Tarea de clase nº3

Antonio José Morano Moriña

"Definir la representación, función de fitness, operador de cruce y de mutación para el problema de la mochila. Se dispone una mochila y un conjunto de n objetos, cada uno de los cuales tiene un volumen positivo y un beneficio. El objetivo el conjunto de objetos con volumen menor a la capacidad de la mochila y mayor beneficio. Prestar atención a los individuos inválidos en el cruce y la mutación. Se puede describir con pseudocódigo, funciones matemáticas o una combinación de ellas, pero tiene que ser claro y conciso. Pon un ejemplo con un par de cromosomas y realice todas las operaciones incluida la evaluación."

1 - Representación:

Podemos representar las soluciones como cromosomas binarios de longitud n, donde n es el número de objetos. Cada gen del cromosoma representa si el objeto correspondiente está o no en la mochila. (Si 1, está en la mochila. Si 0, no está en la mochila)

```python
# Representación de los individuos
def representacion():
    return np.random.randint(0, 2, size=len(objetos))
```

2 - Función de fitness:

Evaluamos el beneficio total de los objetos seleccionados en la mochila. Para ello, multiplicamos el beneficio de cada objeto por su correspondiente gen en el cromosoma, y sumamos los resultados. Si la suma es mayor que la capacidad de la mochila, la solución es inválida y devuelve 0.

```python
# Función de fitness
def fitness(individuo):
    # volumen y el beneficio de los objetos del individuo
    volumen = np.sum(objetos[:, 0] * individuo)
    beneficio = np.sum(objetos[:, 1] * individuo)
    # Si volumen excede, fitness nulo
    if volumen > capacidad_mochila:
        return 0

    return beneficio
```

3 - Operador de cruce:

Para el operador de cruce, se elige un punto aleatorio en el cromosoma y se intercambian las secciones antes y después del punto entre dos padres seleccionados para producir dos hijos. Al igual que en la práctica, pero con un solo punto de corte entre los padres.

```python
# Operador de cruce
def cruce(p1, p2):
    # Elegimos un punto de cruce al azar
    punto_cruce = np.random.randint(len(p1))
    # Cruzamos, generando los hijos
    hijo1 = np.concatenate((p1[:punto_cruce], p2[punto_cruce:]))
    hijo2 = np.concatenate((p2[:punto_cruce], p1[punto_cruce:]))
    # Comprobamos validez individuos
    hijo1 = corregir_individuo(hijo1)
    hijo2 = corregir_individuo(hijo2)
    return hijo1, hijo2
```

4 - Operador de mutación:

Para el operador de mutación, se elige un gen aleatorio en el cromosoma y se invierte su valor. Verificamos que la solución resultante sigue siendo válida. Si no lo es, debemos repararla de la misma manera que en el operador de cruce.

```
# Operador de mutación
def mutacion(individuo):
    # mutamos gen al azar
    gen_mutado = np.random.randint(len(individuo))
    # Mutamos el gen, cambiando su valor a 0 o 1 al
    individuo[gen_mutado] = np.random.randint(0, 2)
    individuo = corregir_individuo(individuo)
    return individuo
```

5- Controlar individuos

Para poder realizar una iteración del problema representado, he creado una función extra que se encarga de comprobar que los individuos no tomen valores fuera de la lógica de la representación del problema.

```
#garantiza que el volumen de un individuo no exceda la capacidad de la mochila
def corregir_individuo(individuo):
    while np.sum(objetos[:, 0] * individuo) > capacidad_mochila:
        #Si excede, quitamos el objeto mas pesado
        objeto_max_volumen = np.argmax(objetos[:, 0] * individuo)
        individuo[objeto_max_volumen] = 0
    return individuo
```

EJEMPLOS:

En cada iteración del algoritmo, se comprueban cada uno de los 3 operadores implementados.

Cada uno de los individuos generados para comprobar los operadores, ha sido creado con la función representación (), por lo que se generan individuos de forma aleatoria en cada iteración. Por lo que los valores de los individuos usados en cada operador no están relacionados.

Los datos usados para el problema son:

```
capacidad_mochila = 50
objetos = np.array([[10, 60], [20, 100], [30, 120], [40, 150]])
```

EJEMPLO 1

```
Resultado del operador de cruce:
Padre 1: [0 0 1 0]
Padre 2: [1 1 0 0]
Hijo 1: [0 0 0 0]
Hijo 2: [1 1 0 0]
Resultado del operador de mutacion
Antes de mutar ->  [1 1 1 0]
Despues de mutar ->  [1 1 0 0]
Resultados de la funcion fitness
Solucion a evaluar:  [1 1 0 0]
fitness equivalente a la solucion:  160
```

En esta iteración podemos ver como realiza el cruce, y cambia uno de los hijos, ya que al comprobar la validez del individuo, la función corregir_individuo() ha tenido que cambiarlo para que no se salga de la lógica de las representación de las soluciones.

El de cruce muta una de las posiciones. El fitness evalúa el individuo correctamente

EJEMPLO 2

```
Resultado del operador de cruce:
Padre 1: [0 1 0 1]
Padre 2: [1 1 1 0]
Hijo 1: [0 1 0 0]
Hijo 2: [1 1 0 0]
Resultado del operador de mutacion
Antes de mutar ->  [1 1 0 0]
Despues de mutar ->  [1 1 0 0]
Resultados de la funcion fitness
Solucion a evaluar:  [1 1 1 1]
fitness equivalente a la solucion:  0
```

El operador de cruce actúa igual que en el ejemplo anterior.

El operador de mutación, falla al mutar debido a que habrá pasado por corregir_individuo() y habrá realizado un cambio para que el individuo mutado no este fuera de la lógica de las representaciones.

El operador fitness devuelve 0, ya que se excede la capacidad de la mochila.