## Problema del Viajante Resuelto con el Bioalgoritmo de las Luciérnagas

## Introducción:

El Problema del Viajante (TSP) es uno de los problemas más estudiados en la teoría de la optimización combinatoria. En este problema, el objetivo es encontrar la ruta más corta posible que permita a un viajante visitar un conjunto de ciudades exactamente una vez y regresar al punto de origen. Este problema es NP-difícil, lo que significa que no existe un algoritmo eficiente conocido para resolverlo en tiempo polinómico para todas las instancias.

El TSP tiene aplicaciones prácticas en logística, planificación de rutas y diseño de circuitos, entre otros campos. Su dificultad proviene del gran número de posibles rutas, que crece exponencialmente con el número de ciudades. Por lo tanto, encontrar una solución óptima es un desafío significativo.

## ¿Por qué este problema y no otro?

El TSP ha sido elegido por varias razones:

* Relevancia Práctica: La optimización de rutas es crucial en áreas como la logística y la planificación de rutas de transporte. Resolver el TSP puede mejorar la eficiencia en la gestión de recursos y reducir costos operativos.
* Desafío Computacional: El TSP es un problema bien conocido en la teoría de la complejidad computacional. Su naturaleza NP-difícil lo convierte en un banco de pruebas ideal para nuevas técnicas de optimización, como el algoritmo de las luciérnagas.
* Aplicación del Algoritmo: El algoritmo de las luciérnagas es adecuado para problemas de optimización complejos y multimodales como el TSP. Este algoritmo puede explorar eficazmente el espacio de soluciones y evitar quedar atrapado en óptimos locales.

## Herramientas a Utilizar

* Algoritmo de las Luciérnagas: Este algoritmo bioinspirado se basa en el comportamiento de las luciérnagas para encontrar soluciones óptimas en problemas de optimización. Utilizaremos este algoritmo para explorar el espacio de soluciones del TSP y encontrar rutas óptimas o cercanas al óptimo.
* Lenguaje de Programación: Python será el lenguaje de programación utilizados para implementar el algoritmo. Este lenguaje ofrece bibliotecas y herramientas adecuadas para el desarrollo y la optimización de algoritmos.
* Entorno de Desarrollo: Se utilizarán entornos de desarrollo como Jupyter Notebook para Python para implementar y probar el algoritmo.

## Datos a Usar

* Dataset del TSP: Utilizaremos un dataset que contiene un conjunto de ciudades y las distancias entre ellas. Este dataset será proporcionado y subido al repositorio. Los datos estarán en un formato estándar que permitirá su fácil integración con el algoritmo de las luciérnagas.
* Formato del Dataset: El dataset incluirá coordenadas de ciudades o una matriz de distancias. Esto permitirá calcular la ruta más corta posible utilizando el algoritmo de las luciérnagas.

## Modelos Matemáticos

1. **Modelo del Problema del Viajante**

El Problema del Viajante (TSP) se puede formular matemáticamente como sigue:

Datos:

* Un conjunto de ciudades,
* Una matriz de distancias donde representan la distancia entra la distancia entre la ciudad y la ciudad

Objetivo:

* Encontrar un ciclo hamiltoniano de mínima longitud en el grafo completo, que visita cada ciudad exactamente una vez y regresa al punto de origen.

Modelo matemático: El objetivo es minimizar la función de costo

Minimizar

donde:

* es una variable binaria que es 1 si el viajante viaja directamente de la ciudad a la ciudad y 0 en el caso contrario.

Sujeto a las siguientes restricciones:

* Cada ciudad es visitada exactamente una vez
* Eliminación de subciclos (para evitar ciclos no deseados)

1. **Modelo de optimización de algoritmo de las luciérnagas**

El algoritmo de las luciérnagas no tiene una formulación matemática exacta como el TSP, pero su comportamiento puede modelarse usando la teoría de optimización. En el contexto del TSP, el algoritmo se basa en la siguiente formulación:

* Función Objetivo: El objetivo es minimizar la distancia total recorrida. Esto se traduce en maximizar la "brillantez" de las soluciones, donde la brillantez está inversamente relacionada con la distancia total.
* Movimiento de las Luciérnagas: Cada luciérnaga se mueve hacia otras luciérnagas con mayor brillantez. La actualización de la posición de una luciérnaga 𝑖 hacia una luciérnaga 𝑗 con mejor solución se define como:

Donde:

* es la posición de la luciérnaga en el tiempo .
* es la intensidad de atracción.
* es el coeficiente de absorción de luz.
* es la distancia entre las luciérnagas y .
* es el factor de aleatoriedad.
* es un término aleatorio.

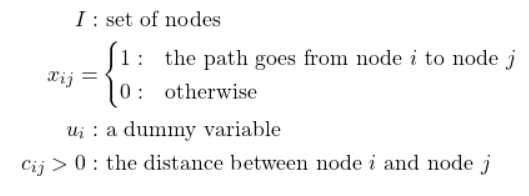
## Soluciones propuestas:

* Programación Lineal Entera:

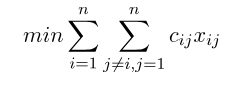
Como mencionamos, existen dos formulaciones principales para el TSP, el propuesto Miller, Tucker y Zemlin (MTZ) y el Dantzig, Fulkerson y Johnson (DFJ). Aunque la formulación DFJ es más fuerte, la formulación MTZ puede ser útil

