UNIVERSIDAD TECNOLÓGICA NACIONAL

FACULTAD REGIONAL ROSARIO

Cátedra: Algoritmos Genéticos

Trabajo Práctico Número 1:

Aplicación de un Algoritmo Genético para encontrar el máximo de una función

Integrantes del grupo:

Nombre	**Legajo**	**Correo**
Bassi, Danilo	43725	danilo-bassi@hotmail.com
Bella, Sebastián	43784	sebastian.o.bella.it@gmail.com
Garello, Iván	43804	ivangarello@gmail.com
Vaccino, Lucas Nehuén	43870	lnvacchino@gmail.com
Zilli, Joel	44107	joelz97@outlook.com

Ciclo lectivo: 2020

Profesores:
Daniela Diaz y Victor Lombardo

Índice

Aplicación de un Algoritmo Genético para encontrar el máximo de una función

Resumen

Este trabajo se orienta a la producción y descripción de un algoritmo genético para la resolución de un problema sobre el cual ya se conoce su solución. Se describe y detalla las distintas operaciones involucradas en el algoritmo y en la ejecución del mismo con distinta cantidad de generaciones a producir para su posterior análisis. Además, se realiza un análisis de sensibilidad sobre la variación de dos de las variables de entrada para así poder examinar la comparación de estos efectos con aquellos producidos con los parámetros originales.

Palabras Clave: Algoritmos Genéticos, Mutación, Crossover, Población, Función Objetivo, Optimización.

1. Enunciado

Este trabajo consiste en la Aplicación de un Algoritmo Genético para buscar un máximo de la siguiente función:

$$f(x) = \frac{x}{2^{30} - 1}$$

en el dominio $[0, 2^{30} - 1]$

Teniendo en cuenta los siguientes parámetros:

- Probabilidad de Crossover = 0.75
- Probabilidad de Mutación = 0.05
- Tamaño de la población: 10
- Ciclos del programa: 20
- Método de selección: Ruleta
- Método de Crossover: 1 Punto
- Método de Mutación: Invertida

Luego, se irán cambiando algunos de estos parámetros (como la probabilidad de mutación y la probabilidad de crossover) y algunos métodos de selección de población (de usar la ruleta pasaremos al método conocido como elitista).

Para iniciar el análisis, se realizarán 3 corridas con 20,100 y 200 generaciones respectivamente para poder observar el comportamiento del algoritmo. Luego de esto, se mostrarán las salidas del script codificado, en el cual se observarán la media, el máximo y mínimo obtenido de la función objetivo. También se dejará como anexo la tabla correspondiente a la gráfica anterior mencionada, en donde agregaremos como dato adicional, el cromosoma correspondiente al máximo obtenido para poder observar su evolución.

2. Metodología

La metodología aplicada, como se dijo en el enunciado, consiste en la utilización de un método sistemático capaz de resolver problemas de búsqueda y optimización. Dicho método es conocido como algoritmo genético. Este algoritmo reside en una secuencia de operaciones basadas en el proceso genéticos de los organismos vivos, utilizando como pasos fundamentales aquellos que también intervienen en la evolución biológica: la selección basada en la población, reproducción sexual y mutación. A lo largo de generaciones, las poblaciones evolucionan en la naturaleza de acuerdo con los principios de la selección natural y supervivencia de los más fuertes, postulados por Darwin. Por imitación de este proceso, los Algoritmos Genéticos son capaces de ir creando soluciones para problemas del mundo real. La evolución de dichas soluciones hacia valores óptimos del problema depende en buena medida de una adecuada codificación de las mismas. Los pasos que hemos aplicado para la resolución de nuestro algoritmo genético se basan en los siguientes procesos:

- Armado de la población inicial
- Selección de a pares de los mejores cromosomas para su reproducción
- Reproducción de las parejas elegidas
- Aplicación de mutación con baja frecuencia.

A continuación se expondrá en detalle en qué consiste cada paso.

2.1. Población inicial

La población que vamos a utilizar deberá estar conformada por un conjunto de individuos, de ahora en más denominados cromosomas. Para que el Algoritmo Genético pueda comenzar a funcionar, se debe establecer una población inicial asignándole un total de 10 cromosomas, cuyos valores serán generados aleatoriamente.

Como hemos dicho antes, nuestra función a maximizar está definida sobre un conjunto de números enteros entre 0 y $2^{30} - 1$. Por lo tanto, nuestro cromosoma se definirá como un conjunto de 30 genes, cada uno pudiendo tomar los valores 1 y 0. En este caso, para producir los primeros cromosomas, se generará una serie de 30 genes, en donde cada gen será un bit (0 o 1).

En resumen, nuestro primer paso fue generar nuestra población inicial con cromosomas representados por listas binarias en donde cada bit fue asignado aleatoriamente.

2.2. *Selección*

Una vez formada nuestra población inicial, pasamos al método de selección en donde se van a elegir a los mejores cromosomas para luego reproducirlos entre si y así formar la próxima generación. Para lograr esto, primero debemos asignar a cada cromosoma una puntuación llamada "Fitness"que representa cuán cerca está una solución de ser la mejor. A partir de este fitness, se determinará los cromosomas que se van a reproducir y aquellos que se van a eliminar.

La funcion Fitness para un cromosoma en particular está definido como:

$$Fitness(x) = \frac{f(x)}{\sum_{i=1}^{n} f(i)}$$

Siendo n el tamaño de la población.

Cuando terminamos de evaluar el fitness de cada cromosoma, se tiene que crear la nueva población tratando de que los buenos rasgos de los mejores se transmitan a ella. Para esto, como método de selección de los mejores padres, hemos aplicado el método de la rueda de ruleta. La rueda de ruleta cosiste en crear un pool genético formado por cromosomas de la generación actual, en una cantidad proporcional a su fitness. Dentro de este pool, se escogerán parejas aleatorias de cromosomas para que se puedan emparejar, sin importar incluso que ambas parejas sean iguales.

2.3. *Reproducción (Crossover)*

Consiste en el intercambio de material genético entre dos cromosomas, y en algunas ocasiones entre más de dos. El crossover es el principal operador genético del Algoritmo que estamos tratando. En caso de que se emparejen dos descendientes del mismo padre, no generaría ningún inconveniente; ello garantizará la perpetuación de un individuo con buena puntuación.

El método de reproducción que hemos usado para realizar este proceso es el crossover de 1 punto, que consiste en tomar dos padres seleccionados y cortar sus ristas de cromosomas en una única posición elegida al azar, para producir dos subristas iniciales y dos subristas finales. Después, se intercambian las subristas finales, produciéndose dos nuevos cromosomas completos. En conclusión, ambos descendientes heredaron genes de cada uno de los padres.

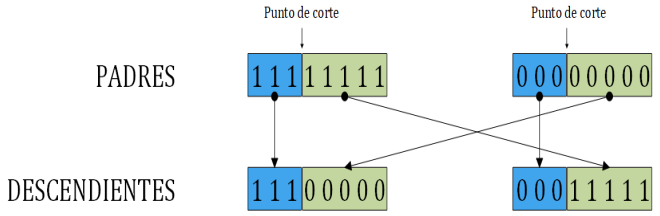

Figura 1: Ejemplo del operador de cruce basado en 1 punto.

Cabe destacar que tanto la reproducción como la mutación tienen una probabilidad de ocurrencia, y en caso que no ocurra la reproducción, los padres pasarán directamente a la próxima generación.

2.4. *Mutación*

En la evolución biológica, una mutación es un suceso bastante poco común (sucede aproximadamente una de cada mil replicaciones) y en la mayoría de los casos suele ser letal. Pero en el largo plazo, contribuyen a la diversidad genética de la especie. En un algoritmo genético tendrán el mismo papel: ocurrir con baja frecuencia para contribuir a la diversidad genética.

