uc3m | Universidad Carlos III de Madrid

MÁSTER UNIVERSITARIO EN CIENCIA Y TECNOLOGÍA INFORMÁTICA

Inteligencia Artificial de Inspiración Biológica

Resolución del problema del viajero con el algoritmo de optimización de colonia de abejas

Nombre y apellidos de los alumnos:

Darío Muñoz Muñoz (100405982)
100405982@alumnos.uc3m.es

Fecha de entrega: 07/01/2024

Índice

Índice de Figuras

Índice de Tablas

Introducción

El Problema del Viajero (Traveling Salesman Problem, TSP) es un desafío fundamental en la optimización combinatoria y la teoría de algoritmos. Consiste en encontrar la ruta más corta que permita a un viajero visitar todas las ciudades deseadas una sola vez y regresar al punto de partida. A pesar de su simplicidad aparente, su complejidad computacional ha atraído la atención de la comunidad científica durante décadas debido a su naturaleza NP-difícil y su aplicabilidad en diversas áreas como la logística, la planificación de rutas y la optimización de recursos [1].

En este trabajo, se aborda el desafío del Problema del Viajero utilizando una técnica de optimización inspirada en la naturaleza: el Algoritmo de Optimización de Colonia de Abejas (ABC) [2]. Este algoritmo, basado en el comportamiento de las colonias de abejas al buscar fuentes de alimento, ha demostrado su eficacia en la resolución de problemas de optimización combinatoria.

En este documento se presentarán la exploración detallada del TSP y la implementación del algoritmo ABC para resolver este problema. Se abordará la descripción del problema, la técnica a utilizar, el planteamiento teórico para resolver el problema del viajero, así como la descripción del entorno experimental y el análisis de los resultados obtenidos. Finalmente, se ofrecerán conclusiones que reflejen los hallazgos y las implicaciones de este enfoque en la resolución del Problema del Viajero.

El propósito de este estudio es no solo aplicar una técnica innovadora para resolver el TSP, sino también comprender cómo los algoritmos bioinspirados, como el Algoritmo de Optimización de Colonia de Abejas, pueden ofrecer soluciones efectivas a problemas complejos de optimización combinatoria en diversas disciplinas.

Este estudio se puede encontrar en el repositorio: https://github.com/dariomnz/TSP-ABC

Descripción del problema

El Problema del Viajero (TSP) es un desafío clásico en la optimización combinatoria. En su formulación más simple, implica a un viajero que debe visitar un conjunto de ciudades exactamente una vez y regresar al punto de partida, minimizando la distancia total recorrida. Matemáticamente, se puede representar como un grafo completo donde cada ciudad es un nodo y las aristas entre los nodos representan las distancias entre ciudades.

El objetivo es encontrar la ruta más corta que permita al viajero visitar todas las ciudades exactamente una vez y regresar al punto de partida. La complejidad del problema radica en la explosión factorial de las posibles soluciones a medida que aumenta el número de ciudades, lo que lo convierte en un problema NP-difícil.

Las aplicaciones del TSP son diversas, abarcando desde la optimización de rutas de entrega y planificación logística hasta la programación de circuitos en microprocesadores. A pesar de su simplicidad aparente, encontrar la solución óptima para grandes conjuntos de ciudades ha sido un desafío persistente en la investigación de algoritmos.

Para el propósito de nuestro estudio, se empleará una selección de desafíos provenientes de TSPLIB, una biblioteca ampliamente reconocida que alberga instancias estándar del Problema del Viajero (TSP) [3]. Estas instancias son utilizadas como herramientas de referencia en la investigación y desarrollo de algoritmos destinados a abordar el TSP. TSPLIB fue concebida y mantenida por la comunidad académica con el objetivo de ofrecer conjuntos de datos consistentes y comparables, los cuales representan una diversidad de escenarios y complejidades inherentes al TSP. Su utilidad radica en proporcionar una base unificada y confiable para la evaluación y comparación de distintos enfoques al resolver este problema.

Para la representación del problema en términos de modelado, se ha optado por una codificación específica: cada instancia de solución sujeta a evaluación se configura como una secuencia de ciudades dispuestas en una lista ordenada. Esta secuencia dicta el recorrido de la ruta, determinando así el orden en el cual se atraviesan las ciudades. En el contexto combinatorio de este enfoque, al considerar un conjunto de n ciudades, se generan n!

(factorial de n) posibles configuraciones de rutas distintas, lo que refleja la vasta cantidad de opciones de recorrido disponibles para ser evaluadas en el contexto del problema del viajante.

Descripción de la técnica a utilizar

El Algoritmo de Optimización de Colonia de Abejas (ABC) es un algoritmo bioinspirado desarrollado en 2005, que se basa en el comportamiento de las colonias de abejas al buscar fuentes de alimento [2].

El ABC simula el comportamiento de las abejas obreras al recolectar néctar de diferentes fuentes de alimento y comunicar la calidad de las fuentes a otras abejas en la colmena mediante movimientos de danza. En el contexto del TSP, este algoritmo se adapta para encontrar soluciones aproximadas al problema de encontrar la ruta más corta que visite todas las ciudades exactamente una vez y regrese al punto de partida.

El proceso del ABC comienza con la inicialización de una población de soluciones candidatas, que representan posibles rutas para el viajero. Estas soluciones se generan aleatoriamente o se inicializan de alguna manera específica. Luego, las abejas exploradoras buscan fuentes de alimento (soluciones candidatas) mediante una estrategia de búsqueda local y global.

