diferentes soluciones de forma independiente. Esto no sólo acelera el tiempo de ejecución, sino que también lo hace especialmente adecuado para sistemas con arquitecturas multiprocesador, optimizando el aprovechamiento de los recursos computacionales en aplicaciones prácticas.

- **Flexibilidad**

  Ofrece una alta capacidad de configuración, lo que le permite adaptarse a problemas específicos mediante el ajuste de sus parámetros. Esta flexibilidad lo convierte en una herramienta versátil, aplicable en diversas áreas como la optimización de redes de telecomunicaciones, la planificación de tareas, el diseño de circuitos electrónicos y el control del tráfico.

- **Capacidad de aprendizaje continuo**

  Este algoritmo emplea un mecanismo de retroalimentación positiva mediante la actualización de feromonas, lo que le permite aprender de soluciones previas y reforzar aquellas que demuestran ser más efectivas. Este proceso contribuye a una mejora continua del rendimiento a lo largo del tiempo.

4.2 Limitaciones del algoritmo de Colonia de Hormigas

El Algoritmo de Colonia de Hormigas (ACO) es una herramienta eficaz para optimizar problemas complejos, pero, como cualquier enfoque, presenta ciertas limitaciones que deben ser tomadas en cuenta al implementarlo. Aunque es una

técnica versátil, su rendimiento puede verse influenciado por diversos factores, y algunos aspectos de su implementación pueden resultar desafiantes. A continuación, se describen las principales restricciones del ACO y se analizan otros factores que pueden afectar su efectividad.

- **Ajuste de parámetros**

  El rendimiento del ACO depende en gran medida de la correcta configuración de sus parámetros, como la tasa de evaporación de las feromonas, el número de hormigas y la intensidad de la retroalimentación. Estos parámetros pueden variar según las características del problema a resolver, y su ajuste adecuado es fundamental para obtener resultados satisfactorios. Sin embargo, encontrar la combinación óptima de estos parámetros no es una tarea sencilla y, frecuentemente, requiere un proceso iterativo de prueba y error, lo que puede resultar largo y trabajoso.

- **Análisis teórico complejo**

  Debido a su naturaleza estocástica, el ACO presenta dificultades para ser analizado matemáticamente de forma rigurosa, especialmente en cuanto a su convergencia. A diferencia de los algoritmos deterministas, resulta complicado predecir su comportamiento a medida que progresa, lo que puede complicar la evaluación de su rendimiento en situaciones particulares o su comparación con otras técnicas de optimización.

- **Requerimientos computacionales**

  En problemas de gran escala, el ACO puede resultar costoso en términos computacionales, ya que implica la simulación de múltiples agentes (hormigas) que exploran el espacio de soluciones de manera paralela. Esto puede generar un alto consumo de recursos, tanto en tiempo como en

memoria. En situaciones con recursos limitados o cuando se requiere una respuesta rápida, el ACO podría no ser la opción más eficiente.

- **Convergencia Lenta**

  En ciertos casos, el ACO puede mostrar una convergencia lenta hacia la solución óptima, particularmente cuando el espacio de búsqueda es extenso o cuando las soluciones iniciales no son adecuadas. Aunque el algoritmo dispone de mecanismos para mejorar progresivamente su rendimiento, en problemas altamente complejos puede ser necesario ejecutarlo durante períodos prolongados para obtener resultados satisfactorios.

- **Sensibilidad al tipo de problema**

  Aunque el ACO es una técnica flexible, no siempre es la opción más adecuada para todos los tipos de problemas. Su rendimiento puede verse influido por la estructura del espacio de soluciones o por características particulares del problema, como restricciones estrictas o objetivos no lineales. En tales situaciones, otros métodos de optimización podrían ser más eficaces.

4.3 Comparaciones con otras metaheurísticas

En optimización, se han creado diversas metaheurísticas para abordar problemas complejos, cada una adecuada para ciertos casos. Se va a realizar la comparación del Algoritmo de Colonia de Hormigas (ACO) con otras metaheurísticas, como los Algoritmos Genéticos (AG) y el Recocido Simulado (RS), enfocándose en aspectos como el mecanismo de búsqueda, la velocidad de convergencia, la capacidad de evitar mínimos locales y su aplicabilidad, para determinar en qué situaciones ACO es más eficaz.

Criterios de comparación	Algoritmo de colonia de hormigas	Algoritmos genéticos	Recocido simulado
Mecanismos de búsqueda	Utiliza feromonas para guiar la exploración del espacio de soluciones.	Usa selección, cruce y mutación para explorar el espacio.	Basado en la temperatura, permite escapar de los mínimos locales.
Exploración vs Explotación	Enfoque más exploratorio, mejor para encontrar soluciones óptimas.	Equilibrio entre exploración y explotación, pero más lento en converger.	Explotación gradual con enfriamiento, más lento en escapar de mínimos locales.
Velocidad de Convergencia	Convergencia más rápida, especialmente en problemas combinatorios.	Más lenta debido a la dependencia de operadores genéticos..	Requiere más iteraciones para obtener resultados óptimos.
Capacidad de escapar de mínimos locales	Alta capacidad de evitar mínimos locales gracias a la retroalimentación positiva.	Puede quedar atrapado en mínimos locales si los operadores no son adecuados.	Depende del enfriamiento, puede ser menos efectivo en escapar de mínimos locales.

En conclusión, el Algoritmo de Colonia de Hormigas se destaca frente a los Algoritmos Genéticos y el Recocido Simulado por su capacidad de converger rápidamente y explorar eficientemente, lo que lo hace ideal para problemas de optimización combinatoria. Aunque los AG y el RS también son útiles, ACO es generalmente más efectivo en problemas dinámicos y de gran escala, donde se requiere rapidez y robustez en la búsqueda de soluciones.

