



CENTRO DE CIENCIAS BÁSICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN
METAHEURÍSTICAS I
7° "A"

PRIMERA EVALUACIÓN PARCIAL

Profesor: Francisco Javier Luna Rosas

Alumno: Joel Alejandro Espinoza Sánchez

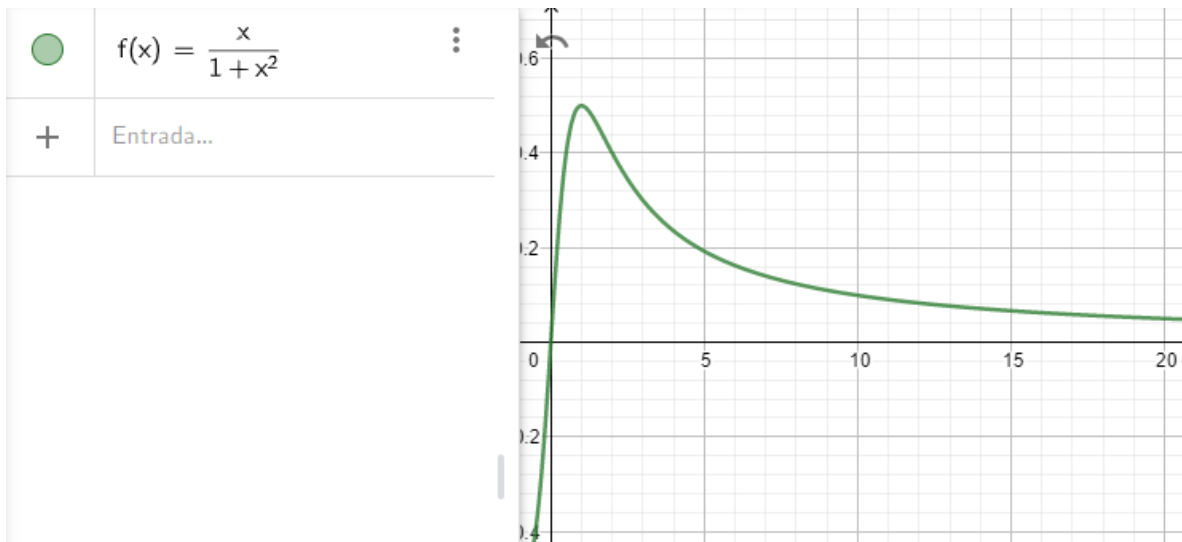
Fecha de Entrega: Aguascalientes, Ags., 15 de septiembre de 2021

Primera Evaluación Parcial

Objetivo

Mediante el desarrollo del examen, se implementará un algoritmo genético para buscar el máximo en el intervalo $[0,20]$ de la función:

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



Introducción

Un algoritmo es una serie de pasos organizados que describe el proceso que se debe seguir, para dar solución a un problema específico.

Estos algoritmos llevan ese nombre debido a que están inspirados en la evolución biológica. “El principio de su operación se basa en la creencia de que, si un individuo es apto para adaptarse a su medio, entonces tiene una mayor probabilidad de procrearse y transmitir sus características genéticas a su descendencia; haciendo así que las poblaciones de individuos sean mejores con el paso del tiempo” (Torres, 2020).

Estos algoritmos hacen evolucionar una población de individuos, o conjunto de soluciones posibles del problema, “sometiéndola a acciones aleatorias semejantes

a las que actúan en la evolución biológica tales como mutaciones y recombinaciones genéticas” (Luna, 2021); así como también a “una selección de acuerdo con algún criterio, en función del cual se decide cuáles son los individuos más adaptados, que sobreviven, y cuáles los menos aptos, que son descartados” (Talbi, 2009).

Procedimiento

Para resolver este problema primero se modeló el problema y su forma de resolverlo:

Modelo

Como variables del programa denotamos las siguientes:

- *qelitism*: Cantidad de cromosomas que se heredan automáticamente.
- *ig*: Iterador de generaciones.
- *pc*: Probabilidad de cruzamiento.
- *pm*: Probabilidad de mutación.
- *qe*: Cantidad de elementos (genes) que tendrá cada individuo.
- *qi*: Cantidad de individuos (cromosomas) que tendrá la muestra.
- *qg*: Cantidad de generaciones. Útil únicamente cuando *autom* = 1.

Con las variables anteriores se crean las siguientes estructuras:

- *sample*: Una matriz que tiene la población de muestra en la generación *ig* de tamaño *qi* filas y *qe* columnas:

$$\begin{bmatrix} i_1e_1 & i_1e_2 & i_1e_3 & \dots & i_1e_{qe} \\ i_2e_1 & i_2e_2 & i_2e_3 & \dots & i_2e_{qe} \\ i_3e_1 & i_3e_2 & i_3e_3 & \dots & i_3e_{qe} \\ \dots & \dots & \dots & \dots & \dots \\ i_{qi}e_1 & i_{qi}e_2 & i_{qi}e_3 & \dots & i_{qi}e_{qe} \end{bmatrix}$$

Donde i_1e_1 significa ser el dato del individuo 1 correspondiente a su elemento 1. Es decir, el gen 1 del cromosoma 1. i_ne_m significa ser el dato del individuo n correspondiente a su elemento m . Es decir, el gen m del cromosoma n .

No sólo lo anterior, `sample`, también contendrá la codificación de los elementos de la siguiente manera:

$$\left[\begin{pmatrix} i_1 e_1 & i_1 e_2 & i_1 e_3 & \dots & i_1 e_{qe} \\ i_2 e_1 & i_2 e_2 & i_2 e_3 & \dots & i_2 e_{qe} \\ i_3 e_1 & i_3 e_2 & i_3 e_3 & \dots & i_3 e_{qe} \\ \dots & \dots & \dots & \dots & \dots \\ i_{qi} e_1 & i_{qi} e_2 & i_{qi} e_3 & \dots & i_{qi} e_{qe} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_{qi} \end{pmatrix} \begin{pmatrix} f(x)_1 \\ f(x)_2 \\ f(x)_3 \\ \dots \\ f(x)_{qi} \end{pmatrix} \begin{pmatrix} f(x)a_1 \\ f(x)a_2 \\ f(x)a_3 \\ \dots \\ f(x)a_{qi} \end{pmatrix} \right]$$

Esto quiere decir que `sample` estará compuesta de cuatro columnas: La primera columna será la cadena de caracteres '0' o '1' para cada espacio. Después, la siguiente columna será la codificación de binario a número decimal del valor que la cadena de la columna anterior representa. La tercera columna es la evaluación dentro de la función que se nos dejó optimizar y finalmente se tendrá la función de forma acumulada para pasos de algoritmos como selección por ruleta.

- `newSample`: Una matriz igual a `sample`.

El proceso del algoritmo es el siguiente

- 1) Se genera aleatoriamente cada valor de la primera columna del arreglo `sample`. Donde: $i_n e_m \in \{0,1\} \forall n = 1,2,3, \dots q_i$ y $\forall m = 1,2,3, \dots q_e$.
- 2) Se interpreta la población bajo la fórmula:
Este valor se guarda en la segunda columna del arreglo `sample`.

$$x = x_{min} + bin_ent(c) \times \frac{x_{max} - x_{min}}{2^L - 1}$$

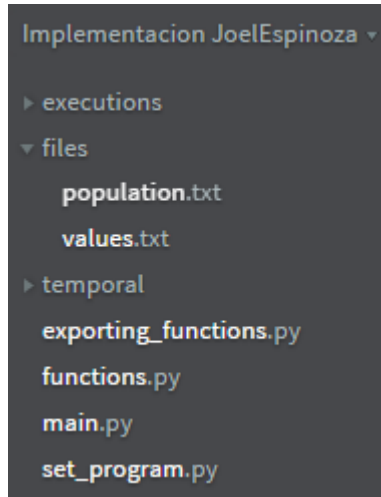
- 3) Se llenará la tercera columna del arreglo `sample` evaluando el valor anterior en la siguiente función:

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

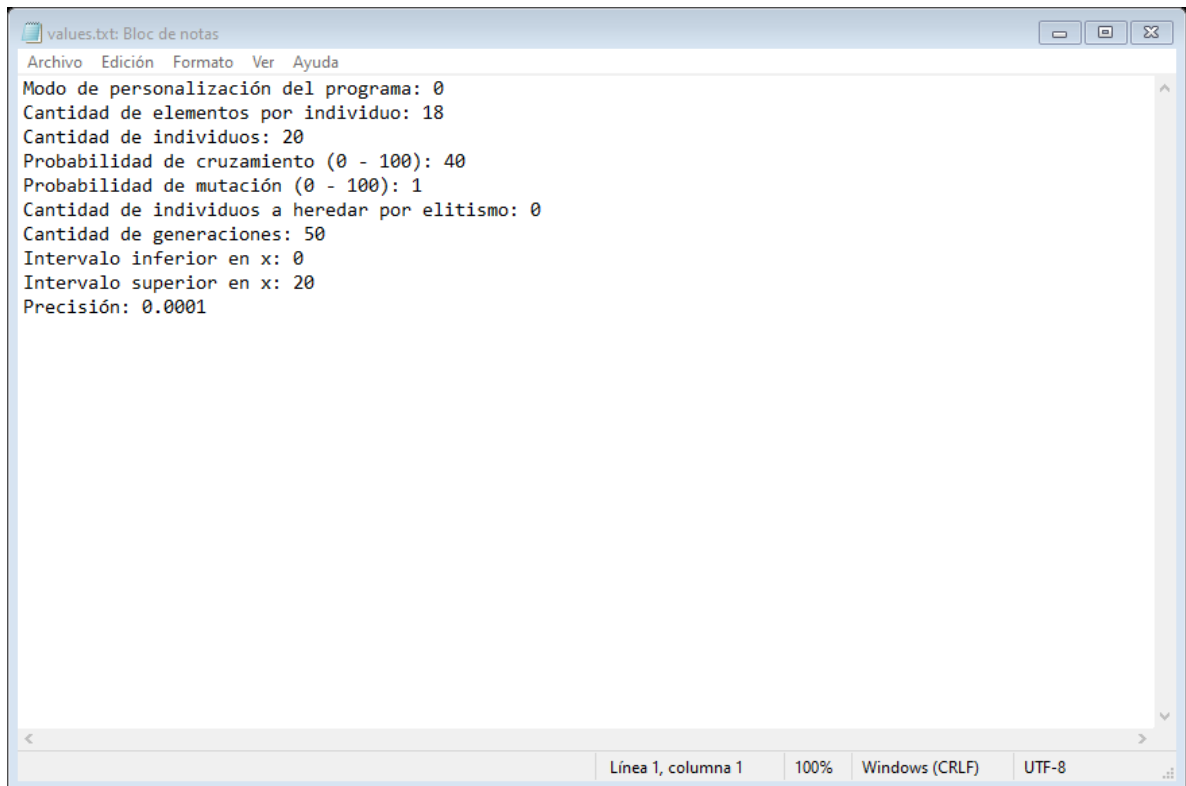
- 4) Se llena la última columna del arreglo `sample` como función acumulada de $f(x)$.
- 5) Se aplicará a `sample` el ordenamiento de la burbuja de manera descendente con base en la tercera columna del arreglo.
- 6) Según el número indicado para realizar elitismo, se tomarán los primeros `qelitism` mejores y pasarán directamente a la estructura `newSample`.
- 7) Se seleccionan dos cromosomas aleatoriamente usando la ruleta con base en la cuarta columna de `sample`.
- 8) Con los dos cromosomas seleccionados, se aplicará el cruzamiento si se respeta la probabilidad de cruzamiento y los dos hijos se añadirán a `newSample`.
- 9) Se repetirán los pasos 7 y 8 desde que un contador $k = qelitism + 1$ hasta $k = q_i$ con paso $k = k + 2$ (porque en cada paso se agregan dos cromosomas nuevos).
- 10) Para cada cromosoma, se aplicará una evaluación y verificar si se activa la mutación en un gen aleatorio.
- 11) Ahora se sobrescribirá `sample` con los valores de `newSample` y `newSample` se volverá una matriz nula.

Implementación en Python

El programa posee la siguiente estructura de archivos y carpetas:



Para iniciar la ejecución del programa, es necesario primero abrir y configurar el archivo `values.txt`; en éste se configuran los parámetros a usar cuando se ejecute el algoritmo. Por defecto, el archivo se encuentra de la siguiente manera:



En donde todas las variables a modificar son muy intuitivas para su uso excepto el modo de personalización de programa. Este modo puede establecerse como 0 o como 1. Si se establece como 0, la personalización se llevará a cabo usando la población escrita en `population.txt` (véase anexo 1). Al establecerse como 1, se usará una población generada aleatoriamente.

A continuación, se podrá ejecutar simplemente el archivo `main.py`, pero se dará explicación de cada paso del procedimiento.

El procedimiento se lleva a cabo al ejecutar el archivo `main.py` (véase anexo 2) como ya se dijo anteriormente, éste controla todo el procedimiento del archivo dividido en los otros archivos que hay dentro del directorio. El primer paso que realiza, de hecho, es cargar el archivo `set_program.py` (véase anexo 3), el cual realizará todas las configuraciones de datos extrayéndolos desde `values.txt`.

A continuación están las instrucciones para la carga de `set_program.py` desde el archivo `main.py`:

```
## Configuración del programa
stream = open("set_program.py", encoding="utf-8")
read_file = stream.read()
exec(read_file)

# Ya se tienen las variables
set_customization, qe, qi, pc, pm, qelitism, qg, x_min, x_max, accuracy, sample = set_values()
ig = 0
finals = []
```

Después, en el archivo `set_program.py` se configura cada variable. Entre las partes más notables del código se presentan las siguientes:

```
def set_values():
    sample = [[], [], [], []]
    values_file = open("files/values.txt", "r")

    # Se obtiene el modo de personalización
    set_customization = int(get_value(values_file.readline()))

    # La personalización se hará parcialmente por values.txt junto con population.txt
    if set_customization == 0:
        # Se ignoran dos renglones en values.txt
        pc = float(get_value(values_file.readline()))
        pc = float(get_value(values_file.readline()))

        # Se abre population.txt
        population_file = open("files/population.txt", "r")
```

```
# La personalización se hará enteramente por values.txt
if set_customization == 1:
    qe = int(get_value(values_file.readline()))
    qi = int(get_value(values_file.readline()))

    pc = float(get_value(values_file.readline()))
    pm = float(get_value(values_file.readline()))
    qelitism = int(get_value(values_file.readline()))
    qg = int(get_value(values_file.readline()))
    x_min = float(get_value(values_file.readline()))
    x_max = float(get_value(values_file.readline()))
    accuracy = float(get_value(values_file.readline()))

values_file.close()
```

En esa parte, el algoritmo hace la operación para recomendar al usuario la mejor cantidad de elementos por individuo según la precisión e intervalo dados:

$$L = \log_2 \left[1 + \left(\frac{x_{max} - x_{min}}{precision} \right) \right]$$

Esto se le recomienda al usuario sólo si este dato y el valor dado en values.txt son discordantes.

Una vez configurado el programa, puede empezar a ejecutar el algoritmo genético. Sin embargo, por convenciones personales, se decidió reportar todo en archivos externos, de modo que fuera más fácil leer todo el procedimiento realizado por el algoritmo.

Para poder exportar esta presentación del reporte de trabajo, se decidió separar toda función que tratará de imprimir alguna clase de información del algoritmo en el archivo exporting_functions.py (véase anexo 4) el cual será cargado como siguiente instrucción, pero en términos del algoritmo genético, su función se limita únicamente a reportar los resultados de éste, así que se explicará hasta el término de la ejecución del procedimiento.

```
stream = open("exporting_functions.py", encoding="utf-8")
read_file = stream.read()
exec(read_file)

PEC = PreviousExecutionCounter()
FirstExport(PEC + 1, set_customization, qe, qi, pc, pm, qelitism, qg, x_min, x_max, accuracy, sample)
```

A continuación se carga el último archivo, que es functions.py (véase anexo 5). Este archivo se apoya en funciones adicionales sólo excluidas del archivo principal

para mejorar la estética del código. Se irán comentando dichas funciones a lo largo de la explicación.

Primero se evalúa la estructura `sample` con una función `evaluate` en `functions.py`. A continuación la invocación a esta función:

```
# Ordenamos
sample = JoulSort(sample, qi)
sample = evaluate(qe, x_min, x_max, sample, 1)
Sorting(PEC + 1, qi, sample)
```

También el código de la función:

```
def evaluate(qe, x_min, x_max, sample, overwrite = 0):
    for ii in range(len(sample[0])):
        bin_ent = int(sample[0][ii], 2)
        x = x_min + (bin_ent * ((x_max - x_min) / ((2**qe) - 1)))
        if overwrite == 0:
            sample[1].append(x)
            sample[2].append(f(x))
            if ii == 0:
                sample[3].append(f(x))
            else:
                sample[3].append(sample[3][ii - 1] + f(x))
        if overwrite == 1:
            sample[1][ii] = x
            sample[2][ii] = f(x)
            if ii == 0:
                sample[3][ii] = f(x)
            else:
                sample[3][ii] = sample[3][ii - 1] + f(x)
    return sample

def f(x):
    return (x)/(1+(x**2))
```

Una vez ejecutado este procedimiento, la estructura `sample` tendrá guardado los datos de evaluación de cada individuo de la población.

```
# Ordenamos
sample = JoulSort(sample, qi)
sample = evaluate(qe, x_min, x_max, sample, 1)
Sorting(PEC + 1, qi, sample)
```

Posteriormente se ordena la población. Nuevamente se presenta la invocación a la función:

```
def JoulSort(sample, qi, order_by = 2):
    ijs = 0
    while ijs < qi - 1:
        if sample[2][ijs] < sample[2][ijs + 1]:
            for auxijs in range(4):
                auxjs = sample[auxijs][ijs]
                sample[auxijs][ijs] = sample[auxijs][ijs + 1]
                sample[auxijs][ijs + 1] = auxjs
            ijs = 0
            continue
        ijs = ijs + 1
    return sample
```

Debido a que ahora se necesitará guardar la población en una nueva estructura, se creará una nueva muestra vacía. A continuación las instrucciones que lo realizan:

```
# Creamos una nueva muestra vacía
newSample = newSampleCreator(qi)
ShowNullNewSample(PEC + 1, qi, newSample)
```

```
def newSampleCreator(qi):
    nullCreator = []
    for ii in range(qi):
        nullCreator.append(0)
    nullNewSample = []
    nullNewSample.append(nullCreator[:])
    nullNewSample.append(nullCreator[:])
    nullNewSample.append(nullCreator[:])
    nullNewSample.append(nullCreator[:])
    return nullNewSample
```

Se implementó por gusto personal y enriquecimiento al experimentar el elitismo en el algoritmo:

```
# Buscamos los qelitism mejores
ii = 0
while ii < qelitism:
    newSample[0][ii] = sample[0][ii]
    newSample[1][ii] = sample[1][ii]
    newSample[2][ii] = sample[2][ii]
    newSample[3][ii] = sample[3][ii]
    ii = ii + 1
ShowNewSampleWithElit(PEC + 1, qi, newSample, qelitism, pc)
```

Luego se aplicará cruzamiento:

```

# Aplicamos cruzamiento
while ii < qi:
    random_num = random.randint(0,99)

    if random_num < pc:
        CrossingTitle(PEC + 1, random_num, pc, 1)

        # Seleccionamos por ruleta
        # Padre 1
        random_pick = random.uniform(0, sample[3][ii])

        for ic in range(qi):
            if random_pick < sample[3][ic]:
                c1 = sample[0][ic]
                RouletteTitle(PEC + 1, random_pick, c1, 1)
                break

```

```

# Padre 2
random_pick = random.uniform(0, sample[3][ii])

for ic in range(qi):
    if random_pick < sample[3][ic]:
        c2 = sample[0][ic]
        RouletteTitle(PEC + 1, random_pick, c2, 2)
        break

# Seleccionamos aleatoriamente un punto de corte
random_cut = random.randint(1, qe - 1)
CutTitle(PEC + 1, random_cut, ii, c1, c2, qe)
newSample[0][ii] = c1[:random_cut] + c2[random_cut:]
try:
    newSample[0][ii + 1] = c2[:random_cut] + c1[random_cut:]
except:
    pass

```

Por último, para esta generación, se aplicará mutación:

```

# Aplicamos mutación
ii = 0
while ii < qi:
    random_num = random.randint(0,99)

    if random_num < pm:
        random_cut = random.randint(0, qe - 1)
        MutationTitle(PEC + 1, random_num, ii, pm, 1, random_cut)

        if newSample[0][random_cut] == '1':
            newSample[0][random_cut] = '0'
        if newSample[0][random_cut] == '0':
            newSample[0][random_cut] = '1'

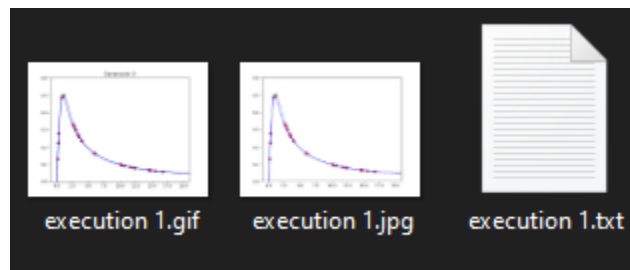
    else:
        MutationTitle(PEC + 1, random_num, ii, pm, 0, random_cut)

    ii = ii + 1

```

La nueva población se ha generado completamente. El siguiente paso es restaurar las variables para repetir según la cantidad de generaciones deseada.

Cabe destacar que todo este procedimiento se reporta en la carpeta de executions donde por cada ejecución se reporta su procedimiento por texto y dos imágenes de apreciación:



El documento de texto describe paso a paso (incluyendo elecciones de probabilidad) los pasos realizados a lo largo de todo el procedimiento (véase anexo 6):

```
execution 1.txt: Bloc de notas
Archivo Edición Formato Ver Ayuda

EJECUCIÓN DEL CRUZAMIENTO
Se realizará la ejecución del cruzamiento con una probabilidad del 40%:

El cruzamiento se ha activado (Probabilidad: 2 < 40)

Aplicamos selección por ruleta para el primer padre obteniendo 0.11446212978823349
Por lo tanto, el cromosoma a escoger será 000011011110111101

Aplicamos selección por ruleta para el segundo padre obteniendo 0.49665548419974664
Por lo tanto, el cromosoma a escoger será 000011011110111101

Se ha seleccionado aleatoriamente un punto de corte entre el gen 8 y 9
Serán los cromosomas 1 y 2 de la nueva generación

(0)(0)(0)(0)(1)(0)(1)[1][1][0][1][1][1][1][0][1]
[0][0][0][0][1][1][0][1](1)(1)(0)(1)(1)(1)(0)(1)

El cruzamiento no se ha activado (Probabilidad: ! 74 < 40 !)

Aplicamos selección por ruleta para el primer padre obteniendo 0.29206309592916363
Por lo tanto, el cromosoma a escoger será 000011011110111101

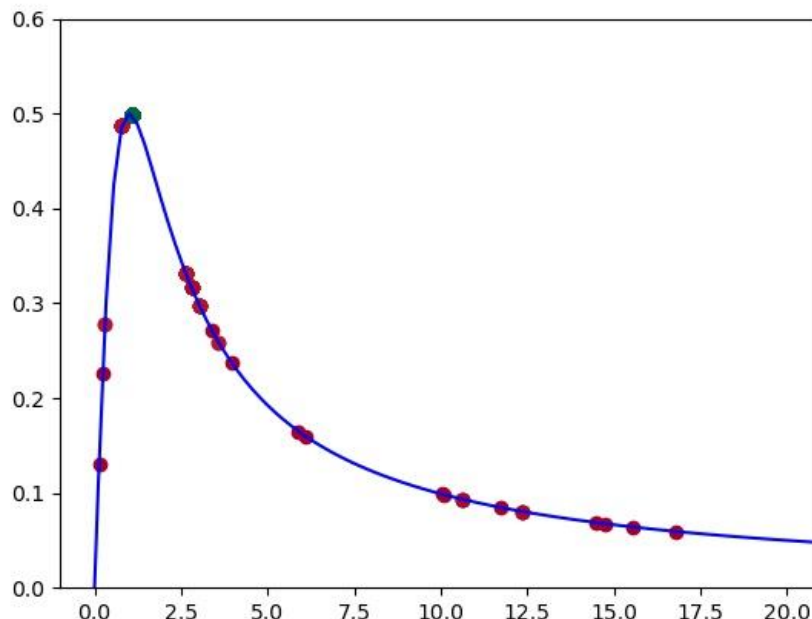
Aplicamos selección por ruleta para el segundo padre obteniendo 1.1986934767585358
Por lo tanto, el cromosoma a escoger será 001000011011000100

Línea 1, columna 1 100% Windows (CRLF) ANSI
```

El algoritmo también genera una visualización creada en matplotlib para revisar el procedimiento del algoritmo a través de las generaciones con el siguiente mapa de colores:



Donde la tendencia a rojo indicaba las primeras generaciones mientras que la tendencia a verde son las últimas generaciones del algoritmo genético:



Por último, debido a la rápida convergencia del algoritmo, no es posible apreciar que la gran mayoría de puntos generados están acumulados en la misma coordenada del punto verde observable, es por ello que también se