Las abejas emplean dos estrategias principales: el reclutamiento de abejas seguidoras para explorar nuevas soluciones alrededor de las fuentes de alimento encontradas y el abandono de fuentes de alimento menos prometedoras en favor de aquellas más ricas en néctar (soluciones de mejor calidad).

A lo largo de las iteraciones, las soluciones van siendo actualizadas y ajustadas, adaptándose a la calidad de las fuentes de alimento descubiertas por las abejas exploradoras. Este proceso iterativo continúa hasta alcanzar un criterio de parada predefinido, como un número máximo de iteraciones o una convergencia satisfactoria hacia una solución aceptable.

La estrategia propuesta implica inicialmente la generación de un conjunto de soluciones de manera aleatoria, asignando una a cada abeja. El proceso iterativo del algoritmo se basa en la actividad diferenciada de dos tipos de abejas: exploradoras y trabajadoras.

Las abejas exploradoras se encargan de buscar soluciones en todo el espacio de posibles soluciones, lo que se traduce en la generación aleatoria y evaluación de soluciones. Por otro lado, las abejas trabajadoras, durante el proceso iterativo, buscan soluciones cercanas a la suya que puedan ser mejoradas. Al finalizar una iteración, todas las abejas retornan a la colmena. En este momento es cuando se actualizan las soluciones de cada abeja si se cumplen unas condiciones.

Las abejas trabajadoras actualizan su solución si se ha encontrado una mejor en esa iteración o después de un número predefinido de intentos de búsqueda, determinado por la longitud máxima de la ruta.

El proceso de búsqueda de soluciones cercanas por parte de las abejas trabajadoras implica la modificación de dos enlaces entre ciudades en su solución actual. La condición de finalización del algoritmo puede ocurrir tras un número máximo de iteraciones o si transcurren 1000 iteraciones sin encontrar una solución mejor.

Los parámetros ajustables del algoritmo incluyen el número de abejas (en este caso, 50), el límite máximo de iteraciones (establecido en 10000), y la distribución de abejas trabajadoras y exploradoras (75% y 25% respectivamente). Estos valores se pueden modificar para adaptarse a diferentes contextos o necesidades específicas.

Planteamiento teórico de resolución del problema

El TSP se modela en términos de un grafo, donde cada nodo representa una ciudad (i o j) y cada arista entre los nodos denota el camino posible entre dichas ciudades $(\boldsymbol{x_{i,j}})$, con una distancia asociada a cada conexión $(\boldsymbol{d_{i,j}})$. La tarea es encontrar la secuencia óptima de nodos que minimice la distancia total recorrida por el viajante.

A continuación, se modelará el problema utilizando programación lineal entera [4]:

Variables de decisión:

$$x_{i,j} = \begin{cases} 1 & si\ se\ toma\ el\ arco\ (i,j) \\ 0 & en\ caso\ contrario \end{cases}$$

$\boldsymbol{x_{i,j}}$representa si se toma el arco entre las ciudades **i** y **j**.

Toma el valor 1 si el arco **(i, j)** está en la solución, lo que significa que el viajante va de la ciudad **i** a la ciudad **j**.

Toma el valor 0 si el arco **(i, j)** no forma parte de la solución.

Función Objetivo:

$$\sum_{i,j=1}^{n} d_{i,j} x_{i,j}$$

Se calcula la suma ponderada de las distancias entre las ciudades, donde $\boldsymbol{d_{i,j}}$ es la distancia entre las ciudades **i** y **j**. Esta expresión representa la distancia total recorrida en la solución. Siendo **n** el numero de ciudades.

Restricciones:

Visitar todas las ciudades una vez:

$$\sum_{j=1}^{n} x_{i,j} = 1 \quad , i = 1, \dots, n$$

$$\sum_{i=1}^{n} x_{i,j} = 1 \quad , j = 1, \dots, n$$

Las dos restricciones aseguran que cada ciudad sea visitada exactamente una vez. La primera asegura que desde la ciudad **i**, solo hay un arco que sale hacia otra ciudad. La segunda garantiza que hacia la ciudad **j**, solo llegue un arco desde otra ciudad.

Evitar subrutas:

$$u_i - u_j + n\, x_{i,j} \leq (n-1) \quad , 1 < i \neq j \leq n$$
$$x_{i,j} \in \{0,1\}, u_i, u_j \in \{1, \dots, n\}$$

Esta restricción se utiliza para evitar la formación de subrutas. Las variables $\boldsymbol{u_i}$ y $\boldsymbol{u_j}$ son variables auxiliares utilizadas para prevenir subrutas y asegurar que la solución sea un ciclo hamiltoniano, es decir, una ruta que visite cada ciudad exactamente una vez sin subrutas internas.

El modelo final sería el siguiente:

$$
\begin{aligned}
\min \quad & \sum_{i,j=1}^{n} d_{i,j} x_{i,j} \\
s.a \quad & \sum_{j=1}^{n} x_{i,j} = 1 && ,i = 1, \dots, n \\
& \sum_{i=1}^{n} x_{i,j} = 1 && ,j = 1, \dots, n \\
& u_i - u_j + n\, x_{i,j} \leq (n-1) && ,1 < i \neq j \leq n \\
& x_{i,j} \in \{0,1\}, u_i, u_j \in \{1, \dots, n\}
\end{aligned}
$$

Descripción del entorno experimental

En entorno experimental se ha utilizado Python para programar, la librería TSPLIB95 para leer los problemas en los que se realizaran las pruebas, y para la visualización de usaran gráficos generados con la librería plotly

En el entorno experimental para evaluar la eficacia del Algoritmo de Optimización de Colonia de Abejas (ABC) en la resolución del Problema del Viajero (TSP), se ha empleado el lenguaje de programación Python debido a su flexibilidad y robustez.