5. Problemas y Códigos

5.1 Planificación de Celdas en Redes.

Este problema también es conocido como el Diseño de Redes de Radio . Se trata de colocar el número mínimo de antenas para cubrir el máximo de una zona.

Este problema en la vida real es muy complejo, por eso la mayoría de Autores.

(Calégari, Guidec, Kuonen, & Kobler, 1997) usan o desarrollan una manera discreta del problema. Digamos que tienes un tablero de 49 x 49 sectores . Si yo quiero que la transmisión de señal llegue a todo el tablero puedo poner un transmisor en cada sector. Pero eso sería muy caro e innecesario. Por eso nosotros identificamos mejores lugares donde puede ir un transmisor y lo marcamos con un 2. Los números 0 son los lugares donde no puede ir un transmisor pero llega señal por un transmisor puesto en una celda 2

Rejilla de 10 x 10 celdas de ejemplo.

Para resolver el problema debemos buscar la menor cantidad de antenas instaladas en los puntos disponibles , pero que cubra la mayor cantidad de espacio.

5.1.1 Código Ejemplo 3.

Declaración de parámetros para el algoritmo

```
n_hormigas = 10
n_iteraciones = 50
feromona_inicial = 0.1
alpha = 1
beta = 3
evaporacion = 0.5
q0 = 0.9
phi = 0.1
```

Declaración de variables de entrada.

```
# Radio de cobertura como variable externa
radio_cobertura = 2

# Matriz de entrada (2: ubicaciones candidatas, 0: no candidato)
matriz_entrada = np.array([
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 2, 0, 0, 0, 0],
```

```
    [0, 0, 2, 0, 0, 0, 0, 2, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 2, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 2, 0, 0, 0, 0, 2],
    [0, 0, 2, 0, 0, 0, 2, 2, 0, 0],
    [2, 0, 0, 0, 0, 0, 0, 0, 0, 0],
    [0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
])
```

Generamos la matriz que nos indicará qué ubicaciones se pueden cubrir con las antenas candidatas . También el vector con las feromonas. Y la heurística que va a ser el criterio para calificar la solución

```
def generar_matriz_cobertura(matriz_entrada, radio_cobertura):
    n_filas, n_columnas = matriz_entrada.shape
    matriz_cobertura = np.zeros_like(matriz_entrada)

    ubicaciones_candidatas = np.argwhere(matriz_entrada == 2)

    for fila, columna in ubicaciones_candidatas:
        for i in range(max(0, fila - radio_cobertura), min(n_filas, fila + radio_cobertura + 1)):
            for j in range(max(0, columna - radio_cobertura), min(n_columnas, columna +
radio_cobertura + 1)):
                matriz_cobertura[i, j] = 1

    return matriz_cobertura, ubicaciones_candidatas

matriz_cobertura, ubicaciones_candidatas = generar_matriz_cobertura(matriz_entrada,
radio_cobertura)
feromonas = np.full(len(ubicaciones_candidatas), feromona_inicial)
heuristica = np.sum(matriz_cobertura, axis=(0, 1))
```

Ahora la clase Hormiga. Esta tendrá su solución, su matriz cubierta y el costo de su solución.

En esta se va a calcular la probabilidad de escoger cada ubicación posible.

Después esta va a ser escogida de manera determinista o por probabilidad.

Después se va a actualizar las celdas cubiertas si se escoge o no una antena posible.

Después se actualiza la feromona localmente.

Y esto se repetirá hasta que todas las celdas que se puedan cubrir estén cubiertas.

```
class Hormiga:
    def __init__(self):
        self.solucion = np.zeros(len(ubicaciones_candidatas), dtype=int)
        self.matriz_cubierta = np.zeros_like(matriz_entrada)
        self.costo = 0

    def construir_solucion(self):
        global feromonas, heuristica
        celdas_por_cubrir = matriz_cobertura.copy()

        while np.any(celdas_por_cubrir > 0):
            # Calcular probabilidad de elegir cada ubicación
            probabilidades = (feromonas ** alpha) * (heuristica ** beta)
            probabilidades /= np.sum(probabilidades)

            if np.random.rand() < q0:
                # Selección determinística
                indice = np.argmax(probabilidades)
            else:
                # Selección probabilística
                indice = np.random.choice(len(ubicaciones_candidatas), p=probabilidades)

            # Añadir antena en la ubicación seleccionada
            self.solucion[indice] = 1
            fila, columna = ubicaciones_candidatas[indice]

            for i in range(max(0, fila - radio_cobertura), min(matriz_entrada.shape[0], fila +
radio_cobertura + 1)):
                for j in range(max(0, columna - radio_cobertura), min(matriz_entrada.shape[1],
columna + radio_cobertura + 1)):
                    celdas_por_cubrir[i, j] = 0
                    self.matriz_cubierta[i, j] = 1

            # Actualización local de feromonas
            feromonas[indice] = (1 - phi) * feromonas[indice] + phi * feromona_inicial

        self.costo = np.sum(self.solucion)
```

Ahora actualizamos todas las feromonas para darle prioridad a las feromonas de la mejor solución de esa iteración.

```
def actualizar_feromonas_global(mejor_solucion, mejor_costo):
    global feromonas
    feromonas *= (1 - evaporacion)
```

```
    for i, antena in enumerate(mejor_solucion):
        if antena == 1:
            feromonas[i] += 1 / mejor_costo
```

En la función colonia de hormigas es donde juntaremos todo los métodos.

```
def colonia_de_hormigas():
    mejor_costo_global = float('inf')
    mejor_solucion_global = None
    mejor_matriz_cubierta = None

    for iteracion in range(n_iteraciones):
        hormigas = [Hormiga() for _ in range(n_hormigas)]

        # Cada hormiga construye una solución
        for hormiga in hormigas:
            hormiga.construir_solucion()

        # Encontrar la mejor solución de esta iteración
        mejor_hormiga = min(hormigas, key=lambda h: h.costo)
        if mejor_hormiga.costo < mejor_costo_global:
            mejor_costo_global = mejor_hormiga.costo
            mejor_solucion_global = mejor_hormiga.solucion
            mejor_matriz_cubierta = mejor_hormiga.matriz_cubierta

        # Actualizar feromonas globalmente
        actualizar_feromonas_global(mejor_solucion_global, mejor_costo_global)

    return mejor_solucion_global, mejor_costo_global, mejor_matriz_cubierta
```

5.2 Camino más corto.

El siguiente código desarrollado en Python tiene como objetivo encontrar el camino más corto de un nodo inicial a un nodo final .

