Computación Neuronal y Evolutiva

Informe de prácticas

Práctica 06: Optimización Multi-Objetivo

Alumnos:

1.- Raúl Negro Carpintero

2.- Mario Núñez Izquierdo

Tabla de contenido

DESCRIPCIÓN

Nota: todas las pruebas descritas en este documento están realizadas usando el fichero "me_at_the_zoo.csv".

Restricciones

En este caso, hemos configurado los individuos y la población en el archivo *principal.py*. Además, en este fichero configuramos los tres experimentos que vamos a evaluar (cruce, mutación, selección y probabilidades), y mostramos los resultados obtenidos mediante gráficas.

```
def configuracionIndividuos():
    global toolbox
    toolbox = base.Toolbox()
    global pop

    tamano = entrada_salida.n_videos * entrada_salida.n_caches

    creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
    creator.create("Individual", list, fitness=creator.FitnessMin)

    toolbox.register("attribute", random.randint, 0, 1)
    toolbox.register("individual", tools.initRepeat,
                     creator.Individual, toolbox.attribute, n=tamano)
    toolbox.register("population",
                     tools.initRepeat, list, toolbox.individual)

    pop = toolbox.population(n = 1000)
```

Ilustración 1 Configuración de los individuos.

```
def configuracionAlgoritmo_Experimento1():
    toolbox.register("mate", tools.cxOnePoint)
    toolbox.register("mutate", tools.mutFlipBit, indpb=0.2)
    toolbox.register("select", tools.selTournament, tournsize=3)
    toolbox.register("evaluate", evaluate)
    toolbox.decorate("evaluate", tools.DeltaPenalty(feasible, 1000000, distance))
```

Ilustración 2 Ejemplo de configuración de experimento.

En el fichero *evaluacion.py* implementamos la nueva función de evaluación (prácticamente igual que en las otras prácticas), y las nuevas funciones de factibilidad y distancia para hacer uso de las restricciones.

```python
def feasible(individual):
    global sobrepeso
    global indi2
    flag = True
    indi = np.array(individual)
    indi2 = indi.reshape(entrada_salida.n_caches, entrada_salida.n_videos)
    tamanos = np.dot(indi2, entrada_salida.tamano_videos)
    sobrepeso = 0

    for i in tamanos:
        if i > entrada_salida.tamano_caches:
            sobrepeso += i - entrada_salida.tamano_caches
            flag = False

    return flag
```

Ilustración 3 Función de factibilidad.

En la función *feasible*, comprobamos si nos hemos pasado de peso en la cache y devolvemos True si no nos hemos pasado (es factible) o False en caso contrario (no es factible).

```python
def distance(individual):
    return sobrepeso * 100000
```

Ilustración 4 Función de distancia.

En la función *distance*, penalizamos los individuos que no sean factibles.

También hemos hechos cambios menores en el módulo *entrada_salida.py*, ya que en las anteriores prácticas no cogíamos del todo bien los datos.

Multi-objetivo

En este caso hemos dividido el programa de la misma manera. En *principal.py* inicializamos los individuos y configuramos los experimentos:

```python
def configuracionIndividuos():
    global toolbox
    toolbox = base.Toolbox()
    global pop

    tamano = entrada_salida.n_videos * entrada_salida.n_caches

    creator.create("FitnessMin", base.Fitness, weights=(-1.0,-1.0))
    creator.create("Individual", list, fitness=creator.FitnessMin)

    toolbox.register("attribute", random.randint, 0, 1)
    toolbox.register("individual", tools.initRepeat,
                     creator.Individual, toolbox.attribute, n=tamano)
    toolbox.register("population",
                     tools.initRepeat, list, toolbox.individual)

    pop = toolbox.population(n = 200)

def configuracionAlgoritmo_Experimento1():
    toolbox.register("mate", tools.cxUniform, indpb=0.2)
    toolbox.register("mutate", tools.mutFlipBit, indpb=0.2)
    toolbox.register("select", tools.selSPEA2)
    toolbox.register("evaluate", evaluate)
```

Ilustración 5 Inicialización de individuos y de configuración del multi-objetivo.

Otra cosa a destacar en este fichero (y que lo diferencia del anterior) es que no podemos usar el eaSimple, hemos tenido que utilizar eaMuPlusLambda.

```python
population1, logbook1 = algorithms.eaMuPlusLambda(pop, toolbox, mu=160, lambda_=300, cxpb=0.5, mutpb=0.2, ngen=100, stats=stats)
```

Ilustración 6 Estructura del eaMuPlusLambda.

En *evaluacion.py* implementamos la nueva función de evaluación. Esta función está basada en la que utilizamos en prácticas anteriores, pero con la diferencia de que ahora retorna dos valores en vez de uno.

```python
def evaluate(individual):
    sobrepeso = 0
    indi = np.array(individual)
    indi2 = indi.reshape(entrada_salida.n_caches, entrada_salida.n_videos)
    tamanos = np.dot(indi2, entrada_salida.tamano_videos)
    score = 0
    peticiones_totales = 0
    latencia_minima = 0
    caches_usadas = list()

    for i in tamanos:
        if i > entrada_salida.tamano_caches:
            sobrepeso += i - entrada_salida.tamano_caches

    for i in range(entrada_salida.n_caches):
        caches_usadas.append(list())
        for j in range(entrada_salida.n_videos):
            caches_usadas[i].append(0)

    for i in range(entrada_salida.endpoints):
        for j in range(entrada_salida.n_videos):
            latencia_minima = 0
            if entrada_salida.peticiones[i][j] is not 0:
                peticiones_totales += entrada_salida.peticiones[i][j]
                latencia_minima = entrada_salida.latencias[i][-1]
                for cache in entrada_salida.latencias[i]:
                    if cache is not -1 and indi2[i][j] == 1 and entrada_salida.latencias[i][cache] < latencia_minima:
                        caches_usadas[cache][j] = 1
                        latencia_minima = entrada_salida.latencias[i][cache]
                score += entrada_salida.peticiones[i][j] * latencia_minima

    score /= peticiones_totales

    for i in range(entrada_salida.n_caches):
        for j in range(entrada_salida.n_videos):
            if caches_usadas[i][j] == 0 and indi2[i][j] == 1:
                score += 10000

    return score,sobrepeso
```

Ilustración 7 Función de evaluación del multi-objetivo.

El módulo *entrada_salida.py* es el mismo que en el caso anterior.

COMPARATIVA

Restricciones

La configuración de los experimentos es la siguiente:

```python
def configuracionAlgoritmo_Experimento1():
    toolbox.register("mate", tools.cxOnePoint)
    toolbox.register("mutate", tools.mutFlipBit, indpb=0.2)
    toolbox.register("select", tools.selTournament, tournsize=3)
    toolbox.register("evaluate", evaluate)
    toolbox.decorate("evaluate", tools.DeltaPenalty(feasible, 1000000, distance))
```

Ilustración 8 Configuración del 1er experimento.

```python
def configuracionAlgoritmo_Experimento2():
    toolbox.register("mate", tools.cxOnePoint)
    toolbox.register("mutate", tools.mutFlipBit, indpb=0.2)
    toolbox.register("select", tools.selBest)
    toolbox.register("evaluate", evaluate)
    toolbox.decorate("evaluate", tools.DeltaPenalty(feasible, 1000000, distance))
```

Ilustración 9 Configuración del 2º experimento.

```
def configuracionAlgoritmo_Experimento3():
    toolbox.register("mate", tools.cxUniform, indpb=0.2)
    toolbox.register("mutate", tools.mutFlipBit, indpb=0.2)
    toolbox.register("select", tools.selTournament, tournsize=3)
    toolbox.register("evaluate", evaluate)
    toolbox.decorate("evaluate", tools.DeltaPenalty(feasible, 1000000, distance))
```

Ilustración 10 Configuración del 3er experimento.