En nuestro caso, el operador mutación se aplica a cada hijo de manera individual, con probabilidad de ocurrencia del 5 %, y consiste en la alteración aleatoria de un gen componente elegido al azar del cromosoma (intercambiando un 0 por 1 o viceversa).

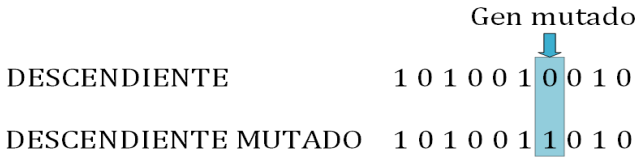

Figura 2: Ejemplo del operador de mutación.

3. Herramientas de programación e informe.

3.1. *Programación*

Para programar el algoritmo genético, elegimos como lenguaje de programación Python. Nuestra elección fue realizada en base a que algunos integrantes del grupo ya tenían conocimientos previos de este lenguaje y en caso de que no se conozca, la curva de aprendizaje, es baja, dado que es un lenguaje de sintaxis sencilla, de tipado dinámico y posee múltiples librerías que ayudan a facilitar considerablemente el desarrollo de programas.

En nuestro caso, utilizamos las siguientes librerías:

- Random: Es una librería muy robusta para la generación de números aleatorios. En nuestro caso fue utilizada para generar números aleatorios con distribución uniforme.
- Matplotlib: Es una librería de trazado 2D que produce gráficas de buena calidad en una variedad de formatos. En nuestro caso, todas las gráficas mostradas como salida del script, fueron generadas con esta librería.
- Math: Es una librería con las funciones matemáticas comunes, por ejemplo: redondeos, raíces , logaritmos, etc.
- Statistics: Es una librería que provee de múltiples funciones para el análisis estadístico. En nuestro caso, solo se utilizó para obtener la media en los arreglos.
- Pandas: Es una librería que proporciona estructuras de datos de alto rendimiento, fáciles de usar y herramientas de análisis de datos. En nuestro caso, se utilizó para pasar los datos que se iban almacenando en arreglos a un archivo de Excel, de donde finalmente se obtuvieron las tablas.

Se utilizó como IDE Visual Studio Code. Esta decisión se tomó porque Code posee una extensión llamada Live Share que permite a los miembros del grupo colaborar en tiempo real en la edición del código, así como también, compartir archivos y una terminal, donde todos los integrantes, pueden ver como el algoritmo se va ejecutando.

Como adicional, se utiliza un repositorio en GitHub para ir subiendo los avances de los trabajos prácticos. El mismo se puede acceder en el siguiente link: `https://github.com/danilobassi8/algoritmos-geneticos`

3.2. Informe

Para realizar el presente informe se utilizó Overleaf. Este es un editor online que permite la colaboración en tiempo real de documentos y que utiliza LaTeX como sistema de composición de textos.

LaTeX posee una gran cantidad de macros con la intención de facilitar la creación de documentos cuidando el formato, por ejemplo: para colocar imágenes, realizar tablas, establecer referencias, etc. Con Overleaf, se puede sacar mucho mas provecho de LaTeX, debido a que existen múltiples plantillas que ayudan a la realización de informes, con un formato preestablecido, de manera colaborativa y a tiempo real.

4. Corridas

Una vez programado el Script en Python (que se encuentra detallado en el anexo) se realizaron 3 corridas para observar el comportamiento del Algoritmo genético.

A continuación, se mostrarán los resultados de la ejecución del Script con 20, 100 y 200 generaciones.

Se dejará en el anexo las tablas de las corridas, producto de cada ejecución realizada, las cuales indican la generación, el máximo, el mínimo y la media de los valores de la función objetivos alcanzados en esa generación, y cual fue el cromosoma que causó ese máximo.

4.1. 20 generaciones

Los datos de esta ejecución se encuentran en la Tabla 1 (ver anexo)

Resumen de la tabla 1

- Máximo valor de la función objetivo obtenido: 0.73266589
- Cromosoma correspondiente: 11011011001000000100001111000
- Obtenido en la generación: 4

Se puede obtener de la Figura 3 varias conclusiones. Primero, se puede observar en la población inicial como la media se ubica casi en la mitad entre los valores máximo y mínimo, dando indicio de la aleatoriedad que se debe presentar en dicha

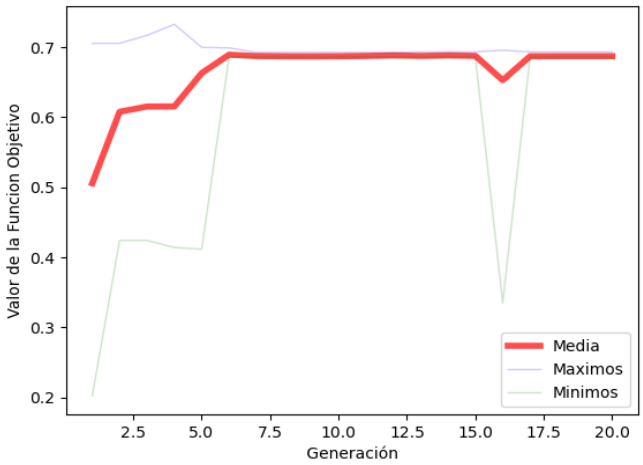

Figura 3: Salida al realizar 20 generaciones.

población. Luego, cabe señalar como en las siguientes generaciones el valor mínimo y la media aumentan de manera apresurada, como resultado del algoritmo utilizado. También, se hará notar el valle brusco que se genera en la población 16. Esto pudo haberse dado por el operador mutación del algoritmo que, pudo haber tenido un impacto negativo en el cromosoma afectado o por una reproducción, en donde alguno de los hijos pudo haber quedado con mayor cantidad de ceros a la izquierda. (este cromosoma se puede visualizar en la Figura 3 o en la Tabla 1 como el valor mínimo de la población 16).

Por último, se observa una tendencia a la convergencia de todos los valores, intentando acercarse a 1, valor óptimo de la función, pero la convergencia no logra llegar a ese valor dado que la cantidad de generaciones generadas, parecería escasa.

En resumen, con la realización de solamente 20 generaciones se pueden examinar todas las operaciones involucradas en el algoritmo. Pero, a la vez, se examina la desventaja de realizar pocas generaciones, ya que dieron como resultado final un valor cercano a 0.7, cuando ya se conoce que el valor máximo que se puede, y se debería obtener, es 1. O sea, con una cantidad chica de generaciones se obtendrá como resultado un valor final que, muy probablemente, no sea el valor óptimo. Este asunto se aclara en el análisis de los próximos experimentos.

4.2. 100 generaciones

Los datos de esta ejecución se encuentran en la Tabla 2 (ver anexo)

Resumen de la tabla 2

- Máximo valor de la función objetivo obtenido: 0.988071643
- Cromosoma correspondiente: 11111110011101111111010101111 1
- Obtenido en la generación: 66

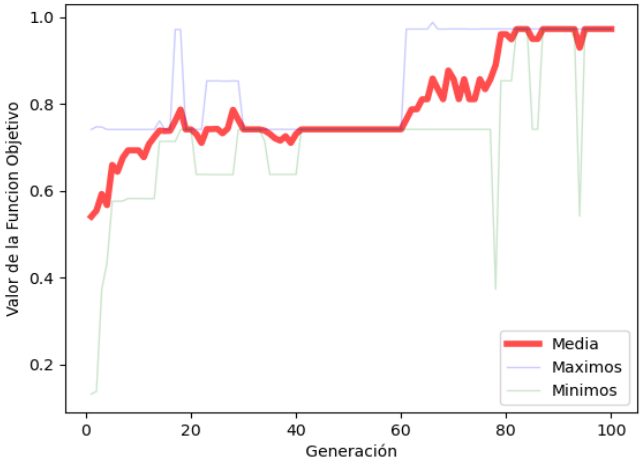

Figura 4: Salida al realizar 100 generaciones.