5.2.1 Código Ejemplo 1.

Parámetros de algoritmo

```
tau_inicial = 0.1  # Feromona inicial.
tau = tau_inicial * np.ones((L, L))  # Vector de feromonas.
ro = 0.01  # Factor de evaporación.
nh = 50  # Número de hormigas de la colonia.
N = 50  # Número de iteraciones.
```

Valores de entrada y matriz de visibilidad

```
A = np.array([
    [0, 4, np.inf, np.inf, np.inf, 100],  # Desde el nodo 0
    [np.inf, 0, 7, np.inf, np.inf, np.inf],  # Desde el nodo 1
    [np.inf, 8, 0, 18, 9, np.inf],  # Desde el nodo 2
    [1, np.inf, np.inf, 0, 12, 8],  # Desde el nodo 3
    [np.inf, np.inf, 3, 11, 0, np.inf],  # Desde el nodo 4
    [np.inf, np.inf, 6, np.inf, np.inf, 0]  # Desde el nodo 5
])


L = len(A)  # Dimensión de A.

# Matriz de visibilidad
V = np.zeros((L, L))
for i in range(L):
    for j in range(L):
        if A[i, j] != 0:
            V[i, j] = 1 / A[i, j]
pi = 0  # Punto inicial.
pf = 5  # Punto final.
```

Clase hormiga con un punto inicial y final. Y un camino. La función sig nodo. Elige el siguiente nodo de manera probabilística basándose en la visibilidad y la feromona de ese camino. La función trayectoria repite el sig nodo hasta llegar a su límite o al destino. La función aporfero sirve para actualizar las feromonas.La función costo calcula el costo del camino.

```
# Clase Hormiga
class Hormiga:
    def __init__(self, pi, pf):
        self.pi = pi
        self.pf = pf
        self.Camino = [pi]

    def sig_nodo(self, n):
        P = V * tau
        P[n, self.Camino[-1]] = 0  # Impide volver al nodo anterior.
        P = P[n, :] / np.sum(P[n, :])  # Vector de probabilidades.
```

```
        indice = np.array(range(L))
        c = P > 0
        P = P[c]
        indice = indice[c]

        for i in range(1, len(P)):
            P[i] = P[i] + P[i - 1]

        u = random.random()
        if 0 <= u <= P[0]:
            y = indice[0]
        else:
            for i in range(1, len(P)):
                if P[i - 1] < u <= P[i]:
                    y = indice[i]

        self.Camino.append(y)

    def trayectoria(self):
        run = 0
        while run < L - 1:
            self.sig_nodo(self.Camino[-1])
            if self.Camino[-1] == self.pf:
                break
            run += 1

    def apor_fero(self, i, j):
        C = self.Camino
        y = 0
        for t in range(len(C) - 1):
            if (i, j) == (C[t], C[t + 1]):
                y = 1
        return y

    def costo(self):
        C = self.Camino
        if C[-1] != self.pf:
            return np.inf
        return sum(A[C[t], C[t + 1]] for t in range(len(C) - 1))
```

En el main inicializamos nuestra colonia y en cada iteración y actualización las feromonas de manera minimizamos los casos con más costo. En cada iteración guardamos el Mejor Caso

```
def main():
```

```
global tau
d = 0
while d < N:
    # Construimos la colonia
    Colonia = [Hormiga(pi, pf) for _ in range(nh)]

    # Calculamos la trayectoria de cada hormiga
    for hormiga in Colonia:
        hormiga.trayectoria()

    # Calculamos el costo de cada hormiga
    for hormiga in Colonia:
        hormiga.costo()

    # Actualización de la feromona
    for i in range(L):
        for j in range(L):
            for hormiga in Colonia:
                tau[i, j] = (1 - ro) * tau[i, j] + hormiga.apor_fero(i, j) * 1 / (hormiga.costo() ** 4)

    # Cálculo del menor costo en esta iteración
    M_C = [hormiga.costo() for hormiga in Colonia]
    mejora.append(np.min(M_C))

    d += 1

# Cálculos finales y gráficos
Costos = [hormiga.costo() for hormiga in Colonia]
indice = np.argsort(Costos)
Mejor_Costo = Costos[indice[0]]
Mejor_Camino = np.array(Colonia[indice[0]].Camino) + 1

print("El mejor individuo es:", Mejor_Camino)
print("Su costo es:", Mejor_Costo)
```

5.3 Planificación de Transporte de Mercancía.

Esta es una versión simplificada. Tenemos 3 centros de distribución y tres tiendas. Cada centro de distribución puede tener una oferta o cantidad disponible para entregar. Cada tienda tiene una cantidad demandada. Hay un costo por cada elemento para entregar de un centro a una tienda. El objetivo es minimizar ese costo buscando satisfacer la oferta y la demanda.

