Revista de Investigación
Hatun Yachay Wasi 1(2), 43 - 57 (2022)
ISSN: 2955 - 8255

ARTICULO ORIGINAL

Recibido: Setiembre 2022
Aceptado: Diciembre 2022

METAHEURÍSTICAS CON PYTHON: CASOS PRÁCTICOS.

METAHEURISTICS WITH PYTHON: PRACTICAL CASES.

Jesús Espínola Gonzales[1] Ángel Cobo Ortega[2] Rocío Rocha Blanco[2]

[1]Universidad Tecnológica de Los Andes, Perú.
[2]Universidad de Cantabria, España.

Correspondencia:
Jesús Espínola Gonzales
jespinolg@utea.edu.pe

Como citar este articulo: Espinola, J., Cobo, A., & Rocha, R. (2022). Metaheurísticas con Python: casos prácticos. *Revista de Investigación Hatun Yachay Wasi, 1(2),* 43 - 57. https://doi.org/10.57107/hyw.v1i2.23

RESUMEN

El presente estudio muestra la potencialidad de uso de las técnicas metaheurísticas, para abordar problemas de toma de decisiones en contextos de gran complejidad y con aplicabilidad a una gran variedad de campos. Se presentan casos prácticos en el campo de la acuicultura y la gestión de granjas de cultivo de peces; así como, en la ingeniería inversa. Estos casos abordan problemas de optimización de gran complejidad por el alto número de variables y restricciones. Para resolver este tipo de problemas se presentan algunos métodos metaheurísticos implementados en Python, como herramientas útiles para conseguir soluciones satisfactorias. Las soluciones obtenidas han sido óptimas o cuasi-óptimas conseguidas en tiempos aceptables.

Palabras clave: Metaheurísticas, Python, Optimización.

ABSTRACT

This study shows the potential use of metaheuristic techniques to address decision-making problems in contexts of great complexity and with applicability to a wide variety of fields. Practical cases are presented in the field of aquaculture and the management of fish farms; as well as, in reverse engineering. These cases address highly complex optimization problems due to the high number of variables and restrictions. To solve this type of problems, some metaheuristic methods implemented in Python are presented as useful tools to achieve satisfactory solutions. The solutions obtained have been optimal or quasi-optimal achieved in acceptable times.

Keywords: Metaheuristics, Python, Optimization.

INTRODUCCIÓN

El término heurística proviene del griego: εὑρίσκειν heurískei, que podría traducirse como: 'hallar', 'inventar'. Según la Real Academia Española, heurística es la "técnica de la indagación y del descubrimiento" y que en algunas ciencias se puede definir como la "manera de buscar la solución de un problema mediante métodos no rigurosos, como por tanteo, reglas empíricas, etc". De una manera simple, se podría decir que una heurística busca la obtención, de una manera eficiente, de buenas soluciones al problema abordado. Esa eficiencia se orienta hacia la consecución de los objetivos básicos en computación: la obtención de buenas soluciones, preferentemente óptimas, y la reducción del esfuerzo y tiempo de cómputo. Las heurísticas pueden renunciar al objetivo de optimalidad siempre que la solución alcanzada se logre dentro de un espacio temporal razonable.

La complejidad de la sociedad actual, caracterizada por la cada vez mayor importancia de la toma de decisiones basada en datos, el uso intensivo de las tecnologías de la información y la presencia de múltiples criterios de decisión no ha hecho más que poner de manifiesto la importancia de disponer de metodologías que ayuden en los problemas decisionales cada vez más complejos.

Esta complejidad de los problemas es lo que provoca muchas veces la inexistencia o poca efectividad de algoritmos exactos de optimización, como los basados en cálculo diferencial (Larson & Bruce, 2012) o de aproximación numérica a la solución exacta (Burden et al., 2017). Problemas de optimización de gran tamaño, con cientos o incluso miles de variables de decisión y restricciones, con la presencia de funciones no lineales o incluso problemas de naturaleza combinatoria, han hecho que en las últimas décadas se haya desarrollado una importante labor de investigación para el desarrollo de modernas técnicas heurísticas de resolución de una enorme variedad de problemas prácticos. Estos métodos heurísticos también son buenos sustitutos de otros procedimientos exactos como Programación lineal, no lineal y dinámica (Bautista-Valhondo, 2020). En particular se han desarrollado métodos que combinan algoritmos heurísticos con otro tipo de técnicas de carácter estadístico, basadas en los principios de la inteligencia artificial, o bio-inspiradas que buscan mejorar la efectividad y rendimiento de los algoritmos. Este tipo de técnicas constituyen lo que se viene denominando "metaheurísticas", entre ellas se encuentran la computación evolutiva, los algoritmos genéticos, las técnicas de optimización inspiradas en colectivos de seres vivos. Según (Vob et al., 1999), "un algoritmo metaheurístico es un proceso iterativo que guía y modifica las operaciones de la heurística subordinada para producir de manera efectiva soluciones de alta calidad.

Puede manipular una sola solución en cada iteración o todo un conjunto de ellas; la heurística subordinada puede ser un procedimiento de búsqueda local o un método constructivo". Esta definición permite deducir la clasificación de las metaheurísticas en base a diferentes criterios. Por un lado, nos encontramos con métodos constructivos, que buscan la construcción de la solución añadiendo componentes paso a paso. Por ejemplo, eso ocurre en las técnicas de optimización de colonias de hormigas (*Ant Colony Optimization,* ACO).

Pero, por otro lado, existen técnicas de búsqueda local que parten de soluciones que tratan de ir mejorando paso a paso. Un ejemplo de estas últimas podría ser el templado simulado (Simulated Annealing). Las metaheurísticas se clasifican también en métodos de trayectoria, que en cada iteración manejan una única posible solución del problema, y metaheurísticas poblacionales que trabajan en todo momento con un conjunto de potenciales soluciones.

Independientemente del tipo de metaheurística, hay una serie de rasgos comunes que suelen caracterizar a todas ellas. En primer lugar, para evitar quedar atrapadas en óptimos locales, suelen incorporar mecanismos de construcción aleatoria o incluso aceptar en determinadas ocasiones empeoramientos locales en la búsqueda de la solución. El hecho de que integren estos mecanismos aleatorios hace que estas técnicas no suelan ser determinísticas, es decir, incluso con las mismas condiciones de partida no siempre generan la misma solución.

Aunque las soluciones sean diferentes y no se tenga garantía de su optimalidad, al menos las técnicas generan siempre soluciones eficientes, cercanas a las óptimas. Otro de los rasgos comunes de estas técnicas es el uso de estrategias de exploración del espacio de búsqueda de una manera eficiente y efectiva, buscando un balance entre exploración y explotación. La exploración o diversificación busca moverse por todo el espacio de búsqueda evitando que queden áreas sin explorar. La explotación o intensificación, en cambio, trata de focalizar la búsqueda en zonas prometedoras, es decir, en zonas del espacio de búsqueda con soluciones de calidad.

Siendo los metaheurísticos algoritmos no triviales, habitualmente con conjuntos de soluciones factibles de gran talla, y con la necesidad de realizar un elevado número de operaciones, resulta imprescindible la programación de estos algoritmos en algún lenguaje de programación. Python es una buena alternativa por varias razones, entre ellas el carácter intuitivo para programar, es un software libre, y existe una gran cantidad de librerías de libre disponibilidad en la red; un buen número de estas librerías incluyen métodos metaheurísticos.

Juntamente con el lenguaje de programación, es importante tener un buen entorno (aplicación) que permita compilar o interpretar el lenguaje con el que se esté programando. En este trabajo se ha usado Jupyter, el cual es un entorno integral y amigable; pues permite escribir texto en formato Latex, código de programación Python (módulos) y mostrar los resultados de la ejecución de los módulos, todo esto integrado en el mismo cuaderno de trabajo (notebook). Estos cuadernos interactivos permiten además la integración en un mismo elemento de explicaciones, gráficos, elementos multimedia y, por supuesto, código ejecutable.

MATERIALES Y MÉTODOS

Como se ha indicado, bajo este concepto general de metaheurísticas tienen cabida una gran variedad de técnicas. Un buen punto de partida, para conocer las características de algunas de las principales técnicas metaheurísticas es el trabajo de (Blum & Roli, 2003). A continuación, se presentan los fundamentos básicos de algunas de ellas.

Metaheurísticas de trayectoria

Dentro de esta categoría, una de las primeras heurísticas fue la búsqueda Tabú, estrategia propuesta por Glover en 1986 (Glover, F.; Kochenberger, G.A. (eds), 2003) y que se basa en la concepción de una búsqueda inteligente como la combinación de memoria y aprendizaje.

Este método de búsqueda prohíbe movimientos hacia puntos ya visitados, al menos en las siguientes iteraciones, pudiendo incluso aceptar cambios que empeoren las soluciones para evitar caminos ya explorados. Los movimientos previos quedan registrados en una “memoria” y se almacenen también una lista de direcciones de movimiento “tabú”, movimientos no permitidos.

Otra de las técnicas más populares dentro de esta categoría es el templado simulado (simulated anealing - SA), técnica propuesta por (Kirkpatrick y otros, 1983) que busca imitar el proceso para obtener estados de baja energía en un baño caliente. Cuando un sólido, por ejemplo un metal, se

calienta hasta un valor en el que el sólido se funde y posteriormente se produce un enfriamiento lento del sistema para que las partículas se organicen en un estado de baja energías, se observan que, especialmente a altas temperaturas, se producen cambios en la estructura que pueden provocar un aumento en esa energía. A temperaturas más bajas, la probabilidad de aumento de energía es menor.

Tomando esta inspiración, el SA es un procedimiento iterativo en el que todo movimiento de mejora en la función objetivo es aceptado, pero también se permiten movimientos que empeoran el valor de la función objetivo de acuerdo a una distribución de probabilidad en función de un parámetro de control.

Metaheurísticas poblacionales

Dentro de las metaheurísticas que trabajan con poblaciones de soluciones una mención especial debe hacerse a las que se engloban dentro del término Swarm Intelligence o inteligencia de enjambre. Se trata de un área de la inteligencia artificial que se orienta hacía el diseño de sistemas descentralizados y auto-organizados en los que la inteligencia "emerge" de la interacción entre agentes simples que interactúan entre sí y con el entorno. Algunas de las técnicas de esta categoría toman su inspiración en la naturaleza o sistemas biológicos como puedan ser comportamientos de colonias de hormigas, estrategias de caza de depredadores, vuelos de insectos o aves,… Entre las metaheurísticas más conocidas se encuentran los algoritmos genéticos (Holland, 1975), la optimización de colonias de hormigas (Dorigo, 1992) o la optimización de enjambres de partículas (Kennedy & Eberhart, 1995).

Metaheurísticas con Python

El lenguaje de programación Python es uno de los instrumentos computacionales de mayor potencialidad de uso en una enorme variedad de campos. Su popularidad ha aumentado notablemente en los últimos años por su adaptabilidad a procesos de gestión eficaz de datos masivos, por su simplicidad y modularidad, además por responder a la filosofía de distribución en abierto de los códigos fuente (open source).

Desde el punto de vista de la computación científica, existen un buen número de librerías Python de utilidad. Entre las básicas y más conocidas se podrían citar la librería numérica Numpy, la librería Scipy, que implemente técnicas avanzadas, o la librería Matplotlib con rutinas gráficas y de visualización. Otro de los paquetes de gran utilidad en el campo de la ciencia de datos es la librería de machine learning scikit-learn.

Algunas de estas librerías implementan técnicas metaheurísticas; por ejemplo, dentro de la librería Scipy el módulo de optimización (optimize) implementa diversas técnicas para la localización de máximos y mínimos de funciones. Incluye solvers para problemas lineales y no lineales, entre ellas algunas técnicas metaheurísticas. Por ejemplo, hasta la versión 0.14 la técnica de *simulated annealig* se encontraba implementada en la función *scipy optimize anneal*, a partir de esa versión se puede encontrar una generalización de la misma en la función scipy.optimize.basinhopping().

También es posible encontrar implementaciones individuales de otras metaheurísticas. Por ejemplo, la implementación del algoritmo de colonias de hormigas que se puede encontrar en: https://github.com/johnberroa/Ant-Colony-Optimization

Pero existen librerías de funciones Python especializadas en la implementación de metaheurísticas. Una de las más conocidas es la librería Inspired (https://pythonhosted.org/inspyred/) (Garrot, 2019) especializada en metaheurísticas bio-inspiradas. Las técnicas estándar que se encuentran implementadas son:

Genetic Algorithm
Evolution Strategy
Simulated Annealing
Differential Evolution Algorithm
Estimation of Distribution Algorithm
Pareto Archived Evolution Strategy (PAES)
Nondominated Sorting Genetic Algorithm (NSGA-II)
Particle Swarm Optimization
Ant Colony Optimization

En su concepción, la librería se basa en el principio básico de separación entre lo que serían componentes específicas del problema a abordar y componentes específicas del algoritmo a utilizar. Entre las componentes específicas que el programador debe aportar se encuentran la definición de cómo las soluciones son creadas (generator) y cómo se mide el ajuste de cada solución (evaluator).

Los elementos específicos del algoritmo serían un sistema de monitorización del estado de la evolución (observer); el criterio de terminación (terminator); un sistema de selección de individuos "padres" para la construcción de soluciones "hijas" (selector); proceso de construcción de esas soluciones "hijas" (variator); determinación de los individuos que sobreviven para una siguiente generación (replacer); proceso que determina cómo las soluciones son transferidas entre diferentes poblaciones (migrator); y determinación de cómo las soluciones son archivadas y guardadas (archiver).

Seguidamente se presenta como ejemplo un problema que se ha resuelto aplicando dos metaheurísticas implementadas en la librería Inspired. Se toma como referencia uno de los problemas más conocidos de la optimización combinatoria:

el problema del agente viajero, problema que suele ser habitual utilizar como prueba para comprobar la efectividad de diferentes técnicas metaheurísticas. Además, haciendo referencia a (Garfinkel, 1985), este problema contiene los dos elementos que hacen atractivo un problema a los matemáticos: planteamiento sencillo y dificultad de resolución.

Supongamos, por ejemplo, que hay 26 ciudades de las cuales se tienen como referencia su latitud y longitud terrestre, y se trata de organizar un vuelo, que empezando en la ciudad "A" se visiten todas las ciudades y se vuelva a la ciudad inicial "A", con las condiciones que, excepto la ciudad inicial, cada ciudad solo se visite una vez y que la distancia total recorrida sea mínima.

Existen diferentes algoritmos metaheurísticos para abordar este problema, aquí se hará uso de dos metaheurísticas: Computación evolutiva y Colonia de hormigas, ambas implementadas en la librería Inspired. En la Tabla 1 se presenta las ciudades junto con sus referencias geográficas (latitud, longitud).

TABLA 1

Referencias terrestres de las ciudades

Ciudad	(latitud, longitud)	Ciudad	(latitud, longitud)
A	(1149.0, 1761.0)	**N**	(505.0, 710.0)
B	(629.0, 1660.0)	**O**	(650.0, 920.0)
C	(41.0, 2091.0)	**P**	(1380.0, 1200.0)
D	(649.0, 1101.0)	**Q**	(199.0, 590.0)
E	(749.0, 2031.0)	**R**	(365.0, 860.0)
F	(1030.0, 2069.0)	**S**	(1030.0, 960.0)
G	(1649.0, 651.0)	**T**	(580.0, 1390.0)
H	(1488.0, 1625.0)	**U**	(835.0, 1785.0)
I	(787.0, 2261.0)	**V**	(491.0, 514.0)
J	(709.0, 1312.0)	**W**	(1890.0, 1220.0)
K	(842.0, 551.0)	**X**	(1250.0, 1600.0)
L	(1155.0, 2319.0)	**Y**	(1290.0, 795.0)
M	(900.0, 1350.0)	**Z**	(495.0, 211.0)

Los datos de la Tabla 1 se han ingresado en el módulo de Python, donde ya se tenía incorporado la librería Inspired; ejecutando dicho módulo se han obtenido los siguientes resultados.

Solución 1 – Usando el Algoritmo de Computación Evolutiva.

El problema se ha resuelto haciendo uso del algoritmo de Computación Evolutiva, parte de la librería Inspired, su código fuente está disponible en https://pythonhosted.org/inspyred/ y se muestra a continuación

La solución obtenida con el siguiente algoritmo es el recorrido empezando en la ciudad A:

[A, F, L, I, C, E, U, B, T, D, N, O, K, Z, V, Q, R, J, M, S, Y, G, W, P, H, X, A]

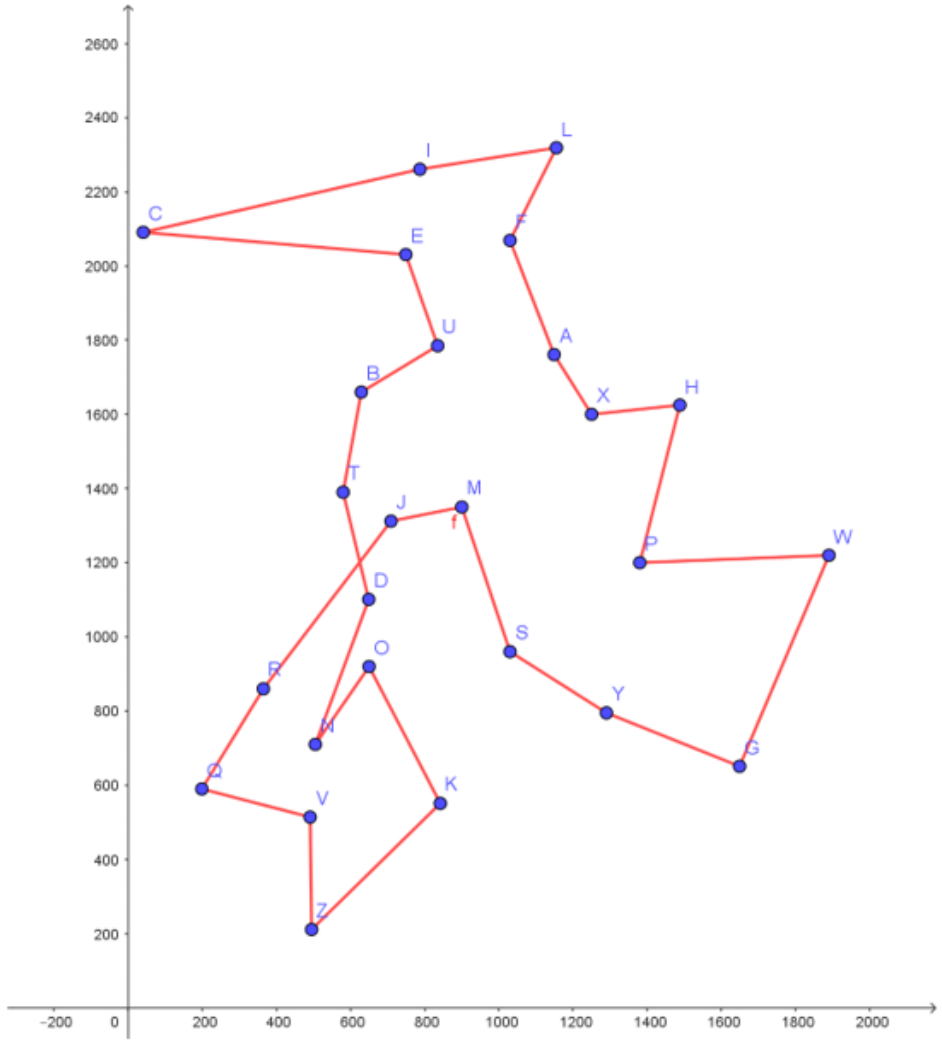
Esta ruta, cuya representación gráfica puede verse en la Figura 1, supone una distancia total recorrida: de 9891.

```
from random import Random
from time import time
import math
import inspyred

def main(prng=None, display=False):
    if prng is None:
        prng = Random()
        prng.seed(time())
    points = [(1149.0, 1761.0), (629.0, 1660.0), (41.0, 2091.0), (649.0, 1101.0),
              (749.0, 2031.0), (1030.0, 2069.0), (1649.0, 651.0), (1488.0, 1625.0),
              (787.0, 2261.0), (709.0, 1312.0), (842.0, 551.0), (1155.0, 2319.0),
              (900.0, 1350.0), (505.0, 710.0), (650.0, 920.0), (1380.0, 1200.0),
              (199.0, 590.0), (365.0, 860.0), (1030.0, 960.0), (580.0, 1390.0),
              (835.0, 1785.0), (491.0, 514.0), (1890.0, 1220.0), (1250.0, 1600.0),
              (1290.0, 795.0), (495.0, 211.0)]

    weights = [[0 for _ in range(len(points))] for _ in range(len(points))]
    for i, p in enumerate(points):
        for j, q in enumerate(points):
            weights[i][j] = math.sqrt((p[0] - q[0])**2 + (p[1] - q[1])**2)

    problem = inspyred.benchmarks.TSP(weights)
    ea = inspyred.ec.EvolutionaryComputation(prng)
    ea.selector = inspyred.ec.selectors.tournament_selection
    ea.variator = [inspyred.ec.variators.partially_matched_crossover,
                   inspyred.ec.variators.inversion_mutation]
    ea.replacer = inspyred.ec.replacers.generational_replacement
    ea.terminator = inspyred.ec.terminators.generation_termination
    final_pop = ea.evolve(generator=problem.generator,
                          evaluator=problem.evaluator,
                          bounder=problem.bounder,
                          maximize=problem.maximize,
                          pop_size=100,
                          max_generations=50,
                          tournament_size=5,
                          num_selected=100,
                          num_elites=1)

    if display:
        best = max(ea.population)
        print('Mejor Solución: {0}: {1}'.format(str(best.candidate), 1/best.fitness))
    return ea

if __name__ == '__main__':
    main(display=True)
```


Hatun Yachay Wasi 1(2), 2022 ISSN: 2955 - 8255

FIGURA 1

Resultado usando el Algoritmo de Computación Evolutiva.



Solución 2 – Usando el Algoritmo de Colonia de Hormigas.

El problema también se ha resuelto haciendo uso de otra metaheurística, el Algoritmo de Colonia de Hormigas, la solución obtenida es el siguiente recorrido empezando en la ciudad A:
[A, W, H, L, F, I, U, E, B, C, T, J, M, D, O, N, V, Z, Q, R, K, S, Y, G, P, X, A]

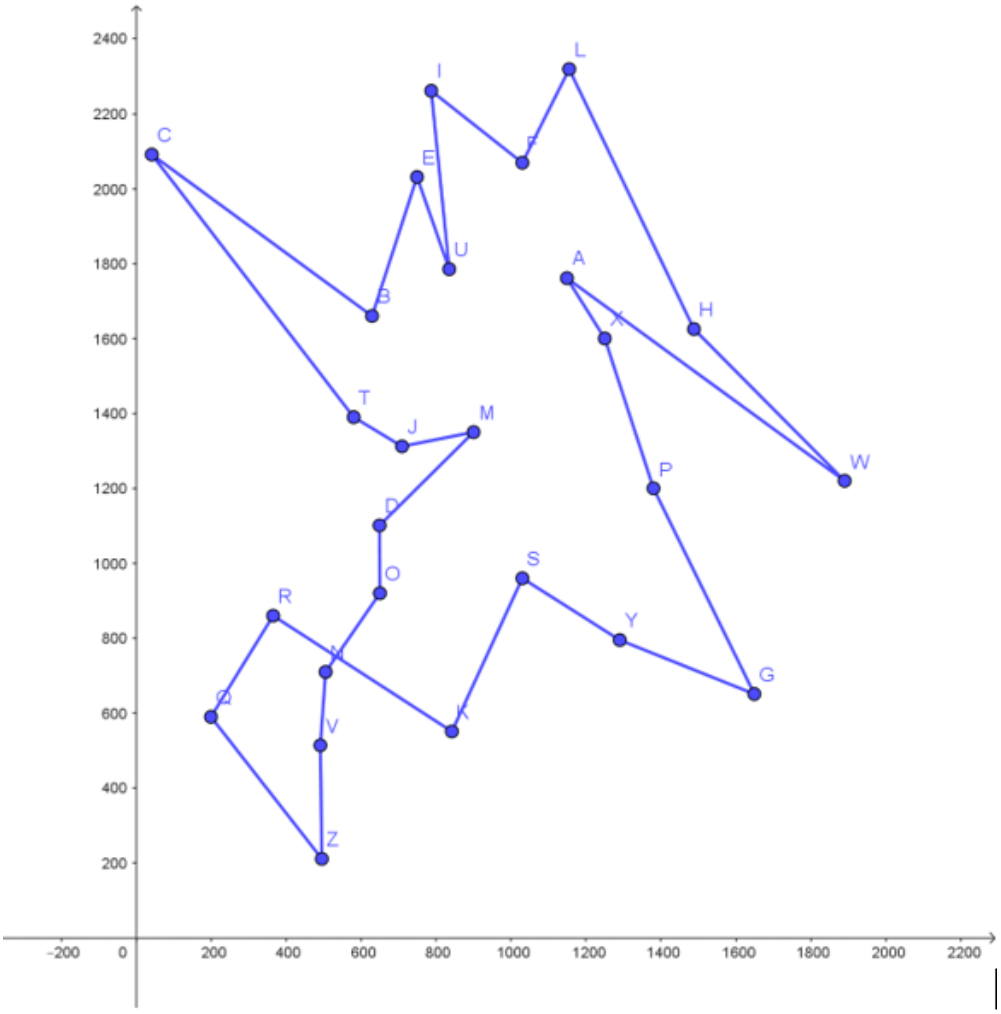
De nuevo, dicho recorrido se muestra gráficamente en la Figura 2, y la distancia total recorrida es de 10957

Esta metaheurística también es parte de la librería Inspired, el código fuente del módulo respectivo también está disponible en https://pythonhosted.org/inspyred/ y se reproduce a continuación.

```
from random import Random
from time import time
import math
import inspyred

def main(prng=None, display=False):
    if prng is None:
        prng = Random()
        prng.seed(time())
    points = [(1149.0, 1761.0), (629.0, 1660.0), (41.0, 2091.0), (649.0, 1101.0),
              (749.0, 2031.0), (1030.0, 2069.0), (1649.0, 651.0), (1488.0, 1625.0),
              (787.0, 2261.0), (709.0, 1312.0), (842.0, 551.0), (1155.0, 2319.0),
              (900.0, 1350.0), (505.0, 710.0), (650.0, 920.0), (1380.0, 1200.0),
              (199.0, 590.0), (365.0, 860.0), (1030.0, 960.0), (580.0, 1390.0),
              (835.0, 1785.0), (491.0, 514.0), (1890.0, 1220.0), (1250.0, 1600.0),
              (1290.0, 795.0), (495.0, 211.0)]
    weights = [[0 for _ in range(len(points))] for _ in range(len(points))]
    for i, p in enumerate(points):
        for j, q in enumerate(points):
            weights[i][j] = math.sqrt((p[0] - q[0])**2 + (p[1] - q[1])**2)

    problem = inspyred.benchmarks.TSP(weights)
    ac = inspyred.swarm.ACS(prng, problem.components)
    ac.terminator = inspyred.ec.terminators.generation_termination
    final_pop = ac.evolve(generator=problem.constructor,
                          evaluator=problem.evaluator,
                          bounder=problem.bounder,
                          maximize=problem.maximize,
                          pop_size=10,
                          max_generations=50)

    if display:
        best = max(ac.archive)
        print('Best Solution:')
        for b in best.candidate:
            print(points[b.element[0]])
        print(points[best.candidate[-1].element[1]])
        print('Distance: {0}'.format(1/best.fitness))
    return ac

if __name__ == '__main__':
    main(display=True)
```

FIGURA 2

Resultado usando el Algoritmo de Colonia de Hormigas.



Comparando los resultados, se ve claramente que la solución 1 es mejor aproximación que la segunda. No obstante, también es cierto que analizando las representaciones gráficas de las rutas obtenidas en ambas soluciones se podrían realizar algunos pequeños cambios que mejorarían un poco la calidad de la solución.

Sabido es que, en general, haciendo uso de las metaheurísticas no se tiene solución única, ni necesariamente la mejor; sí que se tienen buenas aproximaciones a la solución. Esto se ve compensado con el tiempo de cálculo que es muy pequeño comparado con el tiempo que se tardaría intentando hallar la solución óptima.

En el ejemplo mostrado, usando dos metaheurísticas distintas se tienen dos soluciones diferentes. Pero, también podrían obtenerse soluciones distintas ejecutando el módulo con el mismo algoritmo metaheurístico, debido al carácter no determinista de ambas metaheurísticas.

Por lo que en el proceso de obtener una buena

aproximación se puede ejecutar el módulo repetidas veces y quedarse con la mejor aproximación en función del valor de la función objetivo, que para el ejemplo sería la menor distancia recorrida.

RESULTADOS Y DISCUSIÓN

Aplicaciones prácticas

Los campos de aplicación práctica de las metaheurísticas son amplísimos. Son especialmente apropiadas para abordar problemas de optimización de gran complejidad, por el número de variables y restricciones, o por la propia naturaleza de las funciones que intervienen. En esta sección se hace referencia a algunos de los problemas prácticos en los que los autores de este trabajo han participado en su resolución exitosa mediante al uso de metaheurísticas implementadas en Python.

En los casos presentados, se remite a las correspondientes publicaciones de los autores para obtener información más detallada sobre la implementación y uso de las correspondientes metaheurísticas. En todos los casos, se consiguieron buenos resultados haciendo uso de metaheurísticas poblacionales.

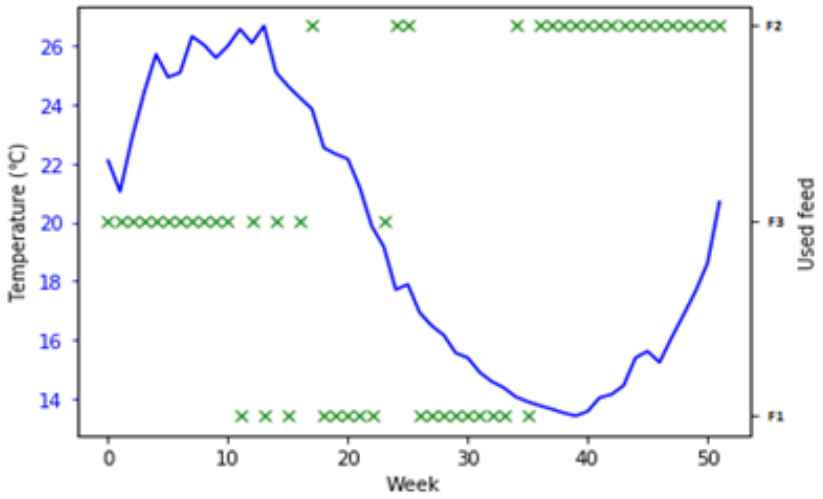
En el campo de la acuicultura y la gestión de granjas de cultivo de peces, uno de los principales retos de los productores es la adecuada planificación de los procesos de cría y engorde de los peces. Las decisiones que se debe tomar en ese contexto son de enorme complejidad y se ven afectadas por factores biológicos, técnicos, ambientales y económicos. Uno de los problemas de mayor impacto en la actividad de producción es la adecuada elección de los piensos a aportar en las distintas fases del engorde.

En su trabajo, (Luna et al., 2022) utilizan algoritmos genéticos para determinar las secuencias de piensos más apropiadas en función de las condiciones ambientales de la granja y el tamaño de los peces. Gracias al modelo desarrollado han podido comprobar cómo la combinación de piensos permite mejorar los resultados. A modo de ejemplo, la Figura 3 muestra la obtención de una planificación óptima de la combinación de tres piensos diferentes durante las 54 semanas de engorde de un lote de doradas en una granja situada en el Mediterráneo.

En ese ejemplo, se disponía para el engorde de tres piensos: uno de alta rentabilidad, buenas tasas de engorde y relación calidad/precio, otro pienso con mayores prestaciones y precio, orientado a periodos de menor crecimiento (invierno) y, finalmente, un tercero adecuado para una producción orgánica/ ecológica, con altas prestaciones y de mayor calidad, pero también el más caro de los tres.

FIGURA 3

Planificación de la utilización de 3 piensos de engorde durante un periodo de 54 semanas.



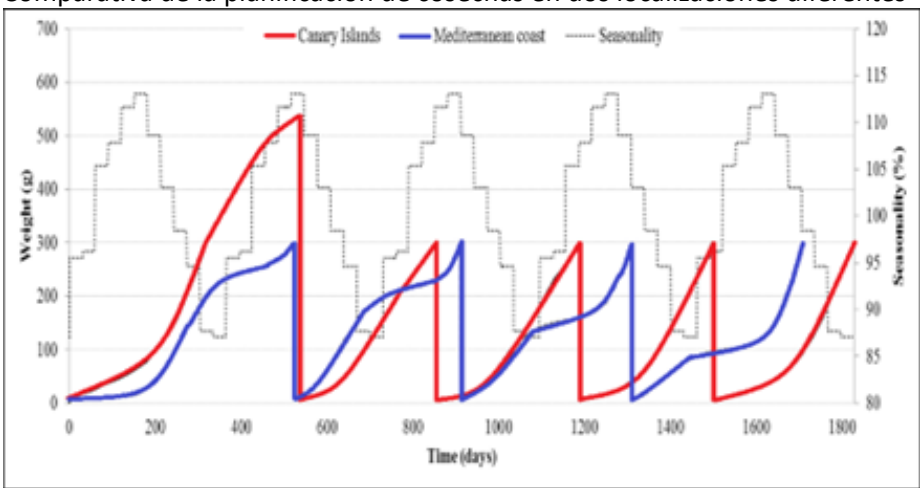
Otro de los problemas que deben enfrentar los productores es la adecuada planificación de los procesos de siembra y despesque de los tanques. Es decir, cuál es el momento óptimo para llenar un tanque de alevines e iniciar el proceso de engorde, y en qué momento comercializar el pez adulto. Es un problema de gran complejidad, ya que deben tener en cuenta restricciones comerciales (compromisos adquiridos con clientes), técnicas (capacidades de los tanques) y ambientales. (Luna et al., 2020) han utilizado para este propósito una metaheurística población como es el PSO (*Particle Swarm Optimization*).

Gracias al uso de esta técnica se le pueden proporcionar al productor un conjunto de planificaciones cercanas al óptimo, definiendo cada planificación como una secuenciación de cosechas dentro del horizonte temporal preestablecido.

Por ejemplo, la Figura 4 muestra cómo gracias a la metaheurística se puede determinar cómo en una mejor localización, en este caso las islas Canarias, es posible incluso planificar una cosecha más y aumentar el rendimiento económico de la explotación.

FIGURA 4

Comparativa de la planificación de cosechas en dos localizaciones diferentes



Estos problemas en el campo de la acuicultura se han abordado de manera exitosa con Python gracias a las posibilidades de implementación de metaheurísticas, conectividad a bases de datos para recuperar la información necesaria para definir un modelo de crecimiento, y la gestión eficaz de todos los datos necesarios (características de los piensos, precios de insumos, precios de venta, condiciones ambientales de la explotación, temperaturas, …)

Un tercer problema que se presenta en este trabajo es en el campo del diseño y fabricación de automóviles y fuselaje de aviones; en el que también los autores han participado (Gálvez et al., 2007). Es un problema de ingeniería inversa.

En este campo es frecuente representar matemáticamente las curvas y superficies mediante curvas y superficies de Bézier. La curva tiene la siguiente representación

$$C(t)=\sum_{j=0}^{M} P_j B_j(t)$$

Donde $t \varepsilon [0,1]$, P_j son los puntos de control y B_j las funciones de base, de grado "d", que se calculan como : $B_i^d(t)=\binom{d}{i} t^i (1-t)^{d-i}$

Mientras que las superficies de Bézier tienen la representación

$$S(u,v)=\sum_{i=0}^{N}\sum_{j=0}^{M} P_{ij} B_i(u) B_j(v)$$

Donde $u,v \in [0,1]$, P_{ij} son los puntos de control, B_i y B_j las funciones de base, que se calculan como en la representación de las curvas de Bézier.

En cada caso, el problema consiste en: partiendo de un conjunto de puntos 3D, que se obtienen a partir de mediciones sobre piezas prototipos, determinar la representación de curva o superficie (según corresponda) que se ajuste a dichos puntos. De nuevo, este es un problema de gran complejidad, por el gran número de variables que se generan, pues las variables a determinar son los valores apropiados para los parámetros "t" o "(u,v)" y los puntos de control respectivos.

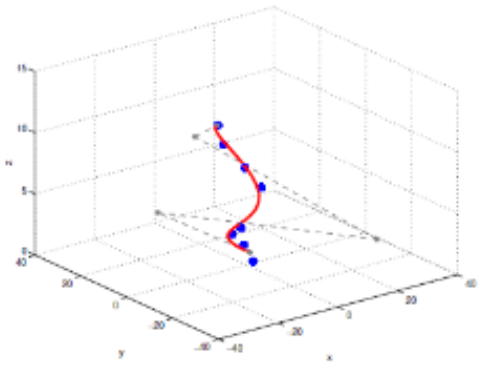
A modo de ejemplo, las Figuras 5 y 6 muestran los ajustes realizados de sendas nubes de puntos, utilizando algoritmos genéticos para optimizar la determinación de la parametrización correspondiente, tanto de curvas como de superficies. En ambos casos las soluciones obtenidas son de calidad.

Para el caso de una curva, se han seleccionado ocho puntos para ser ajustados mediante una curva de Bézier de grado 4 con cinco puntos de control. Esto genera 15 variables: ocho valores para "t" asociados con los ocho puntos 3D dados y 15 variables que son las coordenadas de los puntos de control (cinco puntos de control).

Haciendo uso de algoritmos genéticos se tuvo el vector paramétrico óptimo [0.0,0.0131,0.0583,0.3556,0.5384,0.7138,0.7899,1.0]. En la Figura 5 se muestra los puntos dados (esferas en color azul), la curva de Bézier de ajuste (en color rojo) y los puntos de control (estrellas en color gris).

FIGURA 5

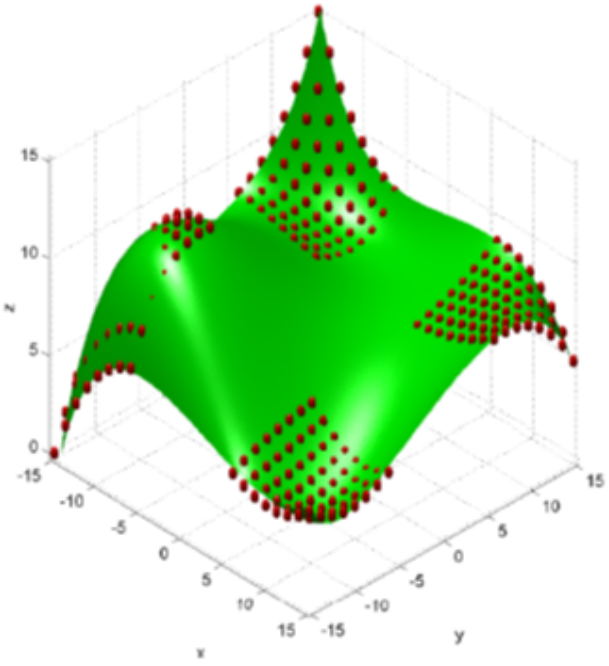
Curva de Bézier de ajuste de puntos 3D.



Para el caso de una superficie, se han seleccionado 256 puntos para ser ajustados mediante una superficie de Bézier bicúbica, es decir grado 3 en el parámetro u y grado 3 en el parámetro v, con 16 puntos de control. Para cada uno de los parámetros u y v se ha fijado que tomen dos grupos de valores equidistantes en los intervalos [0,0.2] y [0.8,1]. Así el número de variables es: 48 coeficientes escalares (que corresponden a las coordenadas de los 16 puntos de control 3D), más los 32 escalares, que corresponden a los valores paramétricos parámetros u y v (16 para cada uno). En total se tienen 80 variables. De nuevo, haciendo uso de algoritmos genéticos se tuvo el resultado que se muestra en la Figura 6.

FIGURA 6

Superficie de Bézier de ajuste de puntos 3D



CONCLUSIONES

- Se mostró la potencialidad de uso de las técnicas metaheurísticas, para abordar problemas de gran complejidad y en una gran variedad de campos. En particular, en campos vinculados con gestión de la producción e ingeniería.
- Se muestran aplicaciones a la planificación de la producción acuícola y procesos de ingeniería inversa orientados a la reconstrucción de curvas y superficies.
- Los resultados obtenidos en los casos prácticos presentados son soluciones óptimas o cuasi-óptimas conseguidas en tiempos aceptables y que ilustran a la perfección la principal utilidad de estas técnicas: obtención de soluciones razonables y eficientes de una manera rápida y simple, sin necesidad de grandes requerimientos sobre la naturaleza de las funciones que intervienen en el problema.
- El lenguaje de programación Python permite la implementación de metodologías de optimización caracterizadas por su adaptabilidad, flexibilidad y efectividad, como son las metaheurísticas. Además, la concepción de este lenguaje de programación como una tecnología de código abierto hace que existan un gran número de librerías y módulos que pueden ser reutilizados, minimizando el esfuerzo de programación necesario para la aplicación práctica de estas técnicas de optimización
- El hecho que las metaheurísticas presenten diferentes soluciones cuasi-óptimas puede ser beneficioso para el tomador de decisiones, al conjugar estas opciones con otras consideraciones que intervengan en la toma de tales decisiones.
- Las metaheurísticas poblacionales como los algoritmos genéticos o las técnicas PSO aportan al decisor todo un conjunto de posibles buenas soluciones, de entre las cuáles se deberá elegir una para su aplicación práctica. Disponer de diferentes alternativas de calidad es sin lugar a duda una ventaja desde el punto de vista de la decisión final.

REFERENCIAS BIBLIOGRÁFICAS

Bautista-Valhondo, J. (2020). Metaheurísticas en ingeniería. Madrid: Dextra Editorial S.L.

Blum, C., & Roli, A. (2003). Metaheuristics in Combinatorial Optimization: Overview and Conceptual Comparison. ACM Computing Surveys, 35, 268-308.

Burden, R., Douglas, J., & Burden, A. (2017). Análisis numérico. Cengage.

Dorigo, M. (1992). Optimization, Learning and Natural Algorithms [in Italian]. PhD thesis, Dipartimento di Elettronica, Politecnico di Milano.

Espinola, J. (2021). Apoyo al aprendizaje autónomo de la derivada y sus aplicaciones en bachillerato mediante laboratorios virtuales con Python. Trabajo Fin de Máster: Máster Univesitario en Formación del Profesorado de Educación Secundaria obligatoria y Bachillerato, Formación Profesional y Enseñanzas de Idiomas (Matemáticas). UNED. Madrid.

Gálvez, A., Iglesias, A., Cobo, Á., Puig-Pey, J., & Espinola, J. (2007). Bezier Curve and Surface Fitting of 3D Point Clouds Through Genetic Algorithms, Functional Networks and Least-Square Approximation. Lecture Notes in Computer Science, 4706, 680-693.

Garrot, A. (2019). Inspyred: Bio-inspired algorithms in Python. Recuperado el 30 de noviembre de 2022, de https://pythonhosted.org/inspyred/

Glover, F.; Kochenberger, G.A. (eds). (2003). Handbook of Metaheuristics. Kluwer's International Series.

Holland, J. (1975). Adaptation in Natural and Artificial Systems. Ann Arbor: University of Michigan Press.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. Proceedings of ICNN'95 - International Conference on Neural Networks, 4, págs. 1942-1948.

Kirkpatrick, S., Gelatt, C., & Vecchi, M. (1983). Optimization by Simulated Annealing. Science, 220(4598), 671-680.

Larson, R., & Bruce, H. (2012). Multivariable calculus. Cengage.

Luna, M., Llorente, I., & Cobo, A. (2020). Aquaculture production optimisation in multi-cage farms subject to commercial and operational constraints. Biosystems Engineering, 196, 29-45.

Luna, M., Llorente, I., & Cobo, A. (2022). Determination of feeding strategies in aquaculture farms using a multiple-criteria approach and genetic algorithms. Annals of Operations Research, Springer, 314(2), 551-576.

Talbi, E.-G. (2009). Metaheuristics. JOHN WILEY & SONS INC.

Vob, S.; Martello, S.; Osman, I.H.; Roucairol, C. (eds). (1999). Meta-Heuristics: Trends in Local Search paradigms for Optimization. Kluwer Academic Publishers.