Hemos ejecutado el algoritmo dos veces para comprobar la tendencia de cada experimento, la primera vez ejecutamos con 100 individuos y 100 generaciones y la segunda vez ejecutamos con 1000 individuos y 100 generaciones. Obtuvimos los siguientes resultados:

```
Su fitness es:
(791200000.0,)
```

Ilustración 11 Fitness del 1er experimento (100 individuos 100 gens).

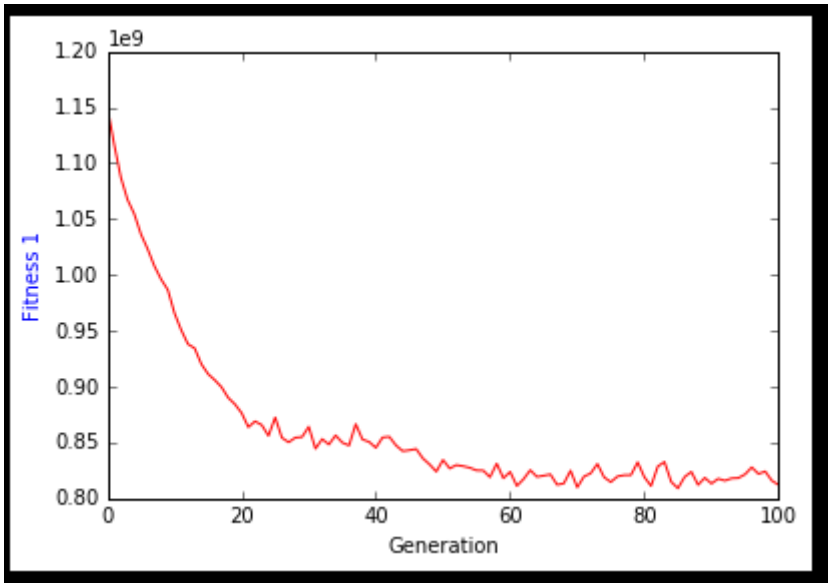


Ilustración 12 Gráfica del 1er experimento (100 individuos 100 gens).

```
Su fitness es:
(425300000.0,)
```

Ilustración 13 Fitness del 1er experimento (1000 individuos 100 gens).

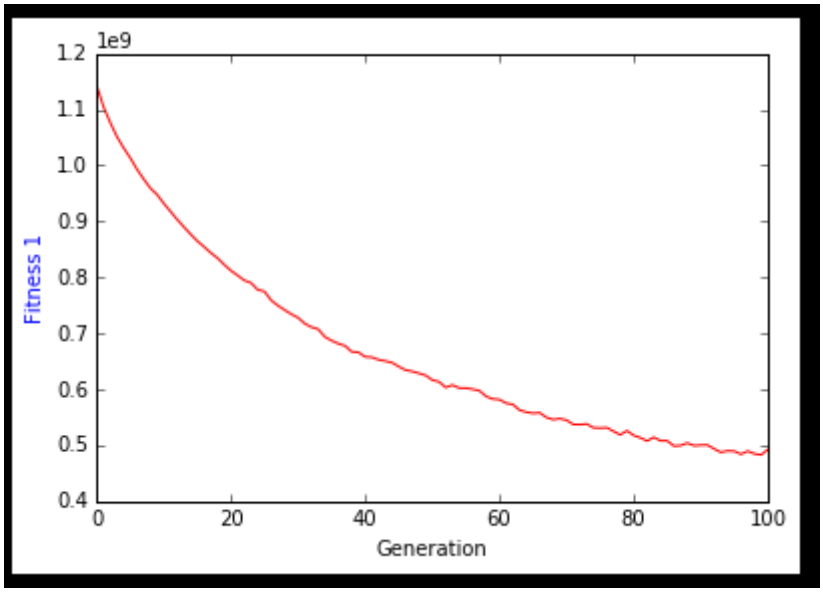


Ilustración 14 Gráfica del 1er experimento (1000 individuos 100 gens).

Como podemos observar, el número de individuos repercute severamente en el resultado, pero aun con todo, la solución que obtenemos es bastante mala y corresponde a un individuo no apto (no respeta el tamaño de las caches).

```
Su fitness es:
(975000000.0,)
```

Ilustración 15 Fitness del 2º experimento (100 individuos 100 gens).

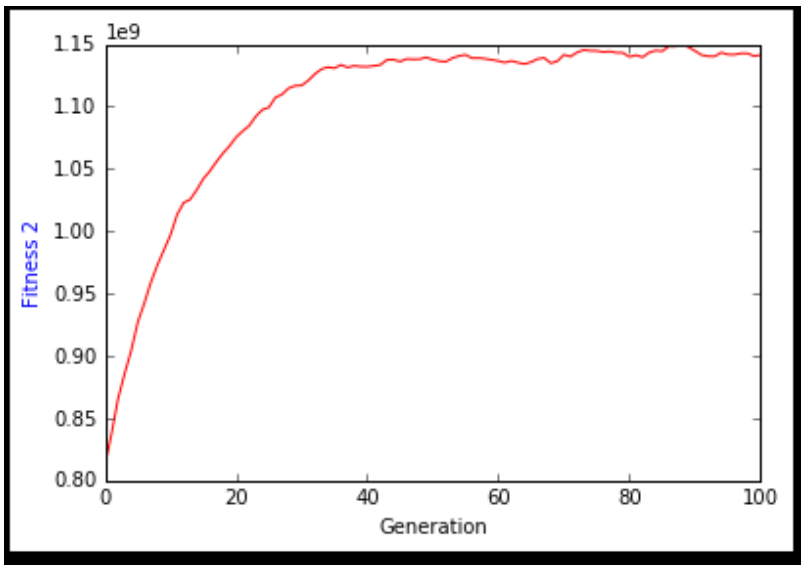


Ilustración 16 Gráfica del 2º experimento (100 individuos 100 gens).

```
Su fitness es:
(906300000.0,)
```

Ilustración 17 Fitness del 2º experimento (1000 individuos 100 gens).

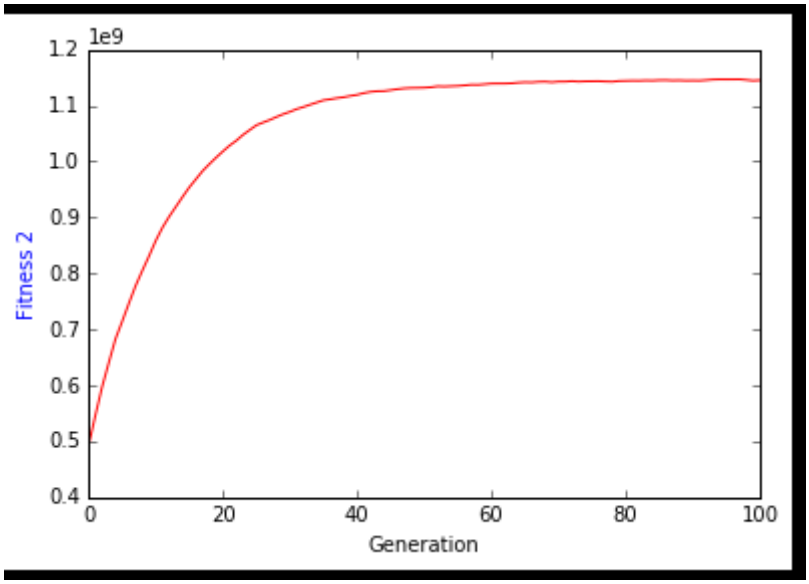


Ilustración 18 Gráfica del 2º experimento (1000 individuos 100 gens).