5.3.1 Código Ejemplo 2.

Parámetros del algoritmo de hormigas

```
# Parámetros del algoritmo de colonia de hormigas
tau_inicial = 0.1  # Feromona inicial
alpha = 1  # Peso de la feromona
beta = 2  # Peso de la heurística
evaporacion = 0.5  # Tasa de evaporación
n_hormigas = 10  # Número de hormigas
n_iteraciones = 100  # Número de iteraciones
```

Entrada

```
# Parámetros del problema
origenes = ["A", "B", "C"]  # Orígenes
destinos = ["X", "Y", "Z"]  # Destinos
costos = np.array([  # Matriz de costos de transporte
    #X #Y #Z
    [4, 3, 2],#A
    [5, 8, 6],#B
    [7, 4, 3]#C
])
demandas = [30, 40, 50]  # Demanda en los destinos
ofertas = [40, 50, 30]   # Oferta en los orígenes
# Inicialización de las feromonas
tau = tau_inicial * np.ones_like(costos)

# Función heurística: 1/costo (mayor probabilidad para menores costos)
heuristica = 1 / costos
```

Clase hormiga.

Los parámetros son la oferta y demanda restante. Junto con el costo y la asignación parte de la solución encontrada por la hormiga.

La función que construye la solución:

Inicia calculando las probabilidades según la feromona y la visibilidad.

Escoge una asignación tienda de manera probabilística. Después escoge la cantidad escogiendo el mínimo entre la demanda y la oferta restante. Y por último actualiza todos los parámetros. Esto lo hará hasta satisfacer la demanda.

```
# Clase Hormiga
class Hormiga:
    def __init__(self):
        self.asignacion = np.zeros_like(costos)
        self.ofertas_restantes = ofertas.copy()
        self.demandas_restantes = demandas.copy()
        self.costo_total = 0

    def construir_solucion(self):
        global heuristica, tau
        while sum(self.demandas_restantes) > 0:
            # Probabilidades para elegir origen-destino
            probabilidades = (tau ** alpha) * (heuristica ** beta)
            probabilidades /= np.sum(probabilidades)

            # Seleccionar un origen-destino basado en probabilidades
            indices_origen, indices_destino = np.unravel_index(
                np.random.choice(probabilidades.size, p=probabilidades.ravel()),
probabilidades.shape
            )
            origen, destino = indices_origen, indices_destino

            # Cantidad a transportar
            cantidad = min(self.ofertas_restantes[origen], self.demandas_restantes[destino])

            # Actualizar la asignación, ofertas y demandas
            self.asignacion[origen, destino] += cantidad
            self.ofertas_restantes[origen] -= cantidad
            self.demandas_restantes[destino] -= cantidad
            self.costo_total += cantidad * costos[origen, destino]
```

Función para actualizar la feromonas

```
def actualizar_feromonas(hormigas):
    global tau
    # Evaporación de feromonas
    tau *= (1 - evaporacion)
    for hormiga in hormigas:
        for i in range(costos.shape[0]):
```

```
            for j in range(costos.shape[1]):
                if hormiga.asignacion[i, j] > 0:
                    tau[i, j] += 1 / hormiga.costo_total  # Incremento proporcional al costo inverso
```

En la función principal todas nuestras hormigas buscarán la solución. Después escogemos y guardamos la iteración al menor costo. En cada iteración actualizaremos las feromonas. Retornaremos los datos relacionados a la mejor hormiga en la ultima iteración.

```
def colonia_de_hormigas():
    mejor_costo_global = float("inf")
    mejor_asignacion_global = None
    costos_mejor_iteracion = []

    for iteracion in range(n_iteraciones):
        hormigas = [Hormiga() for _ in range(n_hormigas)]

        # Cada hormiga construye una solución
        for hormiga in hormigas:
            hormiga.construir_solucion()

        # Buscar la mejor solución de esta iteración
        mejor_hormiga = min(hormigas, key=lambda h: h.costo_total)
        if mejor_hormiga.costo_total < mejor_costo_global:
            mejor_costo_global = mejor_hormiga.costo_total
            mejor_asignacion_global = mejor_hormiga.asignacion

        # Actualizar feromonas
        actualizar_feromonas(hormigas)

        # Guardar el mejor costo de esta iteración
        costos_mejor_iteracion.append(mejor_costo_global)

    return mejor_asignacion_global, mejor_costo_global, costos_mejor_iteracion
```

5.3 Problema del repartidor.

Digamos que eres un repartidor. Que busca ir de nodo en nodo para terminar en el nodo de inicio recorriendo todos los nodos en el proceso.

5.3.1 Código Ejemplo 4.

Empezamos definiendo todos los parámetros del algoritmo de hormigas.

```
n_hormigas = 10
n_iteraciones = 1000
alpha = 1  # Intensidad de la feromona
beta = 2   # Influencia de la distancia
rho = 0.5  # Evaporación de feromonas
Q = 100    # Constante de feromonas

# Inicialización de la feromona
feromona = {arista: 1.0 for arista in distancias}
```

Definimos la entrada del problema

```
nodos = ['A', 'B', 'C', 'D', 'E']  # Los 5 nodos
distancias = {  # Distancias entre los nodos (distancia entre A y B es 10, etc.)
    ('A', 'B'): 20, ('B', 'A'): 20,
    ('A', 'C'): 24, ('C', 'A'): 24,
    ('A', 'D'): 17, ('D', 'A'): 17,
    ('A', 'E'): 14, ('E', 'A'): 14,
    ('B', 'C'): 13, ('C', 'B'): 13,
    ('B', 'D'): 9,  ('D', 'B'): 9,
    ('B', 'E'): 15, ('E', 'B'): 15,
    ('C', 'D'): 12, ('D', 'C'): 12,
    ('C', 'E'): 22, ('E', 'C'): 22,
    ('D', 'E'): 18, ('E', 'D'): 18,
    ('A', 'A'): 0   # Distancia de 'A' a 'A' es 0
}
```