Aumentando el número de generaciones a 100, vemos resultados más prometedores. Se aprecia en principio la mantención de la aleatoriedad en la población, viendo como la media nuevamente se mantiene entre los valores máximo y mínimo de la población inicial. Sobre los valores mínimos, se observa que en las primeras generaciones tienen un aumento bastante abrupto; no tan así para los valores máximos, donde en los inicios se ve que no tienen una gran variación, salvo al acercarse a la generación 17 donde se observa un incremento significativo que incluso logra acercarse bastante al óptimo de la función objetivo. Para la media, hasta la generación 60 se observa que se mantiene oscilando entre valores cercanos al 0.60 y 0.80 de la función objetivo, y no es hasta poco después de esta generación donde comienza a crecer para finalmente llegar a valores cercanos a 1.

Se desea resaltar algo muy interesante entre las generaciones 40 y 60: los valores máximos y mínimos (y por supuesto, la media) se mantuvieron en valores muy cercanos, incluso en la generación 41, el máximo, mínimo y la media coincidieron. Cabe destacar, que aunque en la gráfica parezcan exactamente los mismos valores de la media, mínimos y máximos, los de cada poblaciones generadas entre la generación 40 y la 60 son sutilmente distintos. Esto puede verse mejor en la tabla 2 del anexo, donde se puede ver la diferencia entre el valor mínimo y máximo obtenido. Este comportamiento de valores similares se mantiene hasta la generación 61, en donde el máximo pasa de ser aproximadamente 0.74 a 0.97. Volvemos a reiterar que esto puede ser producto tanto de un cruce beneficioso entre dos cromosomas, o una mutación favorable en uno de ellos.

En cuanto a los máximos a partir de la generación 60, luego de este abrupto aumento, se mantienen en valores similares hasta la generación 100. Encontrando el valor máximo obtenido en la generación 66, siendo este de 0.988071643, un valor mucho mejor que el obtenido la corrida anterior.

En conclusión, vemos cómo nuestro algoritmo mejora de forma considerable con el paso de más generaciones, permitiendo que el resultado final converja aún más al valor óptimo de la función objetivo. Por consiguiente, se espera que para las 200 generaciones el algoritmo converja a un valor aún más cercano al óptimo de la función objetivo. Lo detalles se aclararán a continuación, analizando los resultados obtenidos de estas 200 generaciones.

4.3. 200 generaciones

Los datos de esta ejecución se encuentran en la Tabla 3 (ver anexo)

Resumen de la tabla 3

- Máximo valor de la función objetivo obtenido: 0.991214077
- Cromosoma correspondiente: 11111110110111110110111101110
- Obtenido en la generación: 136

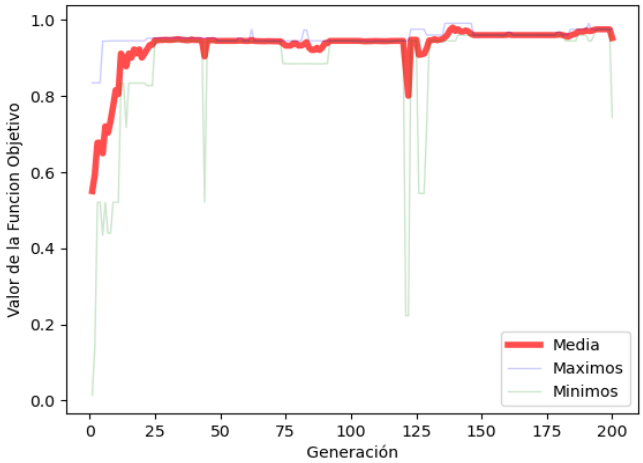

Figura 5: Salida al realizar 200 generaciones.

Finalizamos este proceso de análisis con la muestra de la ejecución del mismo programa pero esta vez con 200 generaciones. Otra vez, notamos todas las características particulares que posee el Algoritmo Genético: la aleatoriedad de la primera población, la convergencia rápida de los valores mínimos, media y máximos (en este caso, se presencia la primera convergencia en la generación 25), las ocasionales mutaciones ocurridas y la tendencia general a obtener valores cercanos a 1. En esta ejecución en particular, notamos como nuestra solución final se acerca todavía más al valor óptimo que en el caso anterior, debiendo este efecto al incremento en la cantidad de generaciones producidas. Y si continuamos de esta forma, aumentando cada vez más la cantidad de generaciones, entonces se podrá llegar a la generación de un cromosoma cuya función objetivo tenga un valor de 1, en otras palabras, se podría llegar a encontrar el cromosoma óptimo para nuestro problema.

5. Análisis de Sensibilidad

En esta sección, nos dedicaremos a realizar cambios las entradas del programa. Estos cambios se realizarán específicamente sobre dos de los parámetros de entradas que posee el algoritmo: La probabilidad de ocurrencia de la operación Crossover y la probabilidad de ocurrencia de la operación Mutación. Esta acción tiene como objetivo observar los efectos que tienen estas operaciones sobre el desarrollo de la evolución de los cromosomas. De esta manera, se podrán notar las ventajas y desventajas de cada uno, ya desarrolladas en el marco teórico, pero esta vez demostradas de forma práctica.

Nuestra metodología será, realizar una ejecución por cada cambio de parámetros, mostrar las gráficas de evolución, indicar los valores obtenidos al final de cada ejecución y realizar una conclusión sobre dichos cambios.

5.1. Variación de la probabilidad de Crossover

Para la variación de la probabilidad de ocurrencia del Crossover, cambiaremos su valor de 0.75 a 0.90.

Ya podemos anunciar que este cambio producirá probablemente una convergencia mas rápida hacia los valores óptimos. Aunque esta alta probabilidad de crossover generará, a la larga, un aumento en la frecuencia de emparejamientos de hijos de los mismos padres, lo que podría derivar en un estancamiento en un mínimo local, generando así un estancamiento en un resultado no óptimo.

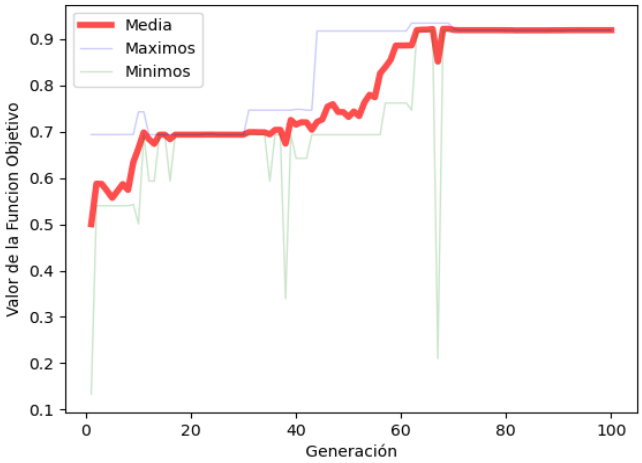

Figura 6: Cambio probabilidad de Crossover (0.90).

Resumen de la corrida

- Máximo valor de la función objetivo obtenido: 0.9337694045275231
- Cromosoma correspondiente: 11110111011000001001010001111 1
- Obtenido en la generación: 62

En el gráfico, se puede observar a simple vista como la alta probabilidad de crossover deriva en una convergencia mas rápida. El problema aquí es que, si observamos el valor máximo obtenido de la función objetivo, el mismo es de 0,93 en la generación 62 aunque luego ese valor disminuye para estancarse alrededor de 0,91. Esto ocurre debido a que probablemente se han emparejado muy frecuentemente hijos de los mismos padres, lo que genera un valor de convergencia no óptimo de la función. Recordemos que, el valor máximo de la función objetivo, con probabilidad de crossover de 0,75, es 0,98.