En este caso, el número de individuos no ha cambiado mucho el resultado. Además, este método de selección no esta funcionando correctamente ya que el fitness esta aumentando y nosotros buscamos disminuirlo.

El único parámetro distinto entre los dos experimentos es el método de selección, siendo selección por torneo en el primero y selección del mejor en el segundo, por lo que podemos asegurar que éste es el causante de las diferencias en los resultados.

```
Su fitness es:
(504800000.0,)
```

Ilustración 19 Fitness del 3er experimento (100 individuos 100 gens).

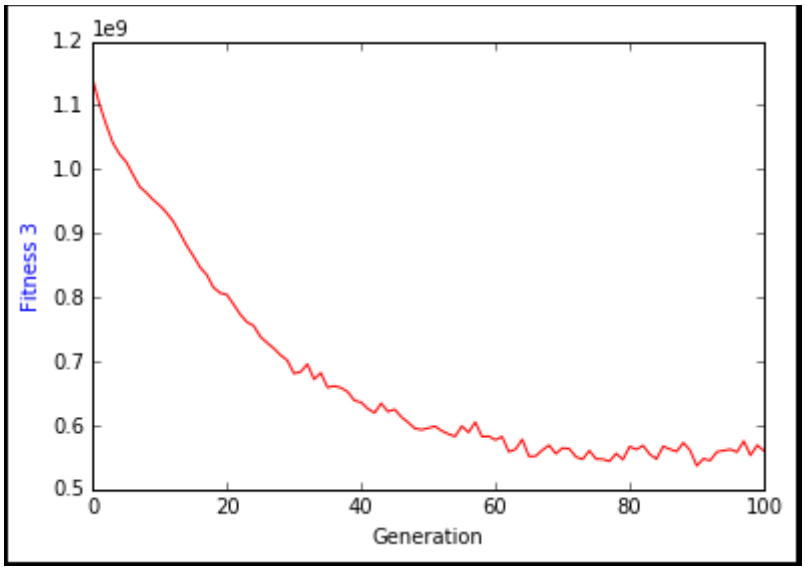


Ilustración 20 Gráfica del 3er experimento (100 individuos 100 gens).

```
Su fitness es:
(860564.9029683331,)
```

Ilustración 21 Fitness del 3er experimento (1000 individuos 100 gens).

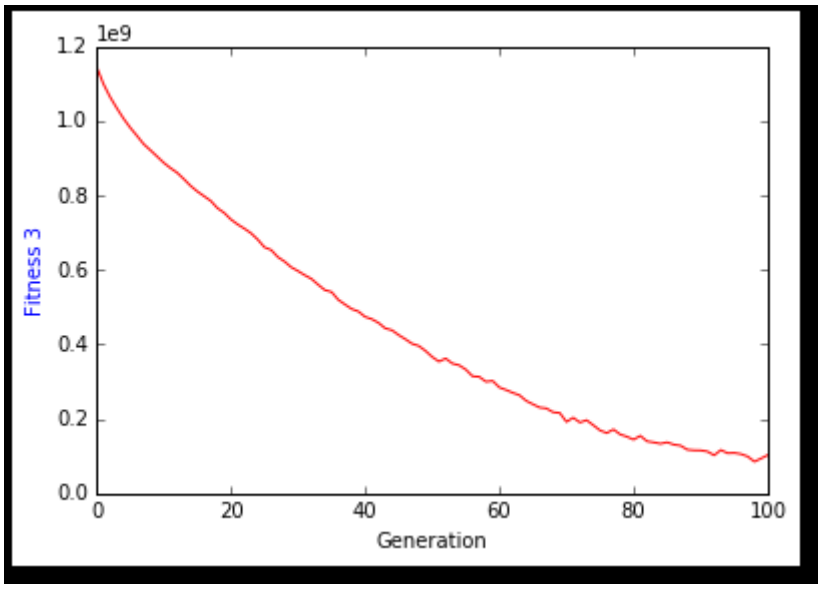


Ilustración 22 Gráfica del 3er experimento (1000 individuos 100 gens).

En este caso, hemos conseguido obtener una solución válida, 860564. Como hemos visto en la configuración de los algoritmos, como mínimo, un individuo no apto tiene 1 millón de fitness por defecto y este valor se ve incrementado aún más por la función distance. Como el fitness obtenido es menor que 1 millón, deducimos que este fitness pertenece a un individuo valido. Sin embargo, no parece ser la más óptima ya que la gráfica no llega a estabilizarse, esto es debido a que 100 generaciones no son suficientes para hallar una solución óptima.

Con estos resultados, podemos deducir que el peor método de selección es el de selección del mejor individuo, y que hay una mejora importante entre el cruce de un punto y el cruce uniforme (el mejor es el uniforme).

Multi-objetivo

De forma similar al algoritmo anterior, tenemos 3 experimentos preparados y los ejecutaremos 2 veces cada uno. La primera con 100 individuos y 100 generaciones y la segunda con 1000 individuos y 100 generaciones.

```python
def configuracionAlgoritmo_Experimento1():
    toolbox.register("mate", tools.cxUniform, indpb=0.2)
    toolbox.register("mutate", tools.mutFlipBit, indpb=0.2)
    toolbox.register("select", tools.selSPEA2)
    toolbox.register("evaluate", evaluate)

def configuracionAlgoritmo_Experimento2():
    toolbox.register("mate", tools.cxOnePoint)
    toolbox.register("mutate", tools.mutFlipBit, indpb=0.2)
    toolbox.register("select", tools.selNSGA2)
    toolbox.register("evaluate", evaluate)

def configuracionAlgoritmo_Experimento3():
    toolbox.register("mate", tools.cxOnePoint)
    toolbox.register("mutate", tools.mutFlipBit, indpb=0.2)
    toolbox.register("select", tools.selSPEA2)
    toolbox.register("evaluate", evaluate)
```

La configuración del eaMuPlusLambda es la siguientes para ambas ejecuciones:

```python
eaMuPlusLambda(pop, toolbox, mu=80, lambda_=150, cxpb=0.5, mutpb=0.2, ngen=100, stats=stats)
```

Ilustración 23Configuracion eaMuPlusLambda

Estos son los resultados obtenidos:

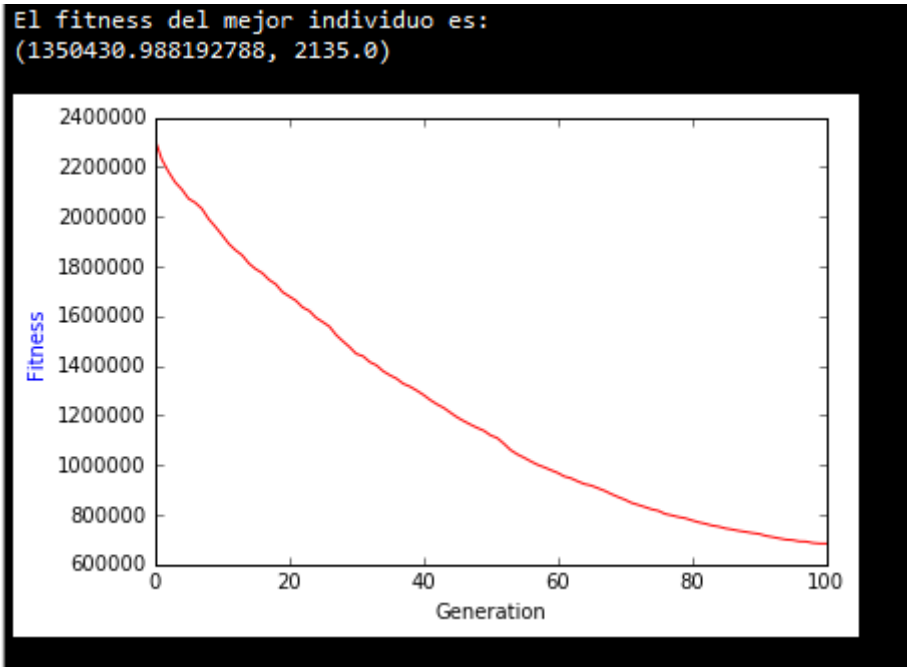


Ilustración 24 Resultados del primer experimento MultiObjetivo (100 individuos 100 gens)

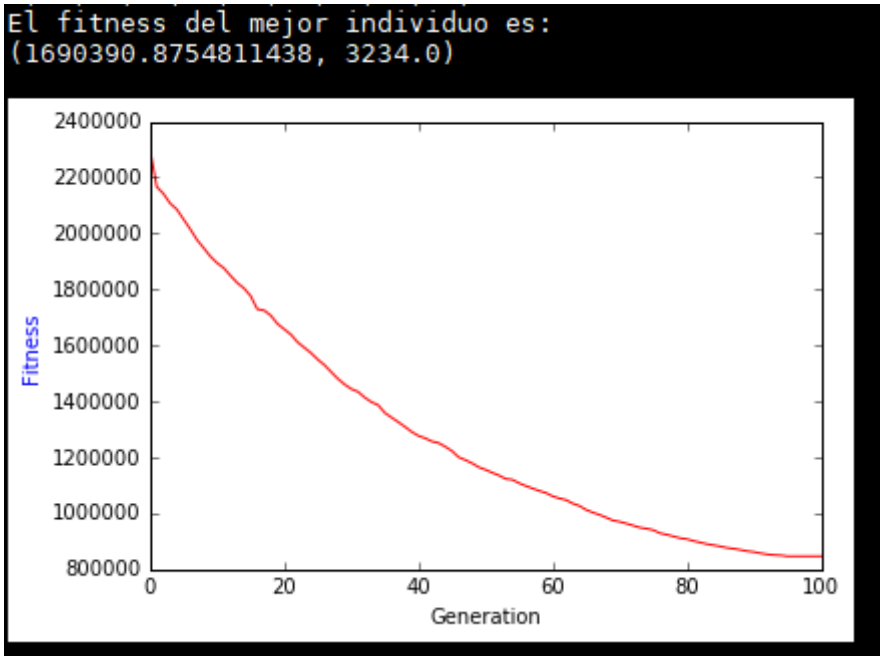


Ilustración 25 Resultados del primer experimento MultiObjetivo (1000 individuos 100 gens)

Como podemos observar, el algoritmo no es capaz de encontrar una solución válida al problema, ya que el segundo fitness nos indica que el mejor individuo obtenido tiene alrededor de 2000 y 3000 megas que sobrepasan el tamaño de las caches.

Además, en este caso nos damos cuenta de que aumentar el número de individuos no ha mejorado la solución si no que, al contrario, la ha empeorado. Esto se debe principalmente a la configuración del eaMuPlusLambda, en ambos casos está configurado para sustituir solo 80 individuos de la población y generar 150 sucesores. Estos datos parecen ser más adecuados para ejecutar con 100 individuos que con 1000.

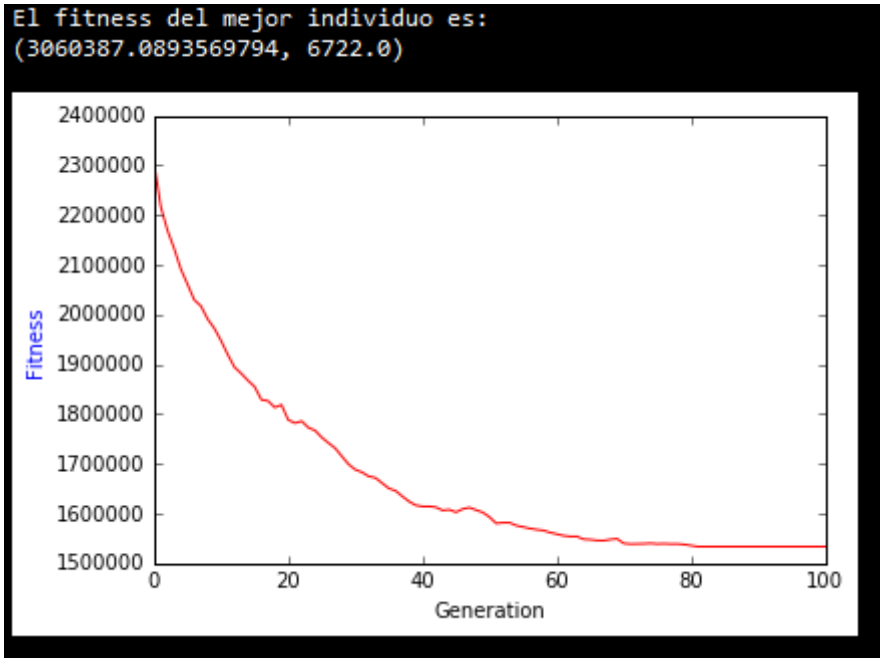


Ilustración 26 Resultados del segundo experimento MultiObjetivo (100 individuos 100 gens)

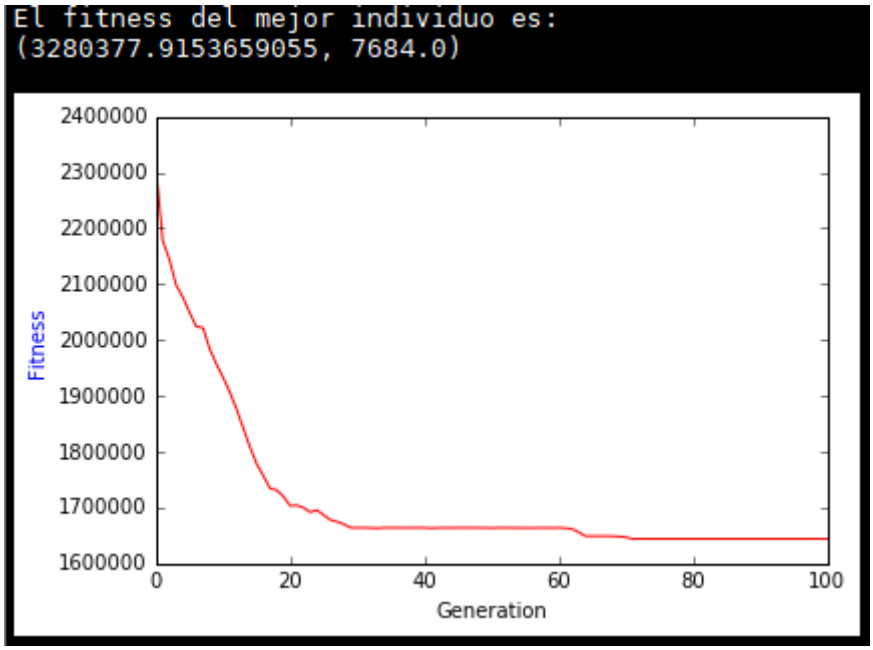


Ilustración 27 Resultados del segundo experimento MultiObjetivo (1000 individuos 100 gens)