Calculamos la distancia de la ruta incluyendo la vuelta al nodo inicial

```
def calcular_distancia(ruta):
    distancia_total = 0
    for i in range(len(ruta) - 1):
        distancia_total += distancias[(ruta[i], ruta[i+1])]
    distancia_total += distancias[(ruta[-1], ruta[0])]  # Vuelve al nodo inicial
    return distancia_total
```

Primero filtramos los nodos sin visitar. Después calculamos las probabilidades de cada nodo según la feromona y la distancia

```
def seleccionar_siguiente_nodo(nodo_actual, nodos_visitados, feromona, distancias):
    candidatos = [nodo for nodo in nodos if nodo not in nodos_visitados]
    probabilidades = []

    # Calcular las probabilidades para cada candidato
    for candidato in candidatos:
        arista = (nodo_actual, candidato) if (nodo_actual, candidato) in distancias else
(candidato, nodo_actual)
        probabilidad = (feromona[arista] ** alpha) * ((1.0 / distancias[arista]) ** beta)
        probabilidades.append(probabilidad)

    # Normalización
    suma_probabilidades = sum(probabilidades)
    probabilidades = [p / suma_probabilidades for p in probabilidades]

    # Selección del siguiente nodo por ruleta
    return random.choices(candidatos, weights=probabilidades, k=1)[0]
```

Reduce todas las feromonas. Y las rutas de menor costo son reforzadas con mas feromonas.

```
def actualizar_feromonas(feromona, rutas, distancias, rho, Q):
    # Evaporación de feromonas
    for arista in feromona:
        feromona[arista] *= (1 - rho)

    # Depósito de nuevas feromonas
    for ruta in rutas:
        distancia_total = calcular_distancia(ruta)
        for i in range(len(ruta) - 1):
            arista = (ruta[i], ruta[i+1]) if (ruta[i], ruta[i+1]) in distancias else (ruta[i+1], ruta[i])
            feromona[arista] += Q / distancia_total
        arista = (ruta[-1], ruta[0]) if (ruta[-1], ruta[0]) in distancias else (ruta[0], ruta[-1])
        feromona[arista] += Q / distancia_total
```

De manera parecida a los demás ejemplos Se itera las veces elegidas y por el número de hormigas. Guardamos los resultados en vectores .Actualizamos las feromonas . Comparamos y escogemos el camino con el menor costo y lo retornamos en la última iteración.

```
def resolver_tsp(nodos, distancias, n_hormigas, n_iteraciones, alpha, beta, rho, Q):
    mejor_ruta = None
    mejor_distancia = float('inf')

    for _ in range(n_iteraciones):
        rutas = []
        for _ in range(n_hormigas):
            ruta = []
            nodo_actual = 'A'  # Aseguramos que todas las hormigas comiencen en "A"
            ruta.append(nodo_actual)
            nodos_visitados = set([nodo_actual])
            while len(ruta) < len(nodos):
                siguiente_nodo = seleccionar_siguiente_nodo(nodo_actual, nodos_visitados,
feromona, distancias)
                ruta.append(siguiente_nodo)
                nodos_visitados.add(siguiente_nodo)
                nodo_actual = siguiente_nodo
            ruta.append('A')  # Aseguramos que la ruta termine en "A"
            rutas.append(ruta)

        # Actualizar feromonas
        actualizar_feromonas(feromona, rutas, distancias, rho, Q)

        # Encontrar la mejor ruta de esta iteración
        for ruta in rutas:
            distancia = calcular_distancia(ruta)
            if distancia < mejor_distancia:
                mejor_ruta = ruta
                mejor_distancia = distancia

    return mejor_ruta, mejor_distancia
```

5.4 Repositorio.

Todos los códigos de los ejemplos se encuentran en el repositorio junto con los ejecutable para casos personalizados.

6. Resultado y Gráficos

6.1. Planificación de Celdas en Redes

6.1.1. Problema

El programa se probó con la siguiente matriz

Figura 10.

Matriz de ejemplo.

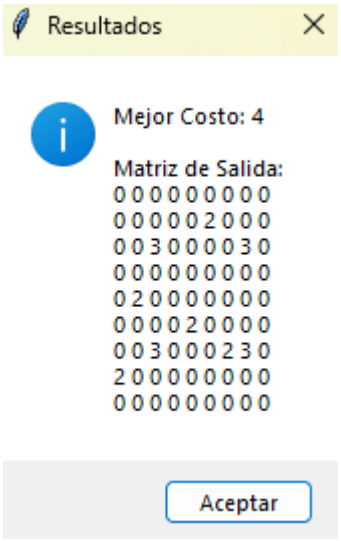
NOTA **:** En la figura se representa una matriz utilizada para el problema de celda de telecomunicacion.

6.1.2. Gráfico

El resultado para el problema fue el siguiente.

Figura 11.

Resultado de ejecución.

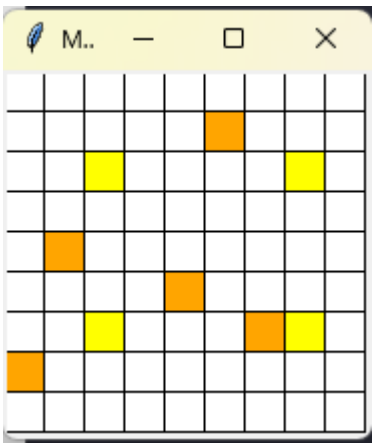


NOTA **:** Resultado de la ejecución con los datos del problema propuesto.

Para una mejor visualización se puede realizar en un mapa de bits

Figura 12.

Mapa de bits..



NOTA **:** Resultado de la ejecución representado en un mapa de bits.

El color amarillo son las ubicaciones candidatas elegidas y las naranjas las ignoradas.