Podemos concluir entonces en que la probabilidad de crossover es conveniente que mantenga un alto valor para garantizar la evolución en cada generación, pero no un valor demasiado alto que derive en resultados no deseados (no óptimos) y estancamientos.

5.2. Variación de la probabilidad de Mutación

Para la variación de la probabilidad de ocurrencia del operador Mutación, cambiaremos su valor de 0.05 a 0.50.

Se puede predecir que este aumento de la probabilidad de mutación hará que la búsqueda no sea eficiente, reduciendo al Algoritmo Genético a una búsqueda aleatoria de valores.

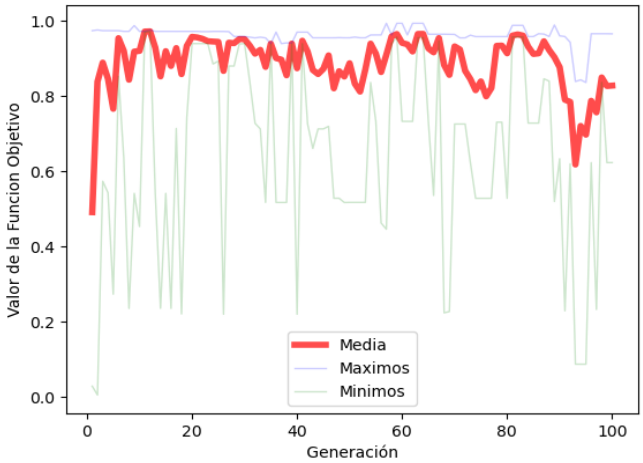

Figura 7: Cambio probabilidad de Mutación (0.50).

Resumen de la corrida

- Máximo valor de la función objetivo obtenido: 0.9940605218768982
- Cromosoma correspondiente: 11111111001111010001010111010 1
- Obtenido en la generación: 57

En el gráfico, se puede notar continuos saltos frecuentes, tanto en los valores máximos como en los valores mínimos. Esto imposibilita encontrar eficazmente el valor óptimo de la

función, dado que, como la probabilidad de mutación es tan alta, el problema se reduce a una búsqueda aleatoria de valores. Con el incremento en la media que ocurre dentro de las primeras 20 generaciones, podría parecer que el algoritmo funciona de manera similar a cuando tenía una probabilidad de mutación baja, pero generaciones posteriores, indicarían lo contrario, dado que se puede ver como en las ultimas generaciones la media disminuye, así como también los máximos y mínimos.

En definitiva, la probabilidad de Mutación debe ser lo suficientemente menor como para evitar los efectos vistos en el caso tratado.

6. Elitismo

El elitismo es un mecanismo que pretende asegurar que los individuos más aptos de la población sobrevivan y continúen participando en el proceso evolutivo, pasando a la siguiente generación de manera intacta, sin reproducirse ni mutarse. En nuestro caso, usaremos este mecanismo para resolver el mismo problema descrito en el Marco Teórico usando los mismos parámetros de entrada.

El elitismo se aplicará en nuestro programa seleccionando en cada generación los dos cromosomas con mejor fitness y preservándolos para pasarlos sin ninguna modificación a la generación siguiente. Luego, para conseguir los ocho cromosomas restantes, se utiliza el mismo procedimiento descrito anteriormente.

6.1. Qué podemos esperar

En base a la teoría podemos estimar que, el resultado final sería, a priori, más próximo al valor óptimo que el obtenido anteriormente sin utilizar elitismo. Esto se debe a que los valores máximos obtenidos pasan directamente a la próxima generación, es decir, la gráfica de los valores máximos no podrá descender.

6.2. Ejecución

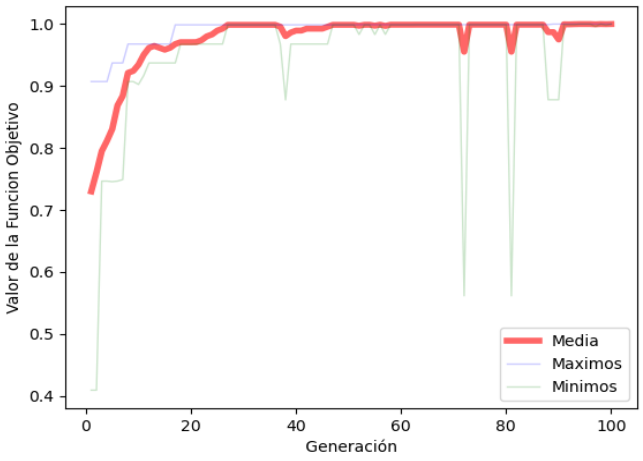

Figura 8: Aplicando Elitismo.

Resumen de la corrida

- Máximo valor de la función objetivo obtenido: 0.9997952014764631
- Cromosoma correspondiente: 11111111111100101001001111011
- Obtenido por primera vez en la generación: 90

6.3. Conclusiones

Podemos notar perfectamente las ventajas que posee un algoritmo genético con elitismo. De la gráfica se puede obtener varias conclusiones:

Primero, el valor máximo nunca disminuye, por lo que, si se llegó en la generación anterior a un máximo por una mutación, o un cruce beneficioso, este se mantendrá forzosamente dentro de nuestra población. Anteriormente, ocurrían picos con los valores máximos de la función objetivo, en donde en algunas generaciones podían disminuir drásticamente. (ver salida de 100 generaciones sin elitismo, generación 20). Aplicando elitismo, lo descrito anteriormente no sucede.

Segundo, los resultados obtenidos, son mucho mas cercanos al óptimo. Podemos esperar que, para una mayor cantidad de generaciones, el algoritmo finalmente converja en el óptimo.

En principio, parecería que aplicación de elitismo es siempre favorable, ya que observamos que garantiza la convergencia teórica global de nuestro algoritmo, además de mejorar la velocidad de esta convergencia. Sin embargo, esto sólo es beneficioso para funciones de evaluación que sean *unimodales*, es decir, funciones que tienen un sólo extremo (mínimo o máximo). La modalidad de las funciones es particularmente importante en optimización. Ahora bien, distinguir un extremo global de uno local hace difícil la optimización de la función. Esto se da en funciones de carácter *multimodales*, es decir, que presentan dos o más extremos. Para este último tipo de funciones, los algoritmos genéticos que usen como método de selección al elitismo tendrán una peor velocidad de convergencia al valor óptimo global. Además, puede suceder que el algoritmo converja de forma muy rápida, es decir, que tenga una convergencia prematura en la cual el algoritmo converja hacia óptimos locales.

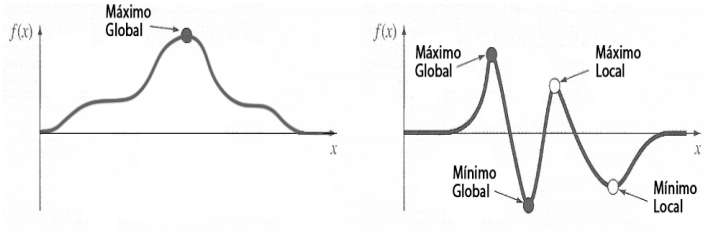

Figura 9: Función unimodal vs multimodal.