**Formulación MTZ**

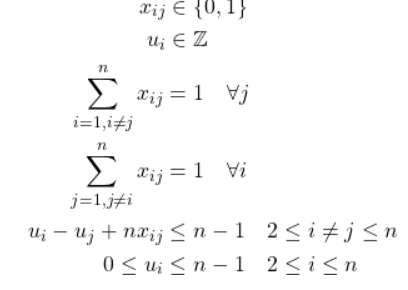
La formulación matemática propuesta por Miller, Tucker y Zemlin es la siguiente:



Usando este conjunto, las dos variables y el parámetro, podemos formular el problema como:



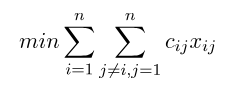
Sujeto a:



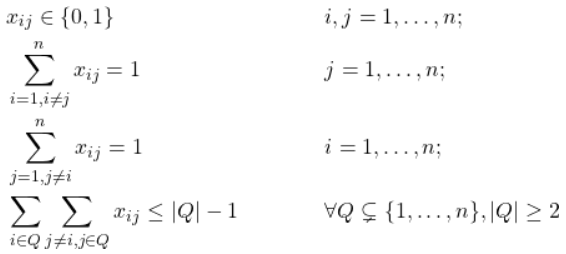
La primera y la segunda ecuación refuerzan el tipo de las diferentes variables, la tercera y la cuarta ecuaciones aseguran que cada nodo sea alcanzado y abandonado solo una vez, mientras que las dos últimas ecuaciones imponen que solo una ruta cruce todos los nodos.

**Formulación DFJ**

En la formulación propuesta por Dantzig, Fulkerson y Johnson partimos de la misma variable x\_ {ij} y parámetro c\_ {ij}> 0 para que la formulación se pueda completar con:

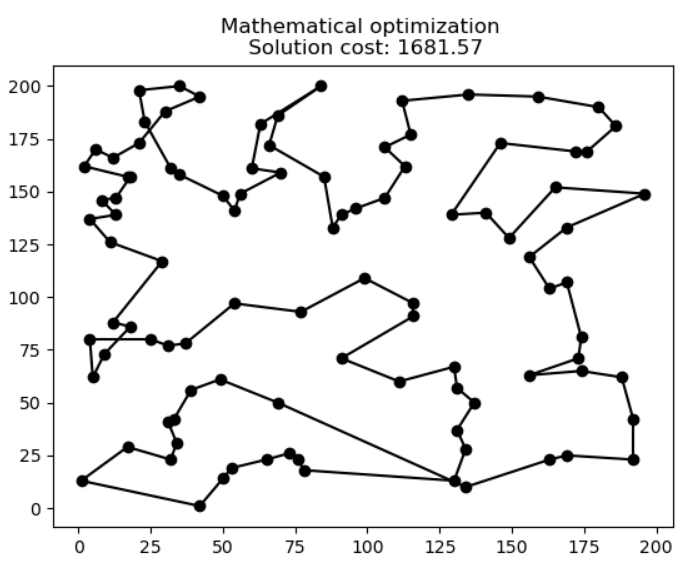


Sujeto a:



Las tres primeras ecuaciones son las mismas que en la formulación anterior, y la nueva restricción (la última) asegura que no haya sub-recorridos, por lo que la solución devuelta es un recorrido único y no la combinación de recorridos más pequeños. Debido a que esto conduce a un número exponencial de posibles restricciones, en la práctica se resuelve con una generación de columnas retrasada.

En el siguiente gráfico podemos ver la solución óptima para un problema de 100 nodos.



* **Optimización por Colonia de Hormigas**

También conocido como **Ant Colony System (ACS)**o**Ant System (AS)**, es una técnica probabilística o heurística utilizada para resolver problemas que se pueden reducir a encontrar buenos caminos sobre grafos. Este método fue propuesto inicialmente por Marco Dorigo en 1992 en su tesis doctoral

* **Algoritmos Genéticos**

Los Algoritmos Genéticos (GA) son heurísticos inspirados en el proceso de evolución de los seres vivos. Cada solución es un cromosoma compuesto por genes que representan los diferentes valores de las variables de la solución.

Comenzando con una población inicial, podemos hacer que las soluciones “evolucionen” en iteraciones. El proceso consiste en seleccionar la mejor mitad de la población de la que elegimos al azar pares de cromosomas (soluciones) para el apareamiento. En estos apareamientos intercambiamos genes de ambas soluciones para generar nuevas soluciones (hijos).

Después del proceso de apareamiento, aplicamos mutaciones al azar en los hijos resultantes para explorar más a fondo el espacio de la solución.

Finalmente seleccionamos los mejores 50 individuos de la población (el inicial más los hijos) y continuamos el proceso hasta que convergemos en una solución.

## Referencias

González-Santander, Guillermo. "Tres métodos diferentes para resolver el problema del viajante." Baobab Soluciones, 1 de octubre de 2020. <https://baobabsoluciones.es/blog/2020/10/01/problema-del-viajante/>.

"Problema del viajante." *Wikipedia, la enciclopedia libre*, 8 de septiembre de 2024. <https://es.wikipedia.org/wiki/Problema_del_viajante>.

Vázquez, Javier. "Revisión de los Algoritmos Bioinspirados." *Universidad Nacional Autónoma de México*, julio de 2014. <https://www.fis.unam.mx/~javazquez/files/Met_num/Algoritmosbioinspiradosfinal.pdf>.

Wikipedia. (n.d.). *Algoritmo firefly*. Wikipedia, la enciclopedia libre. Recuperado el 8 de septiembre de 2024, de <https://es.wikipedia.org/wiki/Algoritmo_firefly>

Visión electrónica. (2021). *Firefly algorithm for facility layout problem optimization*. *15*(2), 218–225. <https://doi.org/10.14483/22484728.17474>