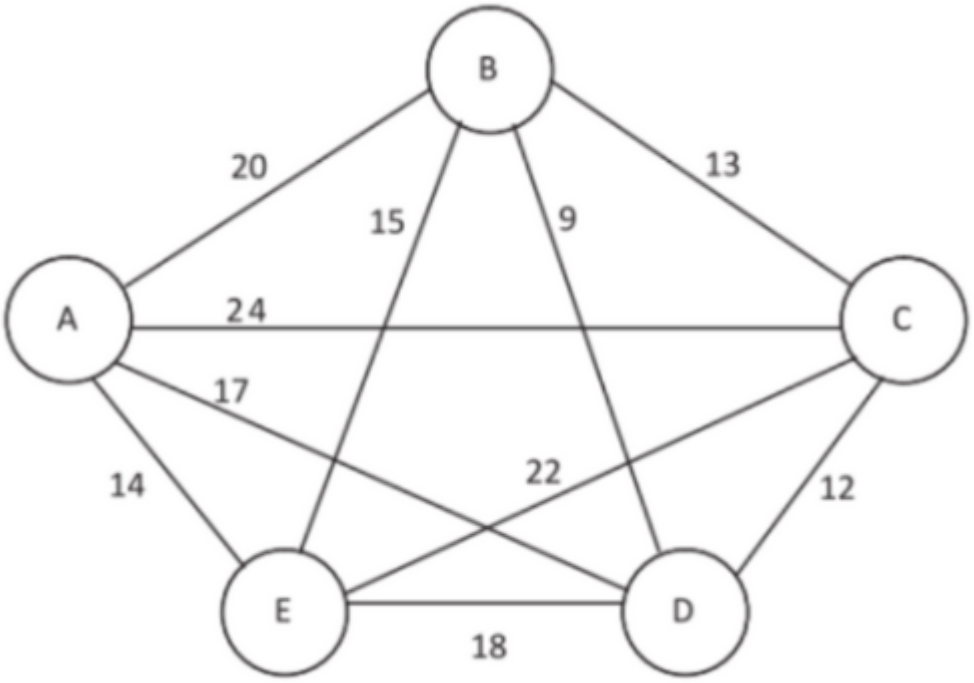
6.2. Problema del repartidor.

6.2.1. Problema

El programa se probó con el siguiente problema.

Figura 13.

Mapa de Nodo Ponderado.

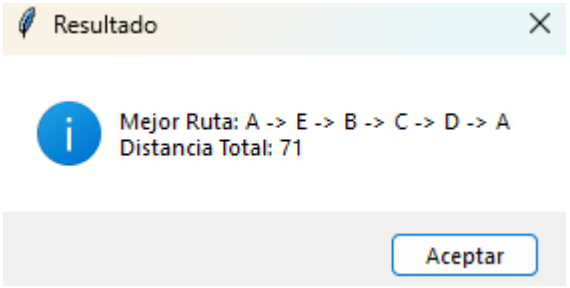


NOTA **:** Mapa de nodos utilizado en una práctica del curso.

Donde el punto de inicio es el nodo “A”.El resultado fue el siguente.

Figura 14.

Resultado de la ejecución.



NOTA **:** Resultado con el ejercicio propuesto.

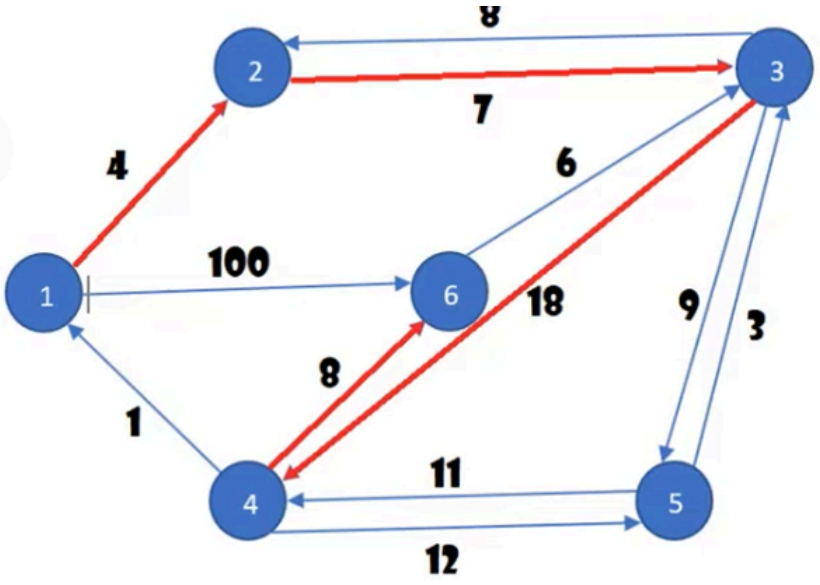
6.3. Camino más corto

6.3.1. Problema

Para probar el programa se probó el siguiente grafo ponderado .

Figura 14.

Nodo ponderado.



NOTA **:** El camino con menor peso está marcado con rojo.