En este segundo experimento vemos que los resultados son peores que en el anterior. No obstante, el método de selección nuevo cumple con su función y disminuye el fitness. Y una vez más, en este caso el aumento del número de individuos también ha afectado negativamente a la solución.

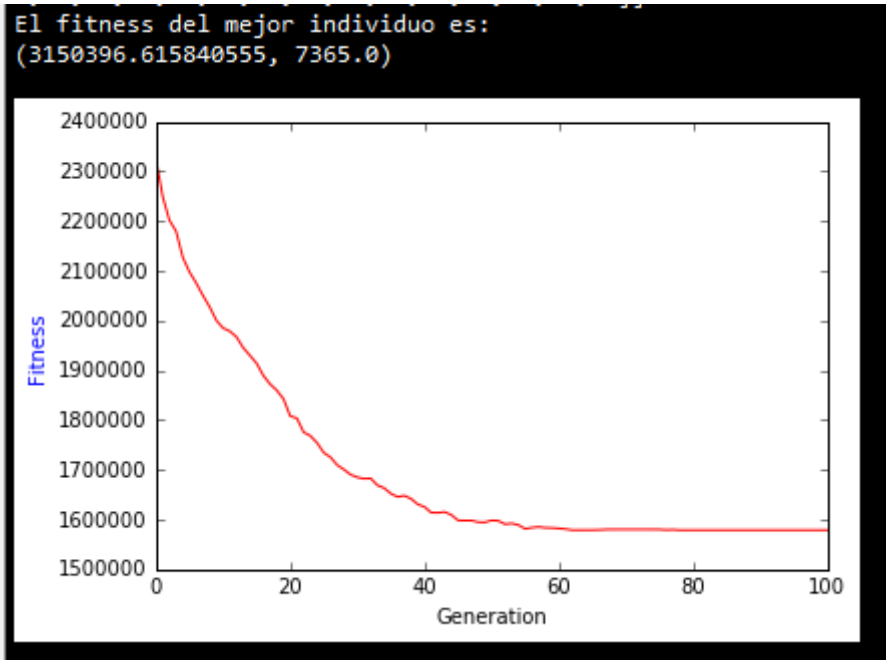


Ilustración 28 Resultados del tercer experimento MultiObjetivo (100 individuos 100 gens)

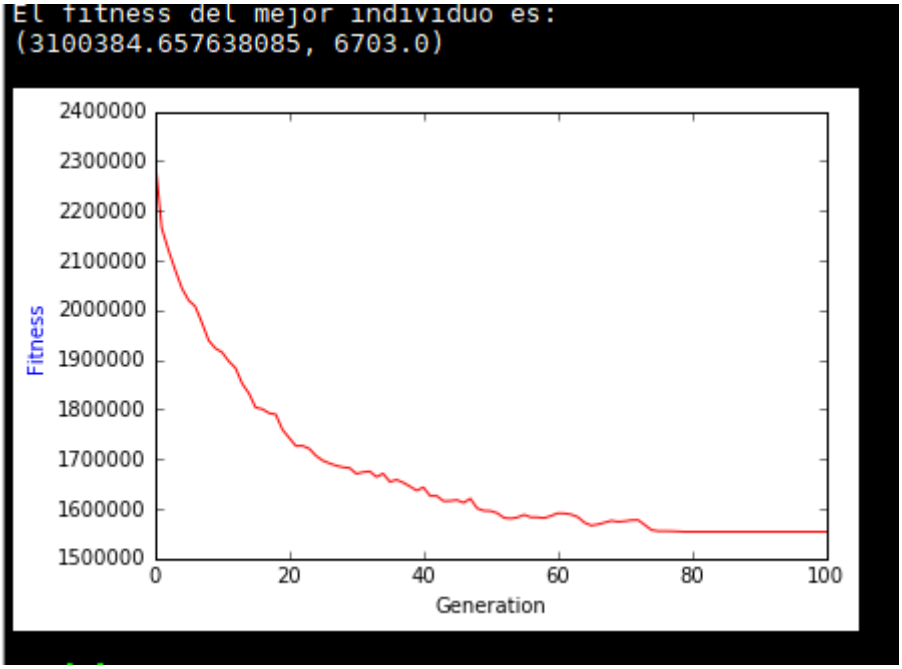


Ilustración 29 Resultados del tercer experimento MultiObjetivo (1000 individuos 100 gens)

En el último de los experimentos volvemos a encontrar unos valores inferiores a los obtenidos en el primero. Lo destacable de este experimento, es que el aumento de individuos ha mejorado la solución, al contrario que en los otros dos.

De los resultados obtenidos en los tres experimentos, el mejor ha sido el del primer experimento (con 100 individuos). De una forma similar al problema de las restricciones, la configuración ganadora es: cxUniform, mutFlipBit y selSPEA2. La forma de selección debe

cambiar estrictamente, pero los métodos de cruce y mutación permanecen iguales. Y de forma similar también al problema anterior, parece que aumentando el número de generaciones la solución podría seguir mejorando.

MEJOR SOLUCIÓN

Teniendo en cuenta los resultados de la comparación y realizando un par de pruebas para encontrar los valores adecuados, hemos realizado una última prueba más ambiciosa con el fin de obtener la mejor solución posible en un lapso razonable de tiempo (estas pruebas han durado unos 15-30min)

Restricciones

La configuración que hemos usado es la del mejor experimento de la comparativa:

```
def configuracionAlgoritmo_Experimento3():
    toolbox.register("mate", tools.cxUniform, indpb=0.2)
    toolbox.register("mutate", tools.mutFlipBit, indpb=0.2)
    toolbox.register("select", tools.selTournament, tournsize=3)
    toolbox.register("evaluate", evaluate)
    toolbox.decorate("evaluate", tools.DeltaPenalty(feasible, 1000000, distance))
```

Ilustración 30 Configuración de la mejor solución Restricciones

El número de individuos es 10.000 y hemos usado 200 generaciones. El mejor individuo devuelto por el algoritmo es el siguiente:

```
La mejor solucion encontrada es:
[[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

Ilustración 31 Mejor solución Restricciones