7. Anexo

7.1. Código en Python

A continuación, se deja el script en Python del algoritmo. El mismo se puede encontrar también en el siguiente link: `https://github.com/danilobassi8/algoritmos-geneticos/blob/master/TP1/TP1-solucion.py`

```
import random
import matplotlib.pyplot as plt
import math
import statistics
import pandas as pd


def rellenarPoblacionInicial():
    global poblacion, cantIndividuosEnPoblacion

    for i in range(cantIndividuosEnPoblacion):
        miCadena = ""
        for i in range(30):
            r = random.uniform(0, 1)
            if(r < 0.5):
                miCadena += '0'
            else:
                miCadena += '1'
        poblacion.append(miCadena)


def rellenarFuncionesObjetivoYFitness():
    global listaFitness, listaFObjetivo, poblacion

    # rellenar F objetivo
    for i in range(len(poblacion)):
        listaFObjetivo.append(funcionObjetivo(poblacion[i]))

    # rellenar F fitness
    sumatotal = sum(listaFObjetivo)
    for i in range(len(poblacion)):
        listaFitness.append(listaFObjetivo[i]/sumatotal)


def funcionObjetivo(cromosoma):
    x = int(cromosoma, 2)
    return (x/((2**30)-1))**2


def seleccionarPareja():
    global poblacion, listaFitness

    pareja = []
    ruleta = []

    # genero la ruleta como un pool proporcional al fitness.
    for i in range(len(listaFitness)):
        if i != 0:
            ruleta.append(listaFitness[i] + ruleta[i-1])
        else:
            ruleta.append(listaFitness[i])

    # elijo cada uno de los padres.
    for veces in range(2):
        r = random.uniform(0, 1)
        indice = 404
```

```
57
58          for i in range(len(ruleta)):
59              if i == 0:
60                  if(0 <= r <= ruleta[i]):
61                      indice = i
62                      break
63              else:
64                  if(ruleta[i-1] <= r <= ruleta[i]):
65                      indice = i
66                      break
67          pareja.append(poblacion[indice])
68
69      return pareja
70
71
72  def crossover(padres):
73      global p_crossover
74      hijo1 = ""
75      hijo2 = ""
76      # recibo los padres como string
77      padre1 = padres[0]
78      padre2 = padres[1]
79
80      r = random.uniform(0, 1)
81      if(r <= p_crossover):
82
83          # elijo el punto de crossover.
84          punto = round(random.uniform(0, len(padre1)-1))
85
86          for i in range(punto):
87              hijo1 = hijo1 + padre1[i]
88              hijo2 = hijo2 + padre2[i]
89          for i in range(punto, len(padre1)):
90              hijo1 = hijo1 + padre2[i]
91              hijo2 = hijo2 + padre1[i]
92
93          return [hijo1, hijo2]
94      else:
95          return [padre1, padre2]
96
97
98  def mutacion(hijo):
99      global p_mutacion
100
101     r = random.uniform(0, 1)
102     if (r <= p_mutacion):
103         # recibo el hijo como string y lo paso a lista para poder editarlo en un indice.
104         hijo = list(hijo)
105
106         indice = round(random.uniform(0, len(hijo)-1))
107
108         if (hijo[indice] == '0'):
109             hijo[indice] = '1'
110         else:
111             hijo[indice] = '0'
112
113         # paso la lista binaria a un string
114         string = ""
115         for i in range(len(hijo)):
116             string = string + hijo[i]
117         hijo = string
118
```

```
    return hijo


def elitismo():
    global poblacion, listaFitness, proximaGeneracion, cantElite

    # Creamos una copia de la lista fitness, elegimos el mejor, lo borramos
    # y volvemos a elegir el mejor. Luego sacamos los indices en el arreglo original.
    # y agregamos en la proximaGeneracion la poblacion en el indice de los mejores.

    indiceMejor = []
    copiaFitness = listaFitness.copy()
    for i in range(cantElite):
        mejor = max(copiaFitness)
        indiceMejor.append(listaFitness.index(mejor))
        copiaFitness.remove(mejor)

    for i in range(cantElite):
        proximaGeneracion.append(poblacion[indiceMejor[i]])


# ------------------- Definiciones de variables ------------------- #
# arreglo para las graficas
ejex = []
medias = []
maximos = []
minimos = []
maximos_en_binario = []
mejor_de_todos = 0

# Poblacion
poblacion = []
proximaGeneracion = []

# Lista F objetivo y F Fitness.
listaFObjetivo = []
listaFitness = []

# Parametros.
cantMaximaGeneraciones = 200
# probabilidades
p_crossover = 0.75
p_mutacion = 0.05
cantIndividuosEnPoblacion = 10


# ------------------- Comienzo del programa ------------------- #

hayElitismo = input("Aplicar elitismo? (s/n): ")
if(hayElitismo.lower() == 's'):
    hayElitismo = True
    cantElite = 2
else:
    hayElitismo = False
    cantElite = 0

rellenarPoblacionInicial()
rellenarFuncionesObjetivoYFitness()

terminado = False
cantidadCiclos = 0
```

```
while (terminado == False):
    cantidadCiclos = cantidadCiclos+1
    print("GENERACIN ", cantidadCiclos, " LISTA.")

    if(hayElitismo):
        elitismo()

    for i in range(int((len(poblacion)-cantElite)/2)):

        # seleccionar 2 individuos para el cruce
        padres = seleccionarPareja()
        # cruzar con cierta probabilidad 2 individuos y obtener descendientes
        hijos = crossover(padres)
        # Mutar con cierta probabilidad
        hijomutado1 = mutacion(hijos[0])
        hijomutado2 = mutacion(hijos[1])
        # Insertar descendientes en la proxima generacion
        proximaGeneracion.append(hijomutado1)
        proximaGeneracion.append(hijomutado2)

    poblacion = proximaGeneracion.copy()
    proximaGeneracion = []
    listaFObjetivo = []
    listaFitness = []

    # rellena funcion fitness y objetivo.
    rellenarFuncionesObjetivoYFitness()

    # actualizo los arreglos para las graficas
    ejex.append(cantidadCiclos)
    medias.append(statistics.mean(listaFObjetivo))
    maximos.append(max(listaFObjetivo))
    minimos.append(min(listaFObjetivo))
    mejor_de_todos = max([mejor_de_todos, max(listaFObjetivo)])

    # - calculo el mejor de esta generacion en binario.
    # veo en que indice est el maximo de la FObj.
    indice_maximo = listaFObjetivo.index(max(listaFObjetivo))
    # busco en la poblacion en ese indice.
    cromo_maximo_actual = poblacion[indice_maximo]
    maximos_en_binario.append(cromo_maximo_actual)

    # verifico si termina el Algoritmo.
    if(cantidadCiclos == cantMaximaGeneraciones):
        terminado = True


# Plotea las graficas.
plt.plot(ejex, medias, label='Media', linewidth=4, color="red", alpha=0.6)
plt.plot(ejex, maximos, label='Maximos', linewidth=1, color="blue", alpha=0.2)
plt.plot(ejex, minimos, label='Minimos',linewidth=1, color="green", alpha=0.2,)
plt.legend()
plt.ylabel(' Valor de la Funcion Objetivo ')
plt.xlabel(' Generacin ')

# Guardo en la grafica en un PNG.
plt.savefig("Resultado.png")

# segun la variable 'mejor_de_todos' que guarda la mejor FO calculo el mejor cromosoma
mejor_numero = round((math.sqrt(mejor_de_todos)*(pow(2, 30)-1)))
print("El maximo resultado obtenido fue: ",mejor_de_todos, " y lo obtuvo el: ", mejor_numero)
print("Corresponde con el binario: ", bin(mejor_numero).replace('0b', ''))
```

```
243
244
245 # guardo los datos en un excel.
246 filas = {
247     'Generacin': ejex,
248     'Media': medias,
249     'Maximo': maximos,
250     'Binarios': maximos_en_binario,
251     'Minimo': minimos
252 }
253
254 df = pd.DataFrame(filas, columns=[
255     "Generacin", "Media", "Maximo", "Binarios", "Minimo"])
256 df.to_excel(r"S:\Drive\UTN\5to ao - 2020\Trabajos Practicos\Algoritmos Geneticos\Ej 1 -
        Inicial\DatosExportadosUltimaCorrida.xlsx",
257             index=False, header=True)
258
259 plt.show()
```