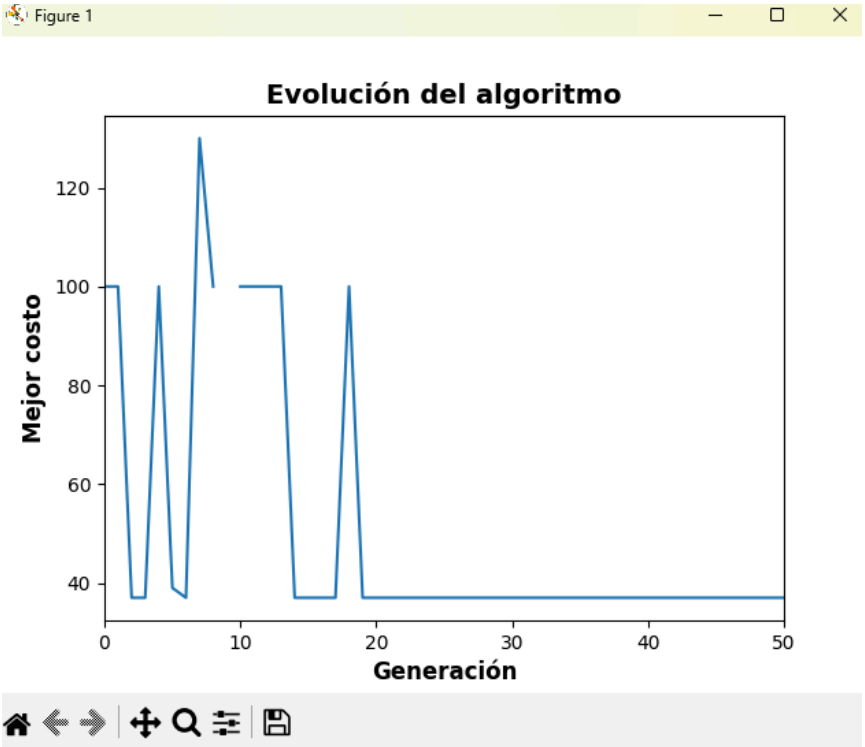
El resultado varió entre 37 y 39. Esto fue porque en el nodo 3 a las hormigas por la visibilidad les costaba ver el 4 como mejor opción. Esto se podría solucionar usando el mejor resultado de todas las iteraciones y no de la última.

6.3.2. Gráfico

Un gráfico interesante es el de mejor resultado en cada iteración . Se logró matplotlib de python.

Figura 14.

Gráfico que compara el Mejor Costo en cada iteración.



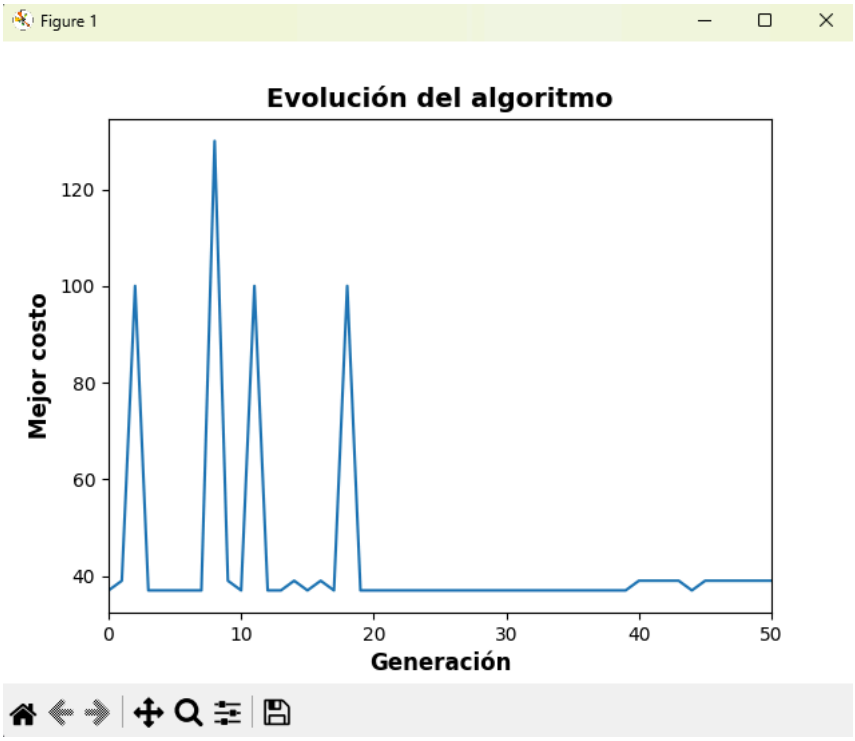
NOTA **:** Elaboración Propia con una librería de Python.

Figura 15.

Gráfico que compara el Mejor Costo en cada iteración.

Figura 16.

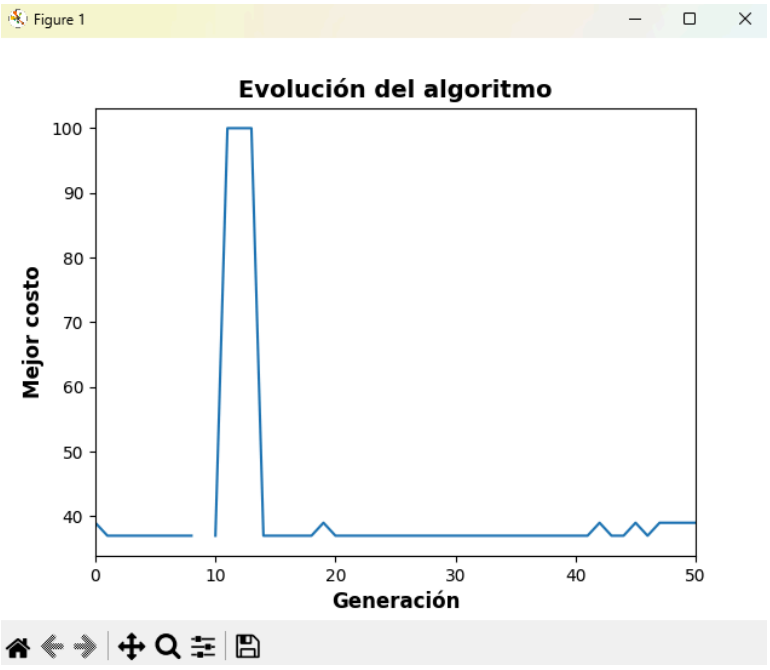
Gráfico que compara el Mejor Costo en cada iteración.



NOTA **:** Elaboración Propia con una librería de Python.

Figura 17.