Para acceder a instancias de problemas del TSP y facilitar la carga de datos, se utilizó la librería TSPLIB95 [5], una librería de Python que permite leer los archivos que se encuentran en TSPLIB [3] que proporciona un conjunto diverso de instancias estándar del TSP. Esta librería permitió la lectura y carga de problemas del TSP en el entorno de desarrollo, brindando una gama de escenarios y complejidades que fueron fundamentales para la evaluación del algoritmo ABC en diversos contextos.

Además, para la visualización de los resultados obtenidos durante las pruebas experimentales, se optó por la utilización de gráficos generados dinámicamente mediante la librería Plotly. Esta elección se basó en la capacidad de Plotly para crear visualizaciones interactivas y de alta calidad, permitiendo una representación efectiva de las soluciones encontradas por el algoritmo ABC en diferentes instancias del TSP.

Análisis de resultados

Para empezar, el análisis de resultados se muestra en la Tabla 1 los resultados obtenidos al ejecutar nuestra implementación del algoritmo ABC en 20 problemas diferentes obtenidos de TSPLIB [3]. Estos están ordenados por el numero de ciudades que tiene cada problema. Para dar una comparación mas gráfica se pude ver en la Figura 1 estos datos.

Nombre	Ciudades	Ciclos	ABC	Óptimo	Error	Tiempo
ulysses16	16	31	6875	6859	0,2332	0,341248
ulysses22	22	24	7013	7013	0,1	0,426259
att48	48	375	11312	10628	6,4358	0,996423
eil51	51	173	448	426	5,1643	0,929984
berlin52	52	317	8100	7542	7,3985	0,955034
st70	70	894	708	675	4,8888	2,044445
eil76	76	605	576	538	7,0631	1,697696
pr76	76	736	114938	108159	6,2676	1,870417
gr96	96	1292	59765	55209	8,2522	3,183353
kroA100	100	1081	23100	21282	8,5424	3,199837
kroC100	100	1542	23077	20749	11,219	3,566749
kroD100	100	1562	23223	21294	9,0588	3,639116
rd100	100	1030	8694	7910	9,9115	3,084292
eil101	101	881	681	629	8,2670	2,675602
lin105	105	1221	15542	14379	8,0881	3,378023
ch130	130	1901	6747	6110	10,425	5,231644
ch150	150	3059	7193	6528	10,186	8,955773
gr202	202	6231	42255	40160	5,2166	21,92509
tsp225	225	4271	4331	3916	10,597	18,22929
a280	280	6551	2984	2579	15,703	34,27986

Tabla 1 Resultados ABC

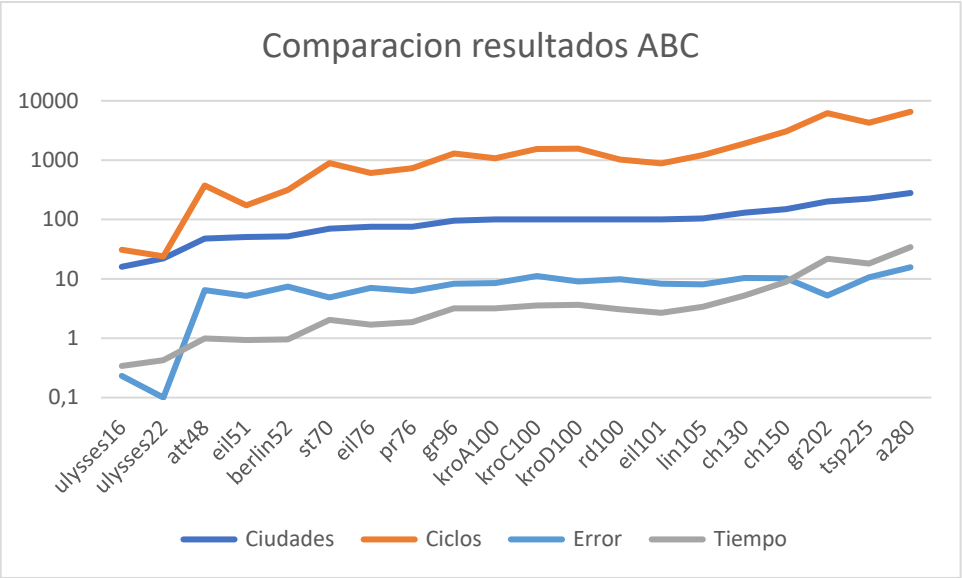

Figura 1 Comparación resultados ABC

Como se pude ver en la figura, tanto el número de ciclos del algoritmo, el error, y el tiempo están correlacionados con el número de ciudades, esto quiere decir que cuanto mas aumenta el numero de ciudades del problema aumentan las demás de forma parecida. Esto se corrobora al realizar una tabla con las correlaciones entre los datos, como se muestra en la Figura 2, que se ve que las ciudades, ciclos y tiempo tienen una correlación cercana a 1.