7.2. Tablas

7.2.1. Tabla para 20 generaciones.

Tabla 1: 20 generaciones

Generación	Media	Máximo	Cromosoma (máximo)	Mínimo
1	0.505728612	0.705525278	11010111000001110101001100 1101	0.202245928
2	0.607795835	0.705525278	11010111000001110101001100 1101	0.42428185
3	0.615357631	0.716870462	11011000110000000010011101 0100	0.42428185
4	**0.615396009**	**0.732665897**	**11011011001000000010000111 1000**	**0.414165653**
5	0.663005314	0.699795066	11010110001001110101001100 0110	0.411795861
6	0.689302907	0.698978374	11010110000001110101001100 0110	0.685975942
7	0.68735406	0.692461824	11010101000001110101000101 0110	0.685975942
8	0.686948201	0.692461824	11010101000001110101000101 0110	0.685975966
9	0.686867308	0.692461824	11010101000001110101000101 0110	0.685975966
10	0.687028962	0.692461824	11010101000001110101000101 0110	0.685974386
11	0.687434695	0.692462008	11010101000001110101001100 1101	0.685974385
12	0.688245111	0.692462008	11010101000001110101001100 1101	0.685974386
13	0.687515982	0.693274702	11010101001001110101000101 0110	0.685974386
14	0.688245656	0.693274702	11010101001001110101000101 0110	0.685974386
15	0.687596712	0.69327469	11010101001001110101000100 1110	0.685974386
16	0.652897788	0.695716382	11010101100001110101001100 1101	0.334921454
17	0.687110078	0.693274702	11010101001001110101000101 0110	0.685975966
18	0.687190984	0.693274702	11010101001001110101000101 0110	0.685975966
19	0.687190985	0.693274702	11010101001001110101000101 0110	0.685975966
20	0.687191143	0.693274702	11010101001001110101000101 0110	0.685975966

7.2.2. Tabla para 100 generaciones.

Tabla 2: 100 generaciones

Generación	Media	Máximo	Cromosoma (máximo)	Mínimo
1	0.540898018	0.741463394	11011100011011111110101000 1110	0.132678689
2	0.554823022	0.747574439	11011101010101111111101011011	0.138708638
3	0.593078345	0.746929549	11011101001111111000101101 0101	0.373274274
4	0.567249	0.74146324	11011100011011111101000101 1110	0.434194339
5	0.659855454	0.74146324	11011100011011111101000101 1110	0.576316019
6	0.644185558	0.74146324	11011100011011111101000101 1110	0.576316018
7	0.67654574	0.74146324	11011100011011111101000101 1110	0.576665086
8	0.693483071	0.74146324	11011100011011111101000101 1110	0.582519859
9	0.693488813	0.741463232	11011100011011111101000101001	0.582519812
10	0.693486536	0.741463232	11011100011011111101000101001	0.58249652
11	0.67757547	0.741463232	11011100011011111101000101001	0.58249652
12	0.707959268	0.741463232	11011100011011111101000101001	0.582262658
13	0.723897727	0.741463232	11011100011011111101000101001	0.582262658
14	0.739167761	0.761412604	11011111011000100000100000 1000	0.714179392
15	0.737733515	0.741254336	11011100011001111111010101 1100	0.714177511
16	0.738420079	0.741254336	11011100011001111111010101 1100	0.714177511
17	0.761585331	0.971634531	11111100010101111101010101 1100	0.713768151
18	0.787413679	0.971634531	11111100010101111101010101 1100	0.741253926
19	0.741421162	0.741671502	11011100011101111101010101 1101	0.741254336
20	0.741504554	0.741674787	11011100011101111111010101 1100	0.741251052
21	0.731085421	0.741674787	11011100011101111111010101 1100	0.637927312
22	0.710376716	0.741674787	11011100011101111111010101 1100	0.637927312
23	0.7422872	0.853231716	11101100011101111111010101 1101	0.637930359
24	0.742248535	0.853231716	11101100011101111111010101 1101	0.637540426
25	0.742946023	0.853231716	11101100011101111111010101 1101	0.637540426
26	0.732529535	0.852780747	11101100011001111111010101 1100	0.637540426
27	0.743766267	0.852780747	11101100011001111111010101 1100	0.637540426
28	0.786817383	0.853231716	11101100011101111111010101 1101	0.637540426
29	0.763853933	0.852780747	11101100011001111111010101 1100	0.741254336
30	0.741632738	0.741674787	11011100011101111111010101 1100	0.741254336
31	0.741632738	0.741674787	11011100011101111111010101 1100	0.741254336
32	0.741674785	0.741674787	11011100011101111111010101 1100	0.741674774
33	0.741674789	0.741674839	11011100011101111111010111 1101	0.741674774
34	0.739007941	0.741674839	11011100011101111111010111 1101	0.715006309
35	0.731300348	0.741674839	11011100011101111111010111 1101	0.637930359
36	0.720925905	0.741674839	11011100011101111111010111 1101	0.637930359
37	0.715662049	0.741885045	11011100011111111111010101 0101	0.637930359

Continua en próxima pagina

Tabla 2 – *Continuación de la pagina anterior*

Generación	Media	Máximo	Cromosoma (máximo)	Mínimo
38	0.726015472	0.741674839	11011100011101111111010111101	0.637930359
39	0.710551459	0.741674787	11011100011101111111010111100	0.637930359
40	0.731289832	0.741674787	11011100011101111111010111100	0.637930359
41	0.741674787	0.741674787	11011100011101111111010111100	0.741674787
42	0.741674787	0.741674787	11011100011101111111010111100	0.741674787
43	0.741674787	0.741674787	11011100011101111111010111100	0.741674787
44	0.741674787	0.741674787	11011100011101111111010111100	0.741674787
45	0.741674787	0.741674787	11011100011101111111010111100	0.741674787
46	0.741674787	0.741674787	11011100011101111111010111100	0.741674787
47	0.74167413	0.741674787	11011100011101111111010111100	0.741668217
48	0.741674089	0.741674787	11011100011101111111010111100	0.741668217
49	0.741674089	0.741674787	11011100011101111111010111100	0.741668217
50	0.741674048	0.741674787	11011100011101111111010111100	0.741667806
51	0.741674751	0.741674839	11011100011101111111010111101	0.741674377
52	0.741674751	0.741674839	11011100011101111111010111101	0.741674377
53	0.741673478	0.741674839	11011100011101111111010111101	0.741661646
54	0.741674792	0.741674839	11011100011101111111010111101	0.741674787
55	0.741674787	0.741674787	11011100011101111111010111100	0.741674787
56	0.741674459	0.741674787	11011100011101111111010111100	0.741671502
57	0.741674623	0.741674787	11011100011101111111010111100	0.741673145
58	0.741674294	0.741674787	11011100011101111111010111100	0.741673145
59	0.741674613	0.741674787	11011100011101111111010111100	0.741673145
60	0.741674787	0.741674787	11011100011101111111010111100	0.741674787
61	0.764767423	0.972601144	11111100011101111111010111100	0.741674787
62	0.787860059	0.972601144	11111100011101111111010111100	0.741674787
63	0.787859871	0.972601144	11111100011101111111010111100	0.741674787
64	0.810952366	0.972601144	11111100011101111111010111100	0.741673145
65	0.810952178	0.972601144	11111100011101111111010111100	0.741673145
66	**0.858705502**	**0.988071643**	**11111110011101111111010111111**	**0.741673145**
67	0.834086515	0.972601144	11111100011101111111010111100	0.741673145
68	0.811036057	0.972601144	11111100011101111111010111100	0.741673145
69	0.87719272	0.972601144	11111100011101111111010111100	0.741673145
70	0.857181662	0.972841932	11111100011111111111010111101	0.741673145
71	0.811041882	0.972841932	11111100011111111111010111101	0.741673145
72	0.857227085	0.972841814	11111100011111111111010011100	0.741673145
73	0.811056772	0.972599263	11111100011101111100101011100	0.741674787
74	0.811077635	0.972599263	11111100011101111100101011100	0.741673145
75	0.857220805	0.972599263	11111100011101111100101011100	0.741673145
76	0.834134673	0.972841932	11111100011111111111010111101	0.741673145
77	0.857251387	0.972841932	11111100011111111111010111101	0.741673145
78	0.889749322	0.972841932	11111100011111111111010111101	0.373572074
79	0.960776139	0.972841932	11111100011111111111010111101	0.853455483
80	0.960776139	0.972841932	11111100011111111111010111101	0.853455483
81	0.948837687	0.972841932	11111100011111111111010111101	0.853399098
82	0.972599675	0.972840051	11111100011111111100101011101	0.97187896
83	0.972575596	0.972841932	11111100011111111111010111101	0.97187708
84	0.97257539	0.972840051	11111100011111111100101011101	0.97187708
85	0.949576024	0.972840051	11111100011111111100101011101	0.741883415
86	0.949576188	0.972840051	11111100011111111100101011101	0.741885057
87	0.972623718	0.972840051	11111100011111111100101011101	0.972599261
88	0.972599826	0.972601144	11111100011101111111010111100	0.972599261
89	0.97259945	0.972601137	11111100011101111111010111001	0.972599261
90	0.972599251	0.972599263	11111100011101111100101011100	0.972599146
91	0.972599251	0.972599263	11111100011101111100101011100	0.972599146
92	0.972502978	0.972599265	11111100011101111100101011110	0.971636411
93	0.972406693	0.972599263	11111100011101111100101011100	0.971636411
94	0.929539039	0.972599263	11111100011101111100101011100	0.541997026
95	0.972599262	0.972599263	11111100011101111100101011100	0.972599261
96	0.97262334	0.972840049	11111100011111111100101011100	0.972599261
97	0.972599262	0.972599263	11111100011101111100101011100	0.972599261
98	0.972599285	0.972599498	11111100011101111100111011100	0.972599261
99	0.972599309	0.972599498	11111100011101111100111011100	0.972599261
100	0.972599333	0.972599498	11111100011101111100111011100	0.972599261