```
Su fitness es:
(518.2542092710228,)
```

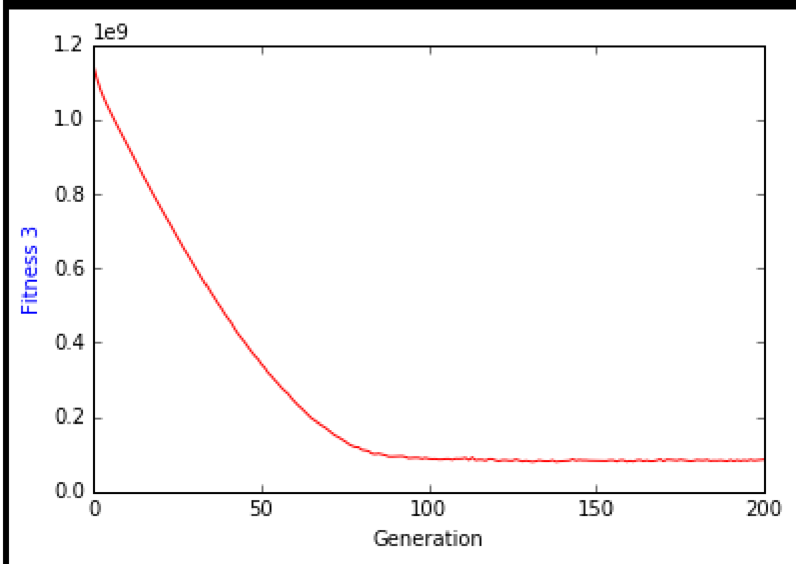


Ilustración 32 Mejor solución Restricciones

Como podemos observar, este individuo es válido (su fitness es menor que 1 millón) y parece ser bastante óptimo, aunque no tenemos forma de comprobarlo ya que el fichero

“me_at_the_zoo.csv” es lo suficientemente complejo como para no poder calcular la solución más óptima a mano. Además, podemos afirmar que no carga videos no utilizados en las caches, si uno de los videos de una cache no tiene peticiones, penalizamos el fitness del algoritmo con +10.000.

Teniendo en cuenta las anteriores consideraciones, pensamos que esta es una buena solución, aunque probablemente no la mejor.

Multi-objetivo

En esta prueba configuraremos el algoritmo del siguiente modo. Utilizaremos el mejor experimento de la comparación:

```
    pop = toolbox.population(n = 500)

def configuracionAlgoritmo_Experimento1():
    toolbox.register("mate", tools.cxUniform, indpb=0.2)
    toolbox.register("mutate", tools.mutFlipBit, indpb=0.2)
    toolbox.register("select", tools.selSPEA2)
    toolbox.register("evaluate", evaluate)
```

Ilustración 33 Configuración mejor solución Multi-Objetivo

El número de individuos es 500 (como se aprecia en la foto). La configuración de eaMuPlusLambda es la siguiente:

```
eaMuPlusLambda(pop, toolbox, mu=400, lambda_=500, cxpb=0.5, mutpb=0.2, ngen=150, stats=stats)
```

Ilustración 34 Configuración mejor solución Multi-Objetivo

Por lo que habrá 500 generaciones. Los resultados son los siguientes:

```
La mejor solucion encontrada es:
[[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]]
```

Ilustración 35 Mejor solución Multi-Objetivo