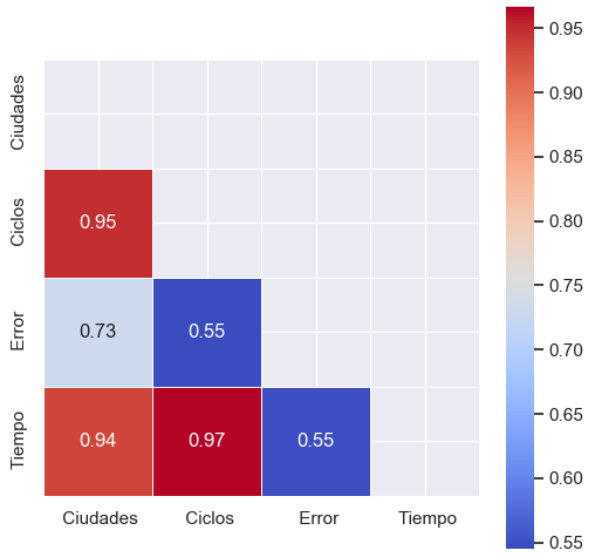

Figura 2 Correlación resultados ABC

También se puede ver en la Figura 1 que el aumento se produce de forma lineal y no exponencial por lo que podemos decir que el algoritmo utilizado tiene un crecimiento casi lineal.

Tras presentar los resultados del algoritmo ABC implementado se mostrará una comparación respecto a otros algoritmos famosos utilizados para la resolución de problemas TSP, estos serán los siguientes (fuente [6]):

- **Nearest Insertion:** Este método inserta un nodo en el conjunto de nodos (vecinos) más cercano, generando la menor cantidad de costo en la longitud total del recorrido. Se busca minimizar el costo de inserción para optimizar la longitud del recorrido.
- **Farthest Insertion:** En este caso, se seleccionan dos ciudades y se conectan para obtener el recorrido de menor costo, luego se busca la ciudad más alejada de este recorrido. Este proceso se repite hasta que todas las ciudades estén conectadas para completar el recorrido.
- **Random Insertion:** Este método selecciona dos ciudades y agrega un nodo de manera aleatoria, donde el orden de los nodos añadidos también es aleatorio, similar al enfoque del vecino más cercano.
- **Nearest Neighbor:** Representa la solución parcial como un camino con un nodo de inicio y final. Se comienza en una ciudad y se selecciona la ciudad más cercana para visitar, repitiendo el proceso hasta que todas las ciudades sean visitadas y la última ciudad se conecte con la primera.
- **2-Opt:** Introduce el método de optimización 2-opt, que consiste en intercambiar los enlaces entre dos pares de nodos consecutivos para mejorar la solución.
- **Minimum Spanning Tree:** Busca minimizar los pesos (longitudes de recorrido) de los bordes del árbol creado.
- **Christofides:** Es un algoritmo heurístico que busca encontrar una solución cercana a la óptima. Involucra la búsqueda de un árbol de expansión mínima, un emparejamiento perfecto mínimo y la creación de un ciclo de Euler en un multigrafo.

- **Cheapest-Link Algorithm:** Selecciona el borde con el menor peso, marcándolo y siguiendo ciertas reglas para evitar circuitos cerrados y múltiples bordes desde un solo nodo.
- **OR-Tools:** Es una herramienta de optimización de Google para problemas combinatorios. Contiene una implementación del problema del enrutamiento de vehículos y utiliza diversas heurísticas y metaheurísticas, incluyendo la Búsqueda Local Guiada.
- **Pointer Network (PN):** Un modelo de red neuronal implementado con aprendizaje supervisado.
- **Bello et al.:** En sus experimentos, utilizaron mini lotes de secuencias, celdas LSTM con unidades ocultas de 128, y optimizaron los modelos con el optimizador Adam. Ajustaron la tasa de aprendizaje según el tamaño del problema, utilizando valores de 10^-3 para problemas más pequeños y 10^-4 para problemas más grandes. Posteriormente, reportaron los resultados obtenidos después de implementar el código.

Como en el desarrollo del algoritmo ha sido usado Python, no se tendrá en cuenta el tiempo de ejecución al comparar ABC con otros algoritmos, ya que Python no es un lenguaje diseñado para ejecutar de la forma más rápida y optima posible. Solo se realiza una comparación de los resultados conseguidos, es decir del valor del error del camino obtenido respecto al camino optimo por los diferentes algoritmos.

Nombre	Or	NETSP	S2V-DQN	ABC	2opt	Furthest	AM	Cheapest	Christ	MST
Eil51	0,23	0,23	3,05	5,16	4,69	33,1	2,11	15,96	23,71	44,13
Eil76	0	0,93	4,83	7,06	9,85	8,36	4,83	12,83	20,07	38,1
Eil101	3,5	0,48	4,77	8,27	11,61	4,77	6,2	10,17	15,74	34,66
St70	1,04	5,48	3,11	4,89	11,56	5,48	22,81	14,96	23,85	27,11
Ch150	1,59	5,24	7	10,19	10,78	10,45	42,43	22,47	17,05	40,98
Ch130	0,61	0,38	2,62	10,43	5,89	6,35	3,58	19,13	20,57	35,52
Berlin52	0	0	0	7,4	3,26	10,14	3,26	19,5	16,97	37,92
Media	**1**	**1,82**	**3,63**	**7,63**	**8,23**	**11,24**	**12,17**	**16,43**	**19,71**	**36,92**