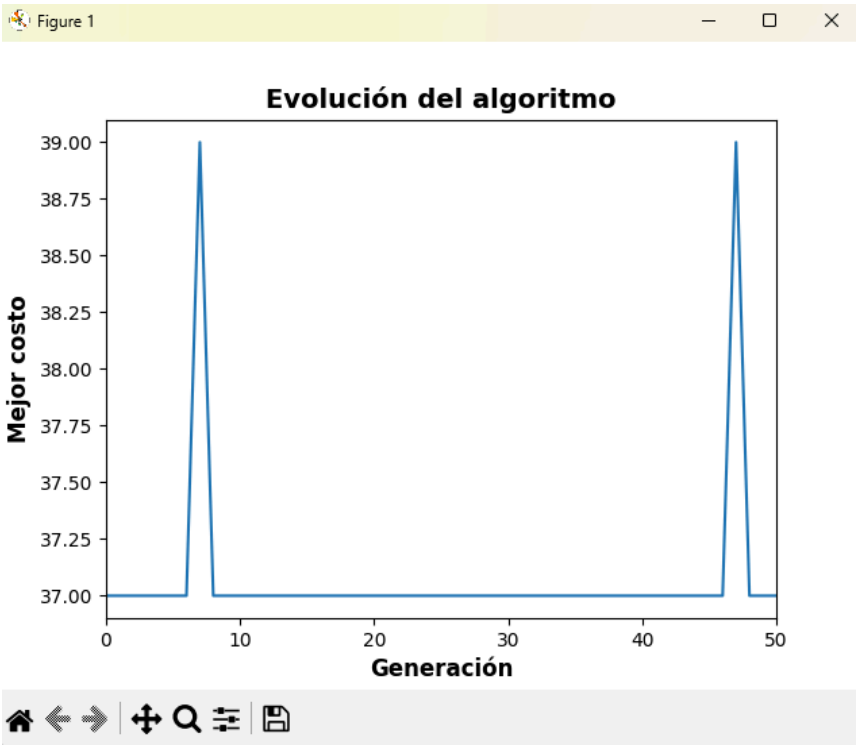
Gráfico que compara el Mejor Costo en cada iteración.



NOTA **:** Elaboración Propia con una librería de Python

Figura 18.

Gráfico que compara el Mejor Costo en cada iteración.



NOTA **:** Elaboración Propia con una librería de Python

Hay casos en los cuales la respuesta óptima en cada iteración varía poco o mucho. Esto se debe a la característica probabilística de este método.

7. Conclusión

Este proyecto nos ha ayudado a entender el concepto de los algoritmos basado en la colonia de hormigas. Comprendimos cómo aplicar el comportamiento de las hormigas al momento de buscar caminos en la resolución de problemas de optimización tales como la optimización de rutas. Optimización de recursos,etc. Además de aprender el manejo de tecnologías tales como python y tkinter.

Este algoritmo ha demostrado ser eficaz en lo que a resolución de problemas de optimización se refiere. A pesar de no dar con certeza siempre la respuesta más óptima es útil para problemas que crecen fácilmente en complejidad. Además gracias al trabajo sabemos para qué tipos de problemas puede ser aplicado el algoritmo y gracias al concepto general de su funcionamiento podemos adaptarlo a estos.

Los algoritmos basados en comportamientos de la naturaleza como estos a pesar de no dar siempre la mejor solución pueden ser útiles en problemas que de otra manera serían muy complejos.

8. Bibliografía

Algarín, C. A. R. (2010). Optimización por colonia de hormigas: aplicaciones y tendencias. *Ingeniería solidaria*, *6*(10-11), 83-89. https://revistas.ucc.edu.co/index.php/in/article/view/454

Barcos, L., Alvarez, M., Rodriguez, V., & Robuste, F. (2015). Algoritmo basado en la optimización mediante colonias de hormigas para la resolución del problema del transporte de carga desde varios orígenes a varios destinos. *ResearchGate.* https://www.researchgate.net/publication/267418426_Algoritmo_basado_en_la_optimizacion_mediante_colonias_de_hormigas_para_la_resolucion_del_problema_del_transporte_de_carga_desde_varios_origenes_a_varios_destinos

Chacón Pérez, O. de J., Aguilar Padilla, J. V., Ríos Tercero, O. A., Basave Torres, R. I., & Cruz Gordillo, R. (2022). *Algoritmo de optimización colonia de hormigas para la generación automática de horarios*. *Revista Tecnología Digital*, 9(2), 1-12. https://www.revistatecnologiadigital.com/pdf/09_02_001_algoritmo_colonia_hormigas_generacion_horarios.pdf

Cruz, J., Pineda, F., & Banca, L. (2015). Optimización de la transmisión de datos en una red de routers mediante el algoritmo de la colonia de hormigas. *Dialnet.* 17(2), 249-256. https://dialnet.unirioja.es/servlet/articulo?codigo=5169795

De Quevedo, U. T. E. (n.d.). *Meta heurística basada en colonia de hormigas en n etapas para problemas de alta dimensión.* uteq.edu.ec. https://www.uteq.edu.ec/es/investigacion/articulo/1256

Lozano, Á. G. G., Cascante, G. E. M., Alulema, J. C. M., Flores, A. N. G., & Martínez, J. C. C. (2018). *Efecto de la aplicación del algoritmo de colonia de hormigas en un servicio logístico*. Dialnet. https://dialnet.unirioja.es/servlet/articulo?codigo=6750182

Ponce, J., Pandilla, F., Pandilla, A., & Meza, M. (2006) ACHPM: Algoritmo de Optimización con Colonia de Hormigas para el Problema de la Mochila. *Departamento de Sistemas Electrónicos, Universidad Autónoma de Aguascalientes.* 3(2), 54-57. https://www.iiisci.org/journal/pdv/risci/pdfs/C821AF.pdf

9. Anexos

Donald1526. (n.d.). *IO-TrabajoGrupal*. [Repositorio GitHub]. https://github.com/Donald1526/IO-TrabajoGrupal