OPTIMIZACIÓN HEURÍSTICA

Algoritmo de Evolución Diferencial

-Daniel Tomás Sánchez

-Aarón Cabero Blanco

-Pablo Bautista Frias Grupo 14

ÍNDICE

1. Introducción
2. Algoritmo de Evolución Diferencial
3. Código
4. Documentación
5. Conclusiones

1. INTRODUCCIÓN

Esta práctica consiste en la implementación de un Algoritmo Evolutivo, más concretamente el Algoritmo de Evolución Diferencial (DE).

En nuestro caso, implementaremos el Algoritmo de Evolución Diferencial (DE) con las siguientes características:

- Crossover: Se realizará una recombinación exponencial, la cual se explicará más adelante.
- Mutación: En el caso de la mutación se utilizará la estrategia *de/rand/1*, en la que se escogen tres vectores de forma aleatoria y diferentes entre sí y, posteriormente, mediante una operación aritmética, se conseguirá la mutación.

2. Algoritmo de Evolución Diferencial

El Algoritmo de Evolución Diferencial es un algoritmo de optimización que pertenece a los algoritmos evolutivos.

Dada una población con soluciones candidatas, el algoritmo se ejecutará para obtener la solución óptima mediante la recombinación y mutación de los individuos de la población.

El algoritmo funciona mediante la repetición en secuencia de los siguientes pasos durante un número concreto de iteraciones para los diferentes componentes de la población: *selección, mutación, recombinación* y *reemplazo*. Este último se realizará tras haber iterado toda la población de soluciones.

Selección: para este proceso se escogerán 3 vectores seleccionados de forma aleatoria entre toda la población. Estos vectores, que a partir de ahora denominaremos *donantes*, deberán ser diferentes entre si y, además, diferentes del vector *target* con el que vamos a trabajar.

Mutación: en nuestro caso, tal y como hemos mencionado en la introducción, la mutación se realizará mediante la estrategia *de/rand/1*, en la cual, mediante los *donantes* elegidos aleatoriamente, se realiza la operación:

$$V_{i,g} = X_{r0,g} + F(X_{r1,g} - X_{r2,g})$$

Dónde $V_{i,g}$ será el vector resultado de la mutación.

Recombinación: realizaremos una recombinación exponencial. Para ello necesitaremos el vector resultado de la mutación, el *mutante*, y el vector *target* que teníamos en la selección. La recombinación de ambos genomas dará el vector que denominaremos *candidato.* También se contará con dos variables (una entera y una real), las cuales cumplen la siguiente función:

J: nos indicará la posición inicial del vector *target* a partir de la cual tenemos que empezar a sustituir por el vector resultado de la mutación.

CR: será el criterio de parada para la recombinación. Se generará un número aleatorio que se comparará con el valor de CR y dependiendo de cuál sea mayor se seguirá o no haciendo la recombinación.

Reemplazo: para realizar el reemplazo, se compara el *fitness* de cada genoma (*target*) presente en la población actual con el *candidato* obtenido tras realizar la recombinación. En caso de que el candidato tenga un mejor *fitness* (calidad de la solución) que el *target*, este será reemplazado por el nuevo para así mejorar la población.

3. Código

Todo lo mencionado en el apartado "Algoritmo de Evolución Diferencial" se ha implementado en Python, siendo el código estructurado en distintas clases las cuales se explicarán a continuación.

EA: La clase EA es el punto de entrada para la ejecución del algoritmo, en ella se encuentran distintos atributos como:

- minfun: Será utilizado para calcular el *fitness* de los genomas.
- bounds: Contiene el mínimo y máximo valor que puede tener cada variable para ser una solución candidata.
- p_size: Es el tamaño (número de individuos) de nuestra población.
- bestGenome: Es el mejor individuo de la población.

Esta clase contiene el método "–init—"que será usado para inicializar los atributos de la clase, el método "run" el cual ejecutará el código del Algoritmo Evolutivo durante unas iteraciones concretas y devolverá una solución óptima de la población. Por último, hay un método denominado "best" que, tras acabar el método "run", devolverá el mejor individuo de toda la población, aunque puede ser llamado en cualquier instante con el fin de saber cuál es el mejor genoma presente en la población.

Genome: Nuestra clase *Genome* está compuesta por los siguientes atributos:

- array: En el cual se encuentran distintos valores, siempre dentro de los parámetros aportados por *bounds*.
- fitness: El cual determina qué tan adecuada es la solución de nuestro *Genome*.

Population: La clase *Population* crea una población de genomas. Por ello, su atributo es una lista de objetos de la clase *Genome*.

Por otro lado, esta clase contiene distintos métodos, los cuales son útiles a la hora de implementar las acciones realizadas por el algoritmo. A continuación, se hará una breve descripción de los métodos que contiene:

- ascendent_sort: Ordena los genomas que se encuentran dentro de la población de forma ascendente
- descendent_sort: Ordena los genomas que se encuentran dentro de la población de forma descendente
- add: Añade un nuevo genoma a la población
- remove: Elimina un genoma de la población
- replace: Reemplaza un genoma dado por uno de nuestra población

AbstractClasses: En esta clase se implementarán las interfaces de las clases definidas en "ImplementedClasses" que se encuentran a continuación.

ImplementedClasses: Aquí podemos encontrar cuatro clases correspondientes a las acciones que realiza el algoritmo descritas en el apartado "Algoritmo de Evolución Diferencial" (selección, mutación, recombinación y reemplazo). Cada clase contiene un método denominado *apply* que realiza la acción descrita anteriormente.

4. Documentación

Se adjunta la carpeta *docs* que contiene la documentación del código del proyecto en la subcarpeta *html* (abrir index.html).

5. Conclusiones

En primer lugar, queremos destacar la utilidad del algoritmo de evolución diferencial pues tras realizar la implementación del código hemos podido observar que los resultados son realmente buenos y la optimización que realiza el algoritmo a pesar de no ser del 100% es realmente óptima.

Por otro lado, como observación de la práctica y al ser la primera vez que se realiza esta asignatura en la Universidad Politécnica de Madrid creemos que es bastante útil. Además, nos ha parecido muy enriquecedora, no tan solo en el ámbito de la Optimización Heurística en el cuál entraremos en unos instantes, sino también a la hora de implementar código y de trabajar con distintas herramientas.

Al igual que muchos otros alumnos, era la primera vez que algunos integrantes de nuestro grupo utilizábamos Python por lo que hemos tenido que empaparnos y aprender un lenguaje nuevo, lo que nos ha resultado muy útil y nos servirá también para un futuro. Por otro lado, hemos aprendido la utilización de herramientas como Anaconda que también era desconocida para nosotros.

En cuanto a la práctica, nos ha servido para profundizar en el método de optimización de la evolución diferencial y hemos afianzado los conocimientos y puesto en práctica estos mismos. Por otro lado, nos da un poco de pena la diferencia de conocimiento en estos momentos en comparación con el algoritmo de evolución genética, pero suponemos que es algo normal y que ya depende del interés de cada alumno profundizar también en el método que no le ha tocado en la práctica.