Tabla 2 Comparación error algoritmos

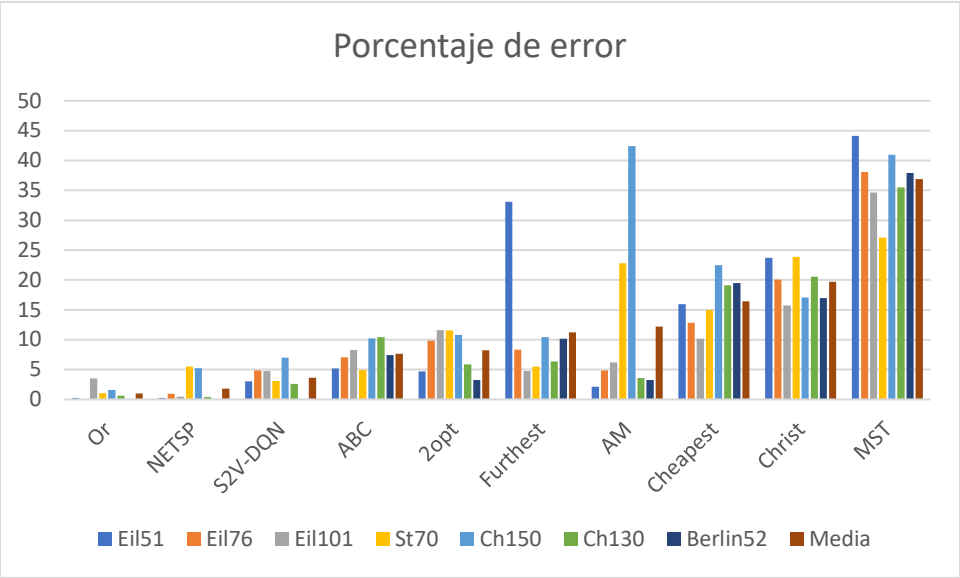

Figura 3 Comparación error algoritmos

El éxito de nuestra implementación del algoritmo de colonia de abejas (ABC) se evidencia en la Tabla 2 y la Figura 3, donde se han ordenado los algoritmos según el error medio obtenido. Nuestra implementación ha logrado una destacada cuarta posición en esta comparativa. Este resultado resalta la eficiencia relativa de nuestro enfoque, mostrando su capacidad para generar soluciones competitivas dentro del contexto de este estudio.

También se ha realizado un análisis de la convergencia de algoritmos iterativos, que constituye un aspecto fundamental en la evaluación de su eficacia y desempeño. Los gráficos de convergencia, representativos de la relación entre el número de iteraciones y la aproximación a la solución deseada, revelan la rapidez con la que un algoritmo converge hacia dicho objetivo. En este sentido, tal y como se observa en la Figura 4 y la Figura 5 ,la convergencia inicial veloz seguida de un progreso más gradual sugiere un comportamiento característico. Esta dinámica implica una etapa inicial de pasos más amplios que conducen a una aproximación rápida de la solución, seguida de una fase donde el algoritmo, encontrándose en proximidad de la solución, opta por reducir la amplitud de sus pasos con el fin de lograr una mayor precisión y refinamiento en el resultado final.

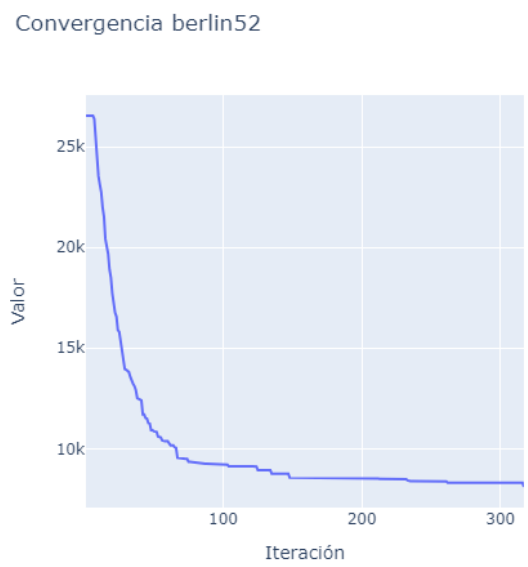

Figura 4 Convergencia berlin52

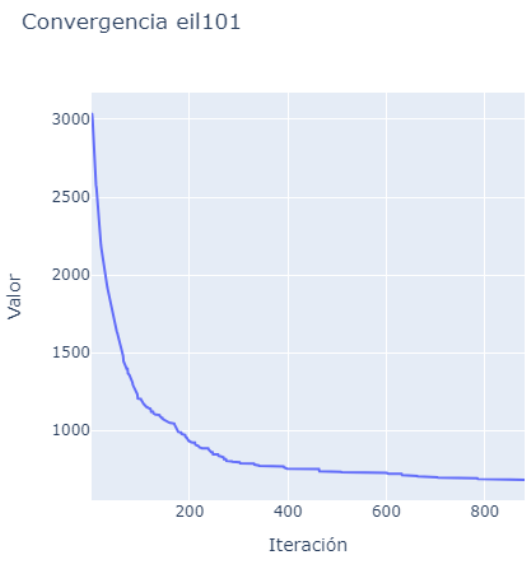

Figura 5 Convergencia eli101

Por último, se mostrarán algunas graficas de las rutas optimas de algunos mapas y las conseguidas por el algoritmo ABC para poder hacer una comparación y ver si en verdad encuentra rutas optimas con un 7,6 % de error de media. Y como se puede ver de la Figura 6 hasta la Figura 12, se comprueba que a pesar de no ser la misma ruta que la ruta optima se consiguen rutas bastante optimas, por eso el 7,6% del error, ya que las rutas de ABC y la optima no son iguales.