7.2.3. Tabla para 200 generaciones.

Se recortó a fines de no ser repetitiva.

Tabla 3: 200 generaciones

Generación	Media	Máximo	Cromosoma (máximo)	Mínimo
1	0.550236657	0.834599079	11101001110111110101011100110	0.014693666
2	0.594526219	0.834599079	11101001110111110101011100110	0.14071608
3	0.677821823	0.834599079	11101001110111110101011100110	0.521022404
4	0.677665873	0.834602839	11101001110111110110110001000	0.521026999
5	0.649431295	0.944442981	11111000110010010111110100110	0.434781155
6	0.720770331	0.944443097	11111000110010010111111100110	0.521026998
7	0.703310558	0.944443097	11111000110010010111111100110	0.440317028
8	0.734061534	0.94509568	11111000110111110111111100110	0.439871721
9	0.773600068	0.94509568	11111000110111110111111100110	0.521027344
10	0.816005874	0.94509568	11111000110111110111111100110	0.521029971
11	0.804961253	0.94509568	11111000110111110111111100110	0.521030786
12	0.911824353	0.94509568	11111000110111110111111100110	0.833990627
13	0.900773202	0.945094767	11111000110111110110111101110	0.833990627
14	0.878004337	0.945094767	11111000110111110110111101110	0.717679806
15	0.911857038	0.945094767	11111000110111110110111101110	0.834153487
16	0.900763542	0.945094767	11111000110111110110111101110	0.834153487
17	0.922893946	0.945094767	11111000110111110110111101110	0.834146519
18	0.911809575	0.945094767	11111000110111110110111101110	0.834156995
19	0.922859026	0.945094767	11111000110111110110111101110	0.834156995
20	0.900719591	0.945094767	11111000110111110110111101110	0.834156873
21	0.9118078	0.945094767	11111000110111110110111101110	0.834101251
22	0.922952945	0.952228359	11111001110011110110110100110	0.827481037
23	0.934046699	0.952228475	11111001110011110110111100110	0.827481037
24	0.934760094	0.952228475	11111001110011110110111100110	0.827481037
25	0.946521449	0.952228472	11111001110011110110111100100	0.945094536
26	0.946569128	0.952704952	11111001110111110110111000110	0.94509471
27	0.947282501	0.952704952	11111001110111110110111000110	0.94509471
28	0.947282704	0.952705025	11111001110111110110111101110	0.94462014
29	0.947996088	0.952705025	11111001110111110110111101110	0.94462014
30	0.947329635	0.952705025	11111001110111110110111101110	0.944612725
31	0.947329643	0.952705054	11111001110111110110111111110	0.944612725
32	0.94813887	0.952705025	11111001110111110110111101110	0.945094767
...	...	...	...	...
130	0.946554929	0.960285988	11111010110111010110111101110	0.945002057
131	0.946549734	0.960285988	11111010110111010110111101110	0.945002057
132	0.949603553	0.960285988	11111010110111010110111101110	0.945002057
133	0.946555715	0.9603458	11111010110111110110111101110	0.945002057
134	0.948083782	0.9603458	11111010110111110110111101110	0.945002057
135	0.94961848	0.9603458	11111010110111110110111101110	0.945002057
136	**0.954235928**	**0.991214077**	**11111110110111110110111101110**	**0.945002057**
137	0.961910161	0.991214077	11111110110111110110111101110	0.945002057
138	0.97421255	0.991214077	11111110110111110110111101110	0.945031724
139	0.980367124	0.991214077	11111110110111110110111101110	0.945035432
140	0.971069077	0.991214077	11111110110111110110111101110	0.945035432
141	0.97568691	0.991214077	11111110110111110110111101110	0.960106564
...	...	...	...	...
182	0.958725017	0.961063684	11111010111101110110111101100	0.945094764
183	0.955675192	0.961063684	11111010111101110110111101100	0.944857435
184	0.95878528	0.975477756	11111100110101110110111101100	0.944857435
185	0.962038736	0.975477756	11111100110101110110111101100	0.944857435
186	0.963575856	0.975477756	11111100110101110110111101100	0.944857435
187	0.969520703	0.975477756	11111100110101110110111101100	0.96010656
188	0.96942499	0.975477756	11111100110101110110111101100	0.96010656
189	0.96932853	0.975477756	11111100110101110110111101100	0.960099085
190	0.972403517	0.975477756	11111100110101110110111101100	0.96010656
191	0.970887797	0.990971022	11111110110101110110111101100	0.944857435
192	0.97087559	0.975477756	11111100110101110110111101100	0.944857435
193	0.973937622	0.975477756	11111100110101110110111101100	0.96010656
194	0.975465676	0.975477758	11111100110101110110111101101	0.97544738
195	0.975464169	0.975492827	11111100110101111110111101100	0.97544738
196	0.975467207	0.975492827	11111100110101111110111101100	0.975447615
197	0.975468667	0.975477758	11111100110101110110111101101	0.975447144
198	0.97546563	0.975477758	11111100110101110110111101101	0.975447144
199	0.975465629	0.975477758	11111100110101110110111101101	0.97544738
200	0.952327879	0.975477758	11111100110101110110111101101	0.74416053

7.2.4. Tabla para Elitismo.

Tabla 4: 100 generaciones con Elitismo

Generación	Media	Máximo	Cromosoma (máximo)	Mínimo
1	0.729933645	0.907080828	11110011110100010000000001001110	0.409482133
2	0.760827621	0.907080828	11110011110100010000000001001110	0.409489879
3	0.794869196	0.907080828	11110011110100010000000001001110	0.74654612
4	0.811555693	0.907080828	11110011110100010000000001001110	0.74654612
5	0.830571757	0.937087716	11110111110100010000000001001110	0.745689404
6	0.868505328	0.937087716	11110111110100010000000001001110	0.746559304
7	0.884037181	0.937087716	11110111110100010000000001001110	0.749066394
8	0.921187284	0.967582885	11111011110100010000000001001110	0.907080828
9	0.92418652	0.967582885	11111011110100010000000001001110	0.907080828
10	0.934313545	0.967582885	11111011110100010000000001001110	0.902450854
11	0.95031383	0.967582885	11111011110100010000000001001110	0.916873015
12	0.961483851	0.967582885	11111011110100010000000001001110	0.937087716
13	0.964533368	0.967582885	11111011110100010000000001001110	0.937087716
14	0.961483851	0.967582885	11111011110100010000000001001110	0.937087716
15	0.958434334	0.967582885	11111011110100010000000001001110	0.937087716
16	0.961483851	0.967582885	11111011110100010000000001001110	0.937087716
17	0.967583783	0.998566335	11111111110100010000000001001110	0.937087716
18	0.970585275	0.998566335	11111111110100010000000001001110	0.967102643
19	0.970585275	0.998566335	11111111110100010000000001001110	0.967102643
20	0.970585557	0.998566335	11111111110100010000000001001110	0.967102643
21	0.970681605	0.998566335	11111111110100010000000001001110	0.967582885
22	0.973779856	0.998566335	11111111110100010000000001001110	0.967582885
23	0.979977203	0.998566335	11111111110100010000000001001110	0.967582885
24	0.983075454	0.998566335	11111111110100010000000001001110	0.967582885
25	0.9892713	0.998566335	11111111110100010000000001001110	0.967582885
26	0.992369645	0.998566335	11111111110100010000000001001110	0.967582885
27	0.998566334	0.998566335	11111111110100010000000001001110	0.998566328
28	0.998566334	0.998566335	11111111110100010000000001001110	0.998566328
29	0.998566333	0.998566335	11111111110100010000000001001110	0.998566328
30	0.998563283	0.998566335	11111111110100010000000001001110	0.99853584
31	0.998566333	0.998566335	11111111110100010000000001001110	0.998566328
32	0.998566333	0.998566335	11111111110100010000000001001110	0.998566328
33	0.998566328	0.998566387	11111111110100010000000001101010	0.998566216
34	0.99856634	0.998566387	11111111110100010000000001101010	0.998566335
35	0.99856634	0.998566387	11111111110100010000000001101010	0.998566335
36	0.998566346	0.998566387	11111111110100010000000001101010	0.998566335
37	0.995464961	0.998566387	11111111110100010000000001101010	0.967582885
38	0.980269265	0.998566387	11111111110100010000000001101010	0.877562221
39	0.986172986	0.998566387	11111111110100010000000001101010	0.967582885
40	0.989271331	0.998566387	11111111110100010000000001101010	0.967582885
41	0.989271331	0.998566387	11111111110100010000000001101010	0.967582885
42	0.992370063	0.99857014	11111111110100010010000010010 10	0.967582936
43	0.992370069	0.99857014	11111111110100010010000010010 10	0.967582936
44	0.992370076	0.99857014	11111111110100010010000010010 10	0.967582943
45	0.992370075	0.99857014	11111111110100010010000010010 10	0.967582936
46	0.995468421	0.99857014	11111111110100010010000010010 10	0.967582936
47	0.998566765	0.99857014	11111111110100010010000010010 10	0.998566387
48	0.99856714	0.99857014	11111111110100010010000010010 10	0.998566387
49	0.998567891	0.99857014	11111111110100010010000010010 10	0.998566387
50	0.998567891	0.998570199	11111111110100010010000011010 10	0.998566328
51	0.99856749	0.998570199	11111111110100010010000011010 10	0.998566328
52	0.997036224	0.998810308	11111111110110010000000010010 10	0.983013567
53	0.998615898	0.998810308	11111111110110010000000010010 10	0.998566328
54	0.998665081	0.998810308	11111111110110010000000010010 10	0.998566328
55	0.997157827	0.998810308	11111111110110010000000010010 10	0.983013567
56	0.998762662	0.998817933	11111111110110010100000010010 10	0.998566328
57	0.996817946	0.998817933	11111111110110010100000010010 10	0.983255641
58	0.998765712	0.998817933	11111111110110010100000010010 10	0.998566328
59	0.998738639	0.998817933	11111111110110010100000010010 10	0.998566328
60	0.998764562	0.998817933	11111111110110010100000010010 10	0.998566328
61	0.99876685	0.998817933	11111111110110010100000010010 10	0.998566328
62	0.998767612	0.998817933	11111111110110010100000010010 10	0.998566328
63	0.998817171	0.998817933	11111111110110010100000010010 10	0.998810308
64	0.998817933	0.998817933	11111111110110010100000010010 10	0.998817933
65	0.998817933	0.998817933	11111111110110010100000010010 10	0.998817933
66	0.998817933	0.998817933	11111111110110010100000010010 10	0.998817933

Continua en próxima pagina

Tabla 4 – *Continuación de la pagina anterior*

Generación	Media	Máximo	Cromosoma (máximo)	Mínimo
67	0.998817933	0.998817933	11111111101100101000001001010	0.998817933
68	0.998817933	0.998817933	11111111101100101000001001010	0.998817933
69	0.998817933	0.998817933	11111111101100101000001001010	0.998817933
70	0.998817932	0.998817933	11111111101100101000001001010	0.998817918
71	0.998817933	0.998817933	11111111101100101000001001010	0.998817933
72	0.955097493	0.998817933	11111111101100101000001001010	0.561613537
73	0.998817939	0.998817993	11111111101100101000001101010	0.998817933
74	0.998817951	0.998817993	11111111101100101000001101010	0.998817933
75	0.998817957	0.998817993	11111111101100101000001101010	0.998817933
76	0.998818058	0.998818886	11111111101100101001001001010	0.998817933
77	0.998818237	0.998818946	11111111101100101001001101010	0.998817933
78	0.998818633	0.998818976	11111111101100101001001111010	0.998817933
79	0.998818729	0.998818976	11111111101100101001001111010	0.998817933
80	0.99881874	0.998818976	11111111101100101001001111010	0.998817933
81	0.955098386	0.998818976	11111111101100101001001111010	0.561614296
82	0.99881886	0.998818976	11111111101100101001001111010	0.998817993
83	0.998818965	0.998818983	11111111101100101001001111110	0.998818888
84	0.998818953	0.998818983	11111111101100101001001111110	0.998818888
85	0.998818947	0.998818983	11111111101100101001001111110	0.998818886
86	0.998818947	0.998818983	11111111101100101001001111110	0.998818886
87	0.998831162	0.998940948	11111111101110101001001101010	0.998818888
88	0.986729174	0.998940948	11111111101110101001001101010	0.877799063
89	0.986838994	0.999795112	11111111111100101001001001011	0.877799033
90	**0.974834613**	**0.999795201**	**11111111111100101001001111011**	**0.877799033**
91	0.999429073	0.999795201	11111111111100101001001111011	0.998818888
92	0.999331463	0.999795201	11111111111100101001001111011	0.998818888
93	0.999612139	0.999795201	11111111111100101001001111011	0.998818977
94	0.999792053	0.999795201	11111111111100101001001111011	0.999764687
95	0.999791959	0.999795201	11111111111100101001001111011	0.999764687
96	0.999792055	0.999795201	11111111111100101001001111011	0.999764687
97	0.999404998	0.999795201	11111111111100101001001111011	0.995893166
98	0.999795201	0.999795201	11111111111100101001001111011	0.999795201
99	0.999600004	0.999795201	11111111111100101001001111011	0.99784323
100	0.999795201	0.999795201	11111111111100101001001111011	0.999795201