- Añadir cosas opcionales forma parte de la nota (0,5 - 1 punto)
- Hay una mutación que si se modifica un poco da muy buenos resultados
- Meter resultados con comparaciones estadísticas en la presentación (10 veces -> media / std ; plots / tablas)
- TSP -> el mejor grupo tiene un extra de 0,5

El profe comenta que a lo mejor se puede usar un modelo de islas

61AH - Máster Universitario en

Aprendizaje Automático y Datos Masivos

2025/26

Carlos Camacho Gómez

Francisco Serradilla García

Cristian Ramírez Atencia

1.Normativa

Esta práctica evaluable está sujeta a una normativa enumerada en los siguientes puntos:

- La práctica se realizará en grupos de hasta 3 alumnos.
- La realización de la práctica se hará en horario de clase.
- La evaluación de la práctica se llevará a cabo el 9 de enero de 2026 a las 18:30.
- Cada grupo deberá realizar una presentación (20 minutos, máximo 25) en donde: se explique el algoritmo implementado; se recojan los resultados obtenidos; se justifiquen las decisiones tomadas a la hora de diseñar el algoritmo evolutivo; se justifiquen los cambios, si los hubiese, resultantes de aplicar dicho algoritmo a una función de fitness u otra; se realicen comparaciones de resultados entre algoritmos a través de imágenes y tablas; y se detalle el reparto de trabajo realizado dentro del grupo.
- Los profesores se reservan el derecho a realizar preguntas a los integrantes del grupo durante y al final de la exposición.
- Se valorará la calidad de los resultados obtenidos.
- Se valorará la calidad de la presentación, su estructura, organización, análisis de los resultados y las conclusiones presentadas.
- Tanto el código desarrollado, los ficheros *CodGrupo_CodProblema.csv* que se pidan, así como la presentación, deberá subirla **un único miembro por cada grupo** a la tarea de Moodle que encontraréis en el apartado de la práctica.
- Se valorará la claridad del código implementado.
- La fecha de cierre de esta tarea será el viernes 9 de enero de 2026 a las 15:30.
- Para aprobar la asignatura es necesario obtener un 5/10 puntos en esta nota.

2.Enunciado

Los algoritmos evolutivos son meta-heurísticas de búsqueda de carácter general, es decir, que pueden aplicarse a un gran número de problemas de optimización. Por eso mismo, en esta práctica se propone los siguientes objetivos:

1. Que los grupos elaboren sus propios algoritmos evolutivos desde cero y que los prueben en varios problemas de optimización monoobjetivo.
2. Para cada uno de estos problemas, que comparen sus resultados debidamente con algún algoritmo de tipo enjambre.
3. Que adapten el algoritmo desarrollado en el punto 1 a problemas multiobjetivo.
4. Que aprendan a trabajar con la librería Pymoo, para optimizar problemas multiobjetivo.

La implementación se llevará a cabo en varios notebooks de Jupyter y se ejecutará sobre Python 3.

2.1.Problemas monoobjetivo

En el proceso de diseño, los alumnos tendrán que tomar la decisión de qué codificación, qué procedimientos de selección, cruce, mutación y reemplazo van a usar y justificarlos debidamente. A su vez, los problemas de optimización tienen características muy diferentes, por lo que los grupos tendrán la libertad de hacer cualquier modificación en sus algoritmos para poderlos aplicar en una u otra función, siempre y cuando lo justifiquen debidamente.

Conforme avancemos en la teoría, iremos viendo algoritmos que incorporan heurísticas más avanzadas, como, por ejemplo, mutaciones diferenciales, hibridaciones con búsquedas locales, etc. Los grupos también tiene libertad de introducir mejoras competitivas en sus algoritmos, sustituyendo o hibridando partes del mismo. Se valorará positivamente si el grupo es capaz de demostrar que los cambios son a mejor.

Como sabéis, estos algoritmos son estocásticos. Por tanto, no vale con lanzar un experimento y rezar porque salga muy bien. Debéis lanzar al menos 10 experimentos y sacar algunos estadísticos (a elegir por el grupo) para mostrar los resultados en las presentaciones.

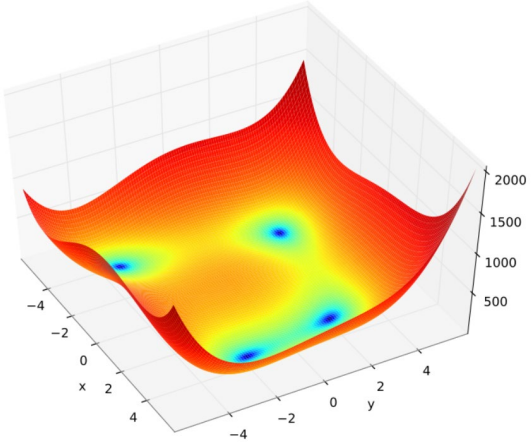
Las funciones a optimizar son:

1. Función de Himmelblau. Es una función con dos variables cuyo espacio de búsqueda es real y tiene una peculiaridad, consta de 4 óptimos globales. La función se expresa matemáticamente como:

$$f(x,y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2$$

El espacio de búsqueda de las variables será [-5,+5]. Cada grupo **debe implementar** esta función. Los algoritmos deben estar configurados para que su **condición de parada sea 3.5k evaluaciones** (llamadas a la función de fitness, no iteraciones del algoritmo o generaciones).

Opcional: puede comparar su algoritmo con el PSO de la siguiente librería: pyswarms.

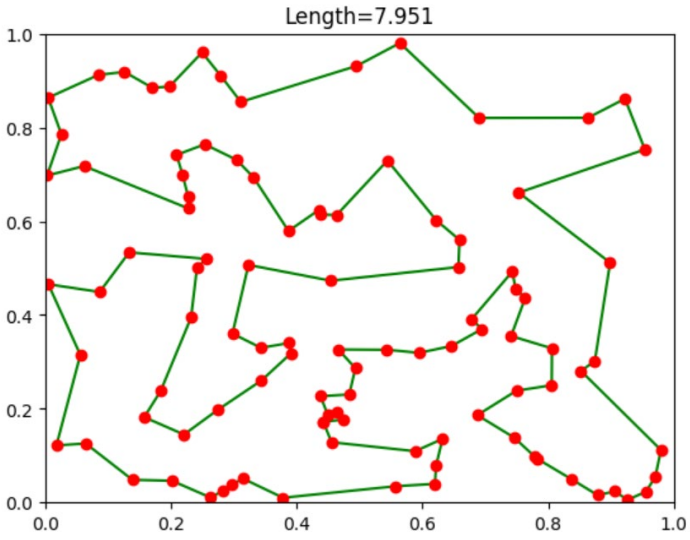


2. *Travelling Salesman Problem* de N=100 ciudades. El problema trata de minimizar la distancia recorrida por un vendedor, que tiene que pasar únicamente una vez por cada ciudad y volver a la ciudad inicial.

$$Min\ F(X) = C(x_N, x_1) + \sum_{i=1}^{N-1} C(x_i, x_{i+1})$$

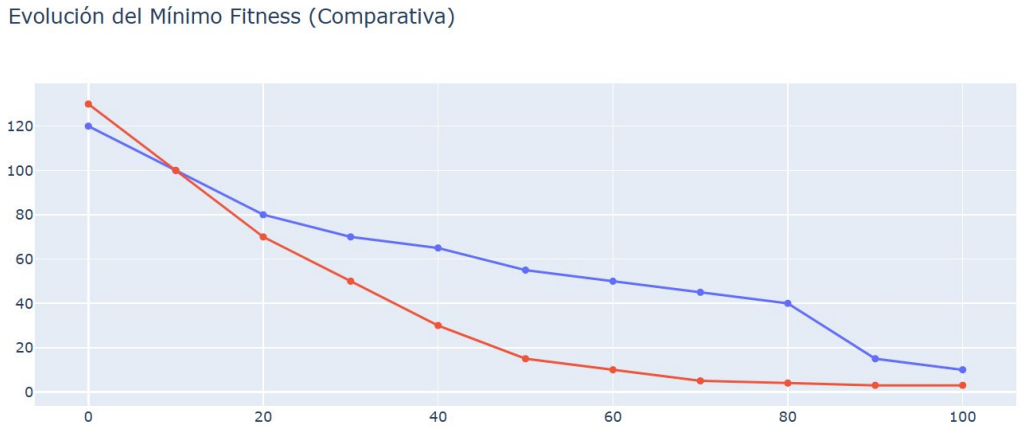
Hay mejores a 7,951

Se parte de un grafo completo, en donde los vértices son las ciudades y las aristas es el coste de ir de una ciudad a otra. Los alumnos pueden encontrar esta función ya implementada en el recurso de Moodle "**Notebook para la práctica**". Los algoritmos deben estar configurados para que su **condición de parada sea 1M evaluaciones**. La representación de la solución óptima (o muy cercana a la óptima) es la que se muestra en la imagen de la derecha.



Opcional: puede comparar los resultados de su algoritmo con el ACO que encontrarás en la siguiente librería: aco.

Para poder realizar una comparativa entre vuestros algoritmos, debéis guardar un histórico con el mejor fitness de la población en cada iteración, de forma que se puedan pintar gráficas como la siguiente:



Sólo para el problema del TSP, guardad esta información con el siguiente formato:

- Fichero *CodGrupo_TSP.csv*
- Condición de parada = 1M de evaluaciones
- Dos columnas:
  - La primera, llamada *generation*
  - La segunda, llamada *min_fitness*

Por último, tendréis que subir a la tarea de Moodle el fichero .csv del experimento que alcanzó vuestra mejor solución.

2.2.Problemas multiobjetivo

Después de completar la parte monoobjetivo, habéis aprendido a diseñar algoritmos meta-heurísticos y los habéis utilizado en problemas que se caracterizaban por tener una única solución óptima, y vuestro objetivo y el de vuestros algoritmos, era encontrarla.

En esta parte vais a adaptar y utilizar vuestros algoritmos diseñados anteriormente para resolver problemas de naturaleza multi-objetivo. En este tipo de problemas, existen un conjunto de soluciones que no se dominan entre sí, es decir, que son óptimas. A este conjunto se le llama **Pareto-optimal**. Los valores de las funciones de fitness representado en su espacio forman lo que se denomina **Frente de Pareto**. Vuestro nuevo objetivo y el de vuestros algoritmos, es encontrar ese conjunto de soluciones óptimas.

En esta parte vais a resolver un total de cuatro problemas de optimización y os vais a comparar con el algoritmo NSGA2, el rival a batir. Usaremos la librería **Pymoo** ya que nos ofrece una implementación de 3 de los 4 problemas de optimización y del algoritmo NSGA2. La interfaz de usuario de Pymoo está diseñada para ser bastante sencilla e intuitiva, especialmente para aquellos que ya tienen algo de experiencia con Python y la optimización. Como podréis ver a través de este enlace, cuenta con una documentación detallada y ejemplos claros que ayudan a los usuarios a entender cómo utilizar las diversas funcionalidades de la librería. También nos permite crear nuestros propios problemas personalizados e integrarlos de manera fácil. Por último, nos ofrece una herramienta para visualizar nuestros resultados e interpretarlos.

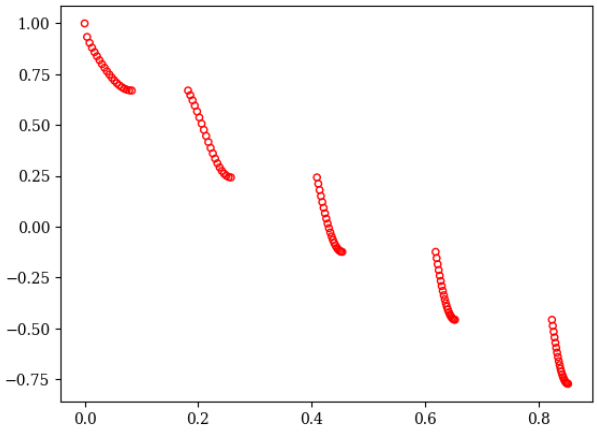
A continuación, se detallan una serie de indicaciones que puedes seguir para la elaboración de esta parte:

- Lo primero que debéis hacer es adaptar vuestro algoritmo diseñado anteriormente a problemas multi-objetivo. Los cambios irán desde modificar el código para que trabaje con no una sino dos valores de fitness por cada individuo, hasta implementar la heurística de selección de la siguiente población (en principio una de las vistas en clase, pero podéis ser creativos).
- Por supuesto podéis introducir si lo deseáis nuevas mejoras en las formas de cruce y/o mutación, se valorará positivamente.
- La implementación se llevará a cabo en un notebook de Jupyter y se ejecutará sobre Python 3. Podéis descargar un cuadernillo desde el Moodle de la asignatura, que contiene una base para el uso de la biblioteca Pymoo.
- Como sabéis, estos algoritmos son estocásticos. Por tanto, debéis lanzar al menos 10 experimentos y sacar algunos estadísticos (a elegir por el grupo) para mostrar los resultados en las presentaciones.
- Los resultados contarán de al menos una tabla donde se recojan al menos dos métricas de las vistas en teoría para comparar y evaluar la calidad de los Frentes de Pareto.
- Los resultados deben ir acompañados de una gráfica donde se muestren los paretos del algoritmo NSGA2 y del mejor experimento (o uno de los mejores) de vuestro algoritmo.
- Los presupuestos (*budget*) de los algoritmos están asociados a cada problema objetivo, que se detallan a continuación.

Las funciones a optimizar son:

3. ZDT3: Es una función con treinta variables con un frente de Pareto separable. La función se expresa matemáticamente como:

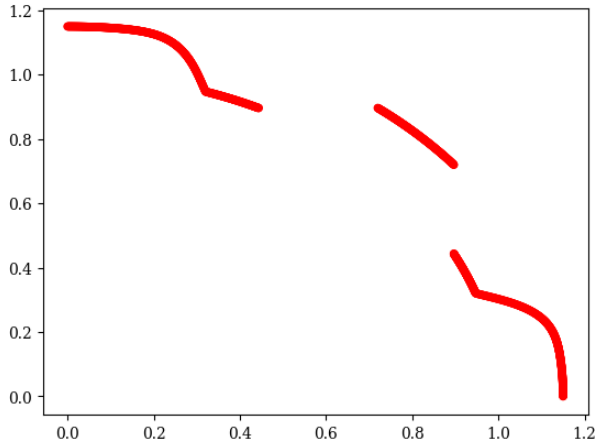
$$f_1(x) = x_1$$
$$g(x) = 1 + \frac{9}{n-1}\sum_{i=2}^{n} x_i$$
$$h(f_1, g) = 1 - \sqrt{f_1/g} - (f_1/g)\ \sin(10\pi f_1)$$
$$0 \le x_i \le 1 \quad i = 1, \ldots, n$$



Donde se trata de minimizar simultáneamente f_1 y h. El espacio de búsqueda de las variables será [0,1].
El número máximo de evaluaciones de la función de fitness será 10k.

4. MW7: Es una función con treinta variables y dos restricciones (c_1 y c_2) que dan lugar a un frente de Pareto separable. La función se expresa matemáticamente como:

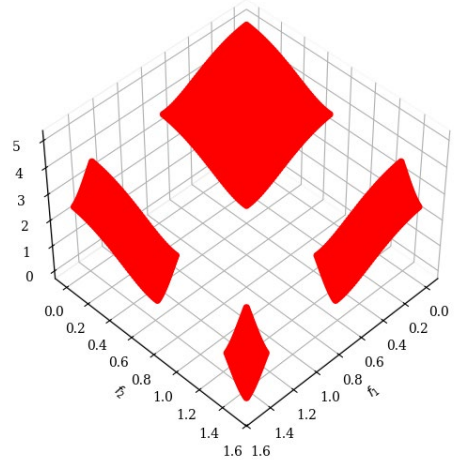
$$\min \begin{cases} f_1(\vec{x}) = g_3 x_1 \\ f_2(\vec{x}) = g_3\sqrt{1-(f_1/g_3)^2} \end{cases}$$
$$\text{s.t.} c_1(\vec{x}) = \left(1.2 + 0.4\sin(4l)^{16}\right)^2 - f_1^2 - f_2^2 \ge 0$$
$$c_2(\vec{x}) = \left(1.15 - 0.2\sin(4l)^{8}\right)^2 - f_1^2 - f_2^2 \le 0$$
$$l = \arctan(f_2/f_1).$$



El espacio de búsqueda de las variables será [0,1].
El número máximo de evaluaciones de la función de fitness será 10k.

5. MW14: Es una función para un número de funciones configurable. Nosotros la usaremos con 3 funciones e individuos de treinta variables. Da lugar a un frente de Pareto separable. La función se expresa matemáticamente como:

$$\min \begin{cases} f_{k=1:m-1}(\vec{x}) = x_k \\ f_m(\vec{x}) = g_3/(m-1)\sum_{i=1}^{m-1}\left(6 - \exp(f_i) - 1.5\sin\left(1.1\pi f_i^2\right)\right) \end{cases}$$
$$\text{s.t.} c(\vec{x}) = 1/(m-1)\sum_{i=1}^{m-1}(6.1 - \alpha(i)) - f_m \ge 0$$
$$\alpha(i) = 1 + f_i + 0.5 f_i^2 + 1.5\sin\left(1.1\pi f_i^2\right).$$



El espacio de búsqueda de las variables será [0,1.5].
El número máximo de evaluaciones de la función de fitness será 10k.

6. *Travelling Salesman Problem MO* de N=100 ciudades. El problema trata de minimizar la distancia y el tiempo que tarda un vendedor que tiene que pasar únicamente una vez por cada ciudad y volver a la ciudad inicial.

$$Min\ O1(X) = D(x_N, x_1) + \sum_{i=1}^{\mathbf{N-1}} D(x_i, x_{i+1})$$

$$Min\ O2(X) = T(x_N, x_1) + \sum_{i=1}^{\mathbf{N-1}} T(x_i, x_{i+1})$$

Para este problema se partirá de una versión adaptada a Pymoo, donde se trabaja con las matrices de tiempo y distancia (simétrica).
El número máximo de evaluaciones de la función de fitness será 100k.

Para poder realizar una comparativa entre vuestros algoritmos, sólo para el problema del TSP multiobjetivo, guardad el frente de Pareto con el siguiente formato:

- Fichero *CodGrupo_TSPMO.csv*
- Condición de parada = 100k evaluaciones
- Dos columnas:
  - La primera, llamada *objective1*
  - La segunda, llamada *objective2*

Por último, tendréis que subir a la tarea de Moodle el fichero .csv del experimento que alcanzó vuestra mejor solución.