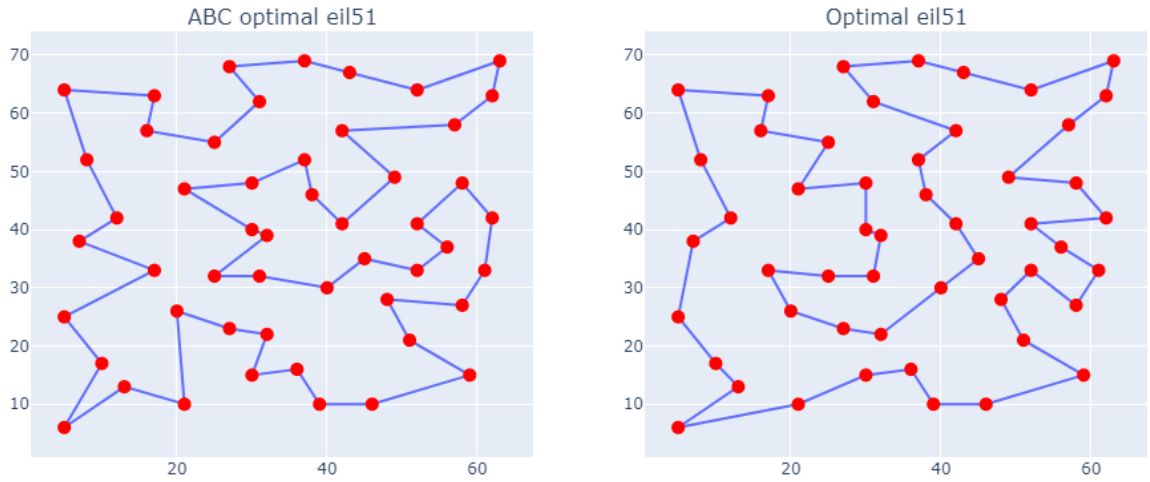

Figura 6 Ruta eil51

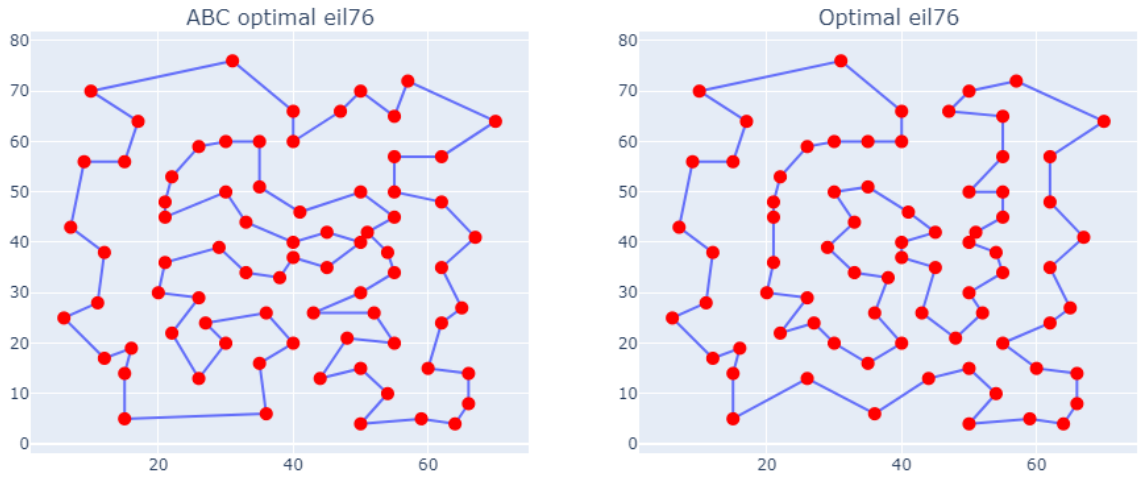

Figura 7 Ruta eil76

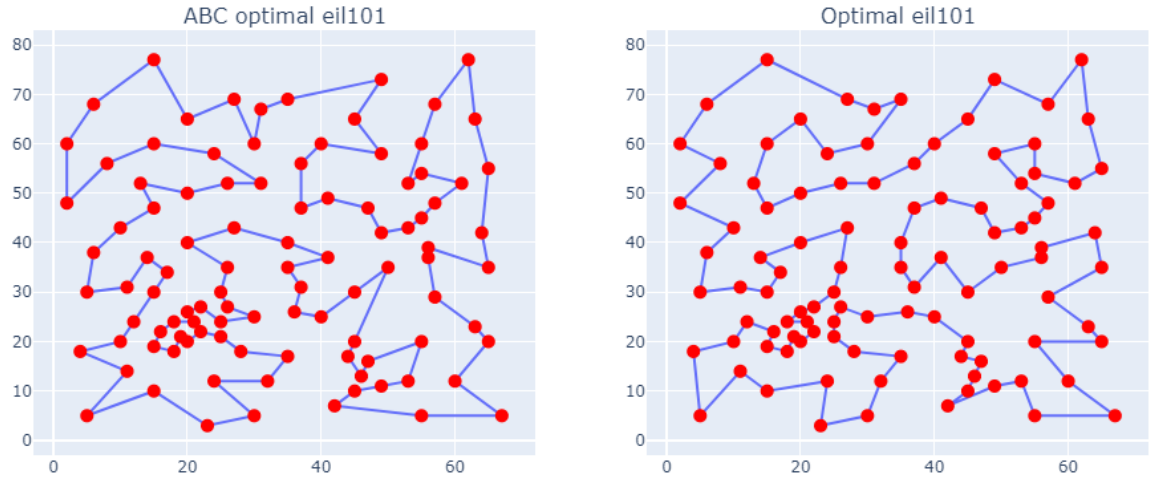

Figura 8 Ruta eil101

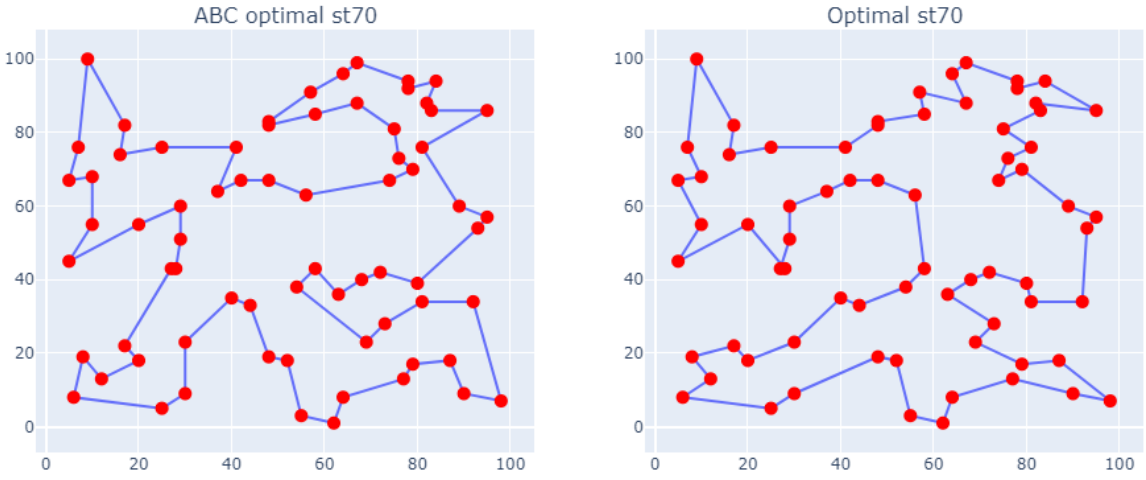

Figura 9 Ruta st70

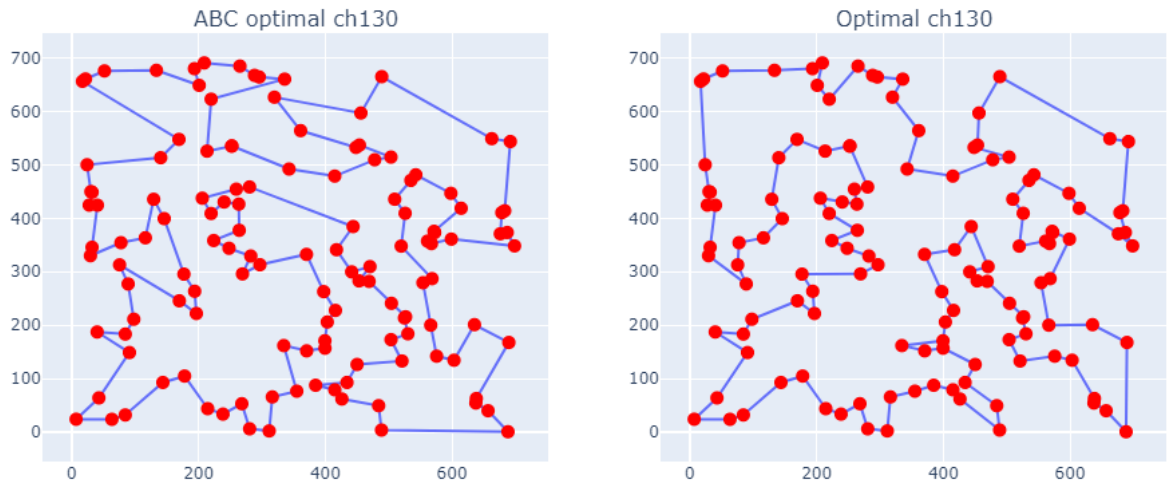

Figura 10 Ruta ch130

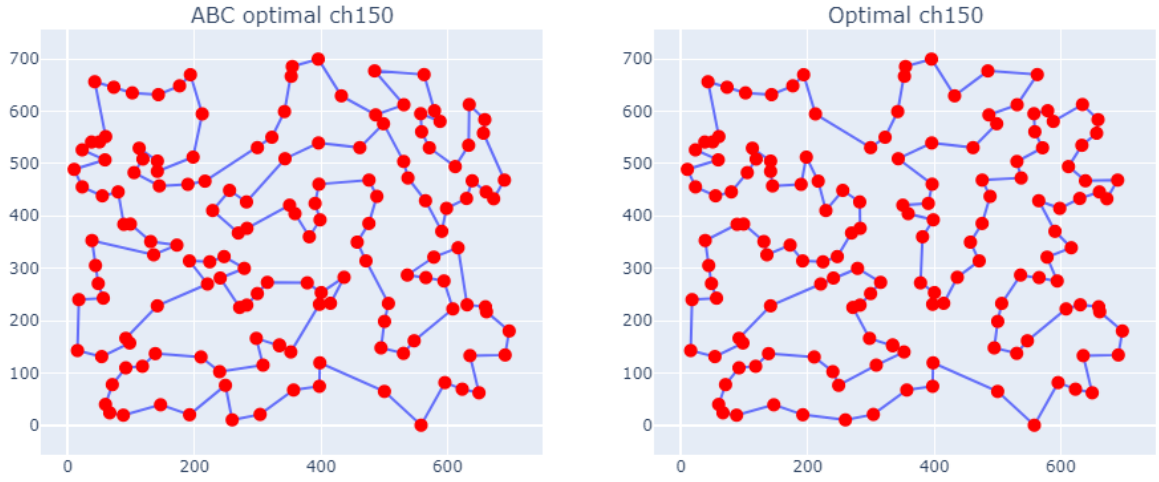

Figura 11 Ruta ch150

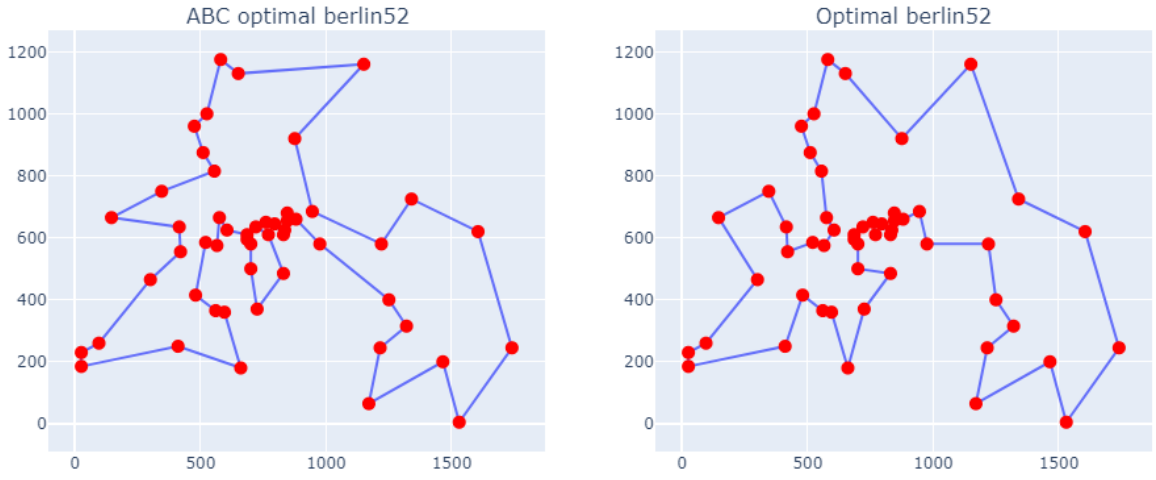

Figura 12 Ruta berlin52

Conclusiones

El algoritmo ABC, inspirado en el comportamiento de las abejas en la búsqueda de fuentes de alimento, ha demostrado su capacidad para aproximar soluciones óptimas al TSP, presentando un error promedio del 7,6%. A pesar de su naturaleza bioinspirada, este algoritmo muestra una sorprendente adaptabilidad a problemas complejos, aunque se ve un aumento proporcional en el tiempo de ejecución y el error a medida que el número de ciudades aumenta, este aumento no es exponencial.

La comparativa con otros métodos para resolver el TSP resalta el buen desempeño del ABC. Aunque no ocupa el primer lugar, se posiciona favorablemente entre métodos heurísticos y otras técnicas reconocidas, ofreciendo soluciones competitivas y confiables.

La elección de Python como lenguaje de programación facilitó la implementación del algoritmo, así como la interacción con las bibliotecas TSPLIB95 y Plotly. Estas proporcionaron un entorno sencillo para cargar datos, realizar pruebas y visualizar resultados.