```
El fitness del mejor individuo es:
(80479.59616974049, 9.0)
```

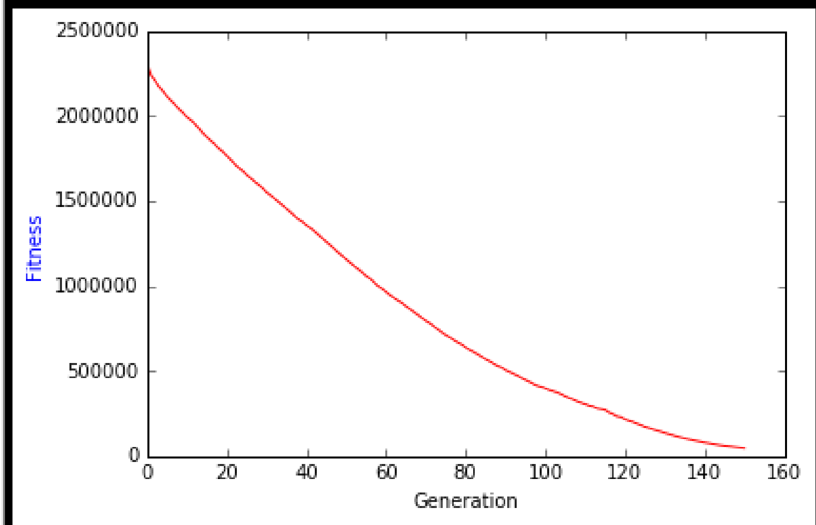
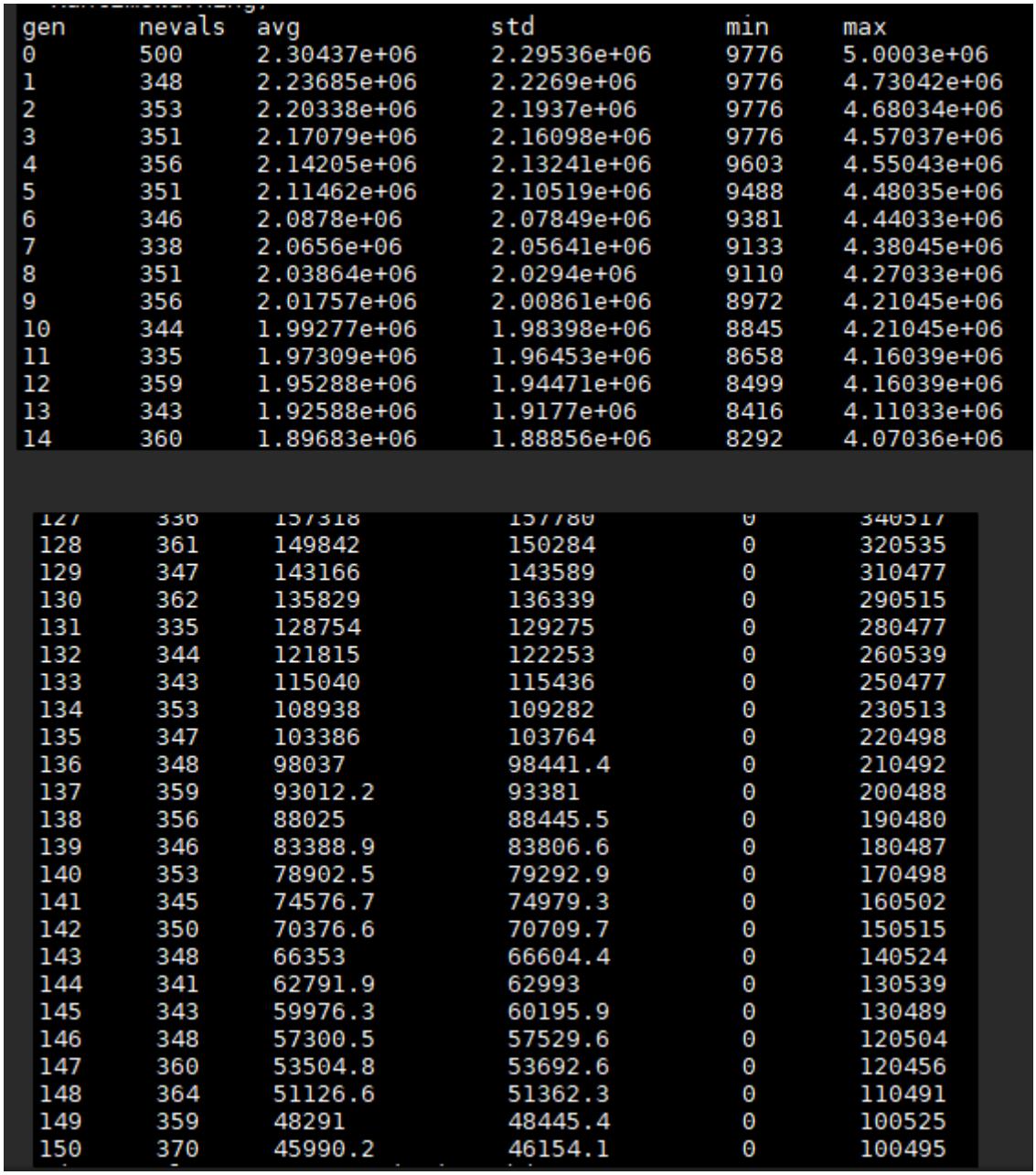


Ilustración 36 Mejor solución Multi-Objetivo

Como podemos observar, la solución ha mejorado mucho, hasta el punto de que casi es una solución válida. Es posible que hayamos recuperado mal el mejor individuo, ya que nosotros lo imprimimos mediante la línea “print(tools.selBest(population1, 1))” (creemos que esta función no es la adecuada para un problema multi-objetivo) y obtenemos el resultado de arriba. Pero en las últimas generaciones, el mínimo estadístico indica que hay un mejor individuo:



gen	nevals	avg	std	min	max
0	500	2.30437e+06	2.29536e+06	9776	5.0003e+06
1	348	2.23685e+06	2.2269e+06	9776	4.73042e+06
2	353	2.20338e+06	2.1937e+06	9776	4.68034e+06
3	351	2.17079e+06	2.16098e+06	9776	4.57037e+06
4	356	2.14205e+06	2.13241e+06	9603	4.55043e+06
5	351	2.11462e+06	2.10519e+06	9488	4.48035e+06
6	346	2.0878e+06	2.07849e+06	9381	4.44033e+06
7	338	2.0656e+06	2.05641e+06	9133	4.38045e+06
8	351	2.03864e+06	2.0294e+06	9110	4.27033e+06
9	356	2.01757e+06	2.00861e+06	8972	4.21045e+06
10	344	1.99277e+06	1.98398e+06	8845	4.21045e+06
11	335	1.97309e+06	1.96453e+06	8658	4.16039e+06
12	359	1.95288e+06	1.94471e+06	8499	4.16039e+06
13	343	1.92588e+06	1.9177e+06	8416	4.11033e+06
14	360	1.89683e+06	1.88856e+06	8292	4.07036e+06
127	336	157318	157780	0	340517
128	361	149842	150284	0	320535
129	347	143166	143589	0	310477
130	362	135829	136339	0	290515
131	335	128754	129275	0	280477
132	344	121815	122253	0	260539
133	343	115040	115436	0	250477
134	353	108938	109282	0	230513
135	347	103386	103764	0	220498
136	348	98037	98441.4	0	210492
137	359	93012.2	93381	0	200488
138	356	88025	88445.5	0	190480
139	346	83388.9	83806.6	0	180487
140	353	78902.5	79292.9	0	170498
141	345	74576.7	74979.3	0	160502
142	350	70376.6	70709.7	0	150515
143	348	66353	66604.4	0	140524
144	341	62791.9	62993	0	130539
145	343	59976.3	60195.9	0	130489
146	348	57300.5	57529.6	0	120504
147	360	53504.8	53692.6	0	120456
148	364	51126.6	51362.3	0	110491
149	359	48291	48445.4	0	100525
150	370	45990.2	46154.1	0	100495

Ilustración 37 Generaciones mejor solución Multi-Objetivo

En esta imagen podemos observar que las estadísticas nos indican que hay un individuo cuyo fitness mínimo (referido al tamaño extra de los videos) es 0, por lo que dicho individuo seria valido.

CONCLUSIÓN

Restricciones

Si comparamos el mejor resultado de esta práctica con el mejor resultado de la práctica 3, podemos observar que los resultados han mejorado mucho.

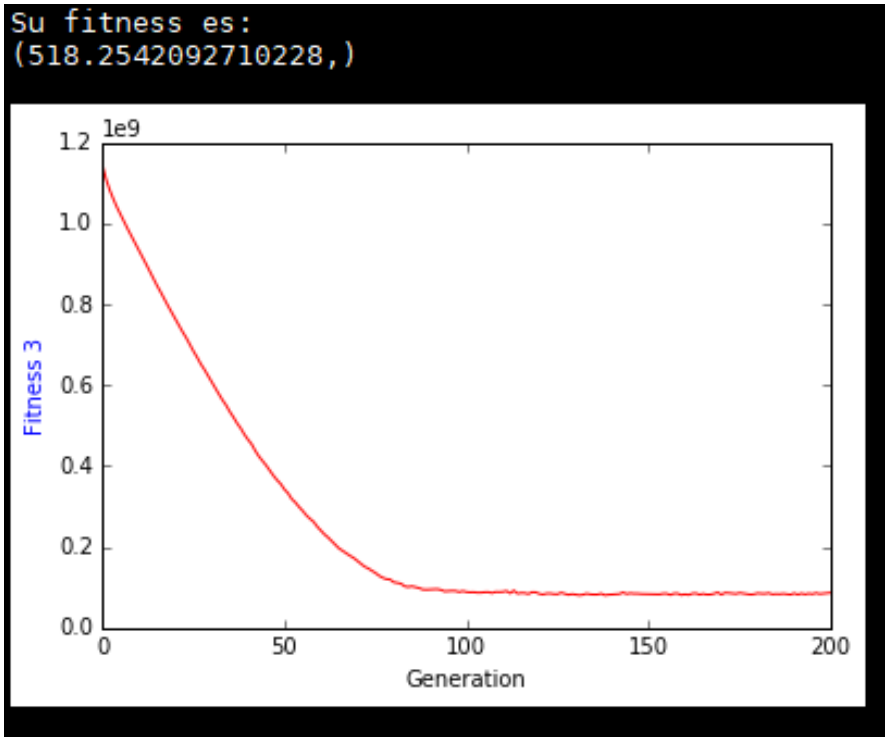


Ilustración 38 Mejor solución restricciones

```
(548200000.0,)
```

Ilustración 39 Fitness del mejor resultado de la práctica 3.

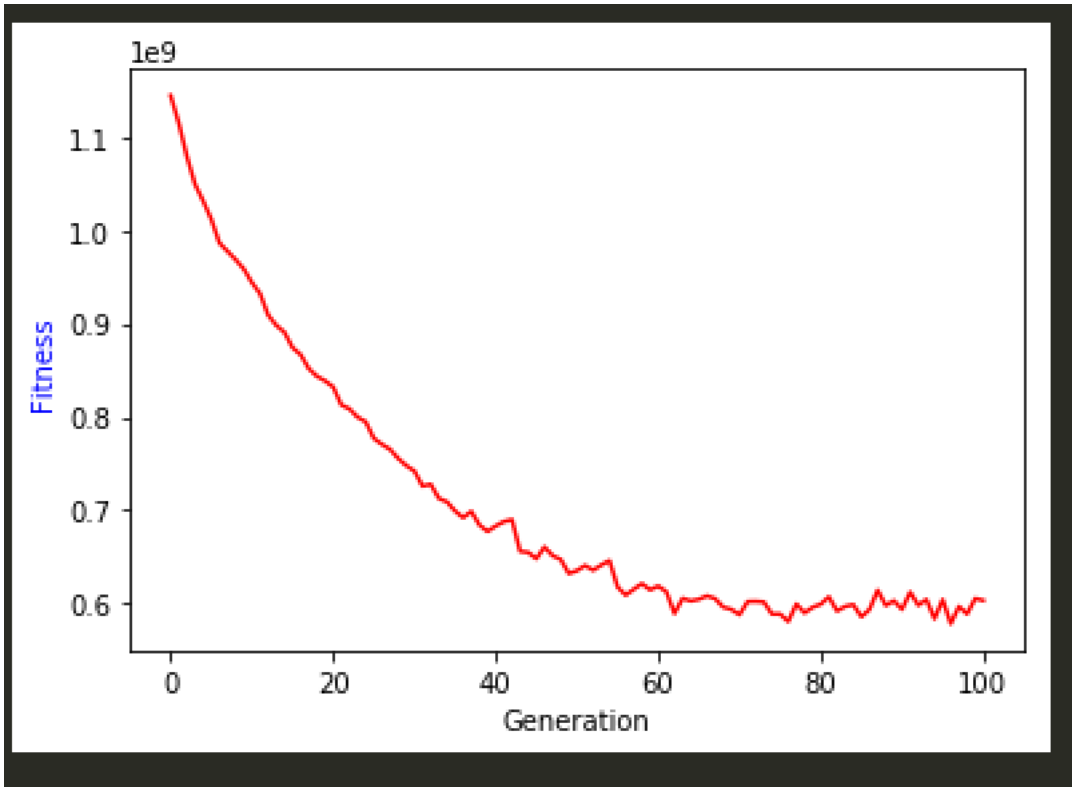


Ilustración 40 Gráfica del mejor resultado de la práctica 3.

Se ve a simple vista que el fitness ha mejorado muchísimo de una práctica a otra. Ambas graficas comienzan a equilibrarse entorno a las 100 generaciones, pero a diferente nivel de fitness.

Además, hemos observado que la configuración de la practica 3 y la del mejor resultado de esta comparativa son las mismas (cxUniform, mutFlipBit y selTournament).

Multi objetivo

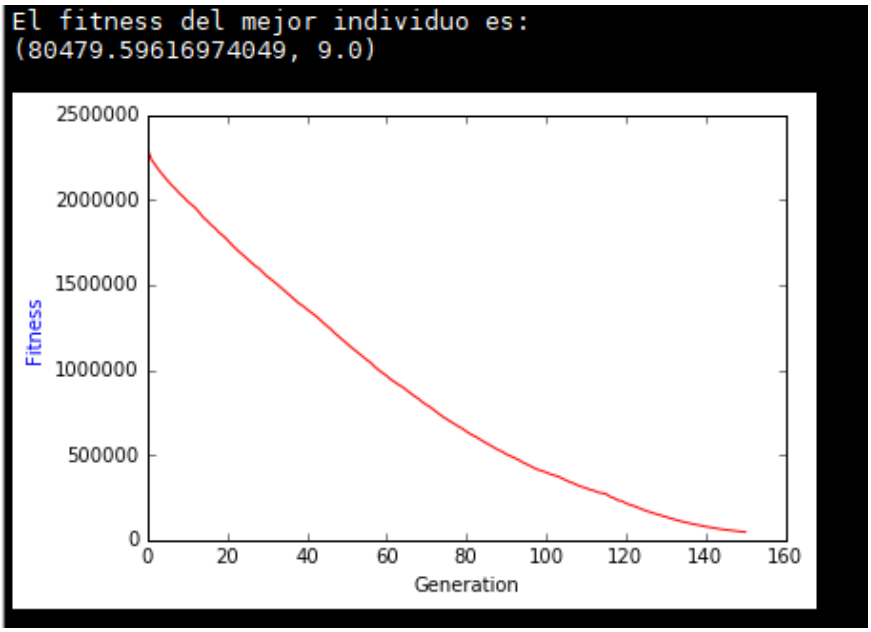


Ilustración 41 Mejor solución Multi-Objetivo

Si comparamos la mejor solución obtenida en esta práctica con la de la practica 3:

```
(548200000.0,)
```

Ilustración 42 Fitness del mejor resultado de la práctica 3.

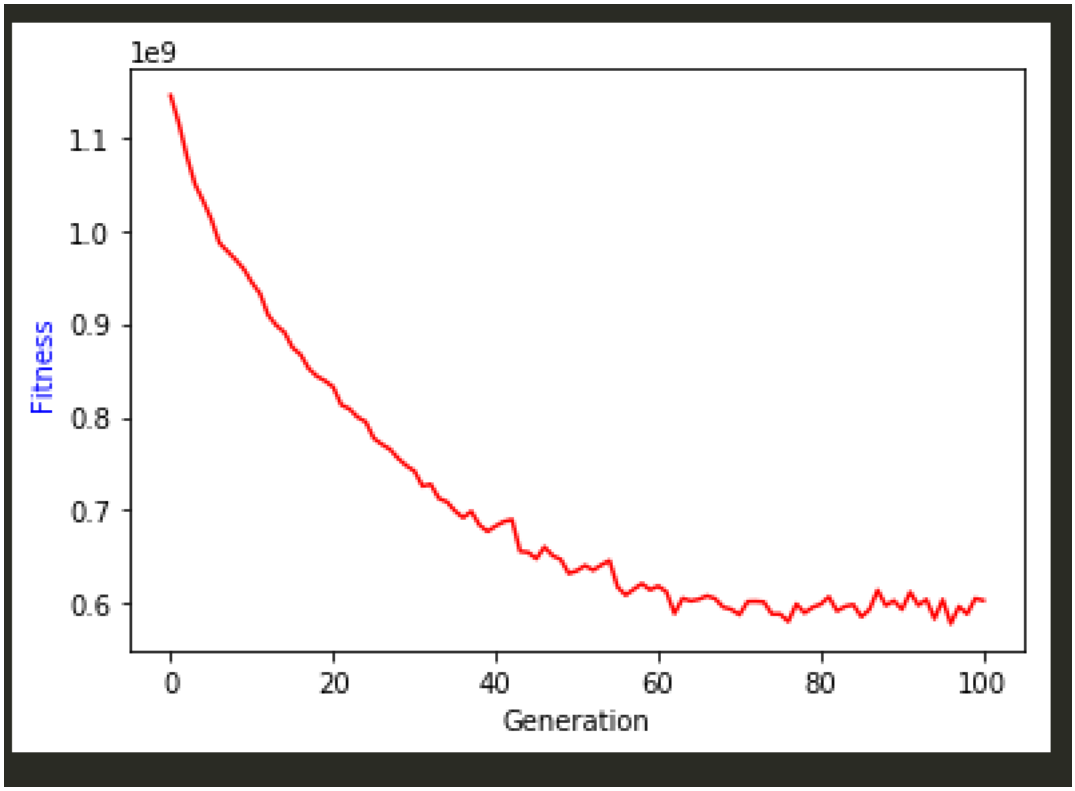


Ilustración 43 Gráfica del mejor resultado de la práctica 3.

Observamos que el fitness ha mejorado en gran medida, esto indica que esta configuración del problema es mucho más óptima que la que usamos en aquel entonces.