Si bien el ABC muestra resultados prometedores, se reconoce la existencia de oportunidades para mejorar tanto el algoritmo como su implementación. Las limitaciones computacionales al enfrentar un número creciente de ciudades resaltan la necesidad de seguir investigando y optimizando el rendimiento del algoritmo en escenarios más complejos.

En conclusión, este estudio destaca la eficacia del Algoritmo de Optimización de Colonia de Abejas como una técnica viable y competitiva en la resolución del Problema del Viajero. Sin embargo, persisten áreas de mejora y optimización que abren el camino para futuras investigaciones en este campo de la optimización combinatoria.

Finalmente, es importante destacar que en el archivo “.ipynb”, un notebook de Python, se presenta de forma interactiva todo el proceso de ejecución del algoritmo. En este archivo se proporciona una visualización dinámica y detallada de cómo evolucionan las rutas a lo largo de los ciclos de ejecución. Además, permite explorar de manera interactiva cada uno de los resultados obtenidos, brindando una comprensión más profunda del comportamiento del algoritmo y de las soluciones generadas.

Referencias

[1] M. Jünger, G. Reinelt and G. Rinaldi, "Chapter 4 The traveling salesman problem," *Handbooks in Operations Research and Management Science,* vol. 7, pp. 225-330, 1995. Available: https://www.sciencedirect.com/science/article/pii/S0927050705801215 DOI: 10.1016/S0927-0507(05)80121-5.

[2] D. Teodorovic and M. Dell'Orco, "Bee colony optimization–a cooperative learning approach to complex transportation problems," *Advanced OR and AI Methods in Transportation,* vol. 51, pp. 60, 2005. Available: https://neuro.bstu.by/ai/To-dom/My_research/failed%201%20subitem/For-courses/Job-SSP/Bee/ID161%5B1%5D.pdf .

[3] Anonymous "TSPLIB," Available: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/index.html .

[4] R. Matai, S. P. Singh and M. L. Mittal, "Traveling salesman problem: an overview of applications, formulations, and solution approaches," *Traveling Salesman Problem, Theory and Applications,* vol. 1, *(1),* pp. 1-25, 2010. Available: https://books.google.com/books?hl=en&lr=&id=gKWdDwAAQBAJ&oi=fnd&pg=PA1&dq=tsp+theory&ots=aa8x-57kJ1&sig=rrvuh-aetPvoGRFPKcVi9_jV340 .

[5] Anonymous "TSPLIB 95 — TSPLIB 95 0.7.1 documentation," Available: https://tsplib95.readthedocs.io/en/stable/pages/readme.html .

[6] N. Sultana *et al*, "Learning to optimise general TSP instances," *International Journal of Machine Learning and Cybernetics,* vol. 13, *(8),* pp. 2213-2228, 2022. Available: https://arxiv.org/pdf/2010.12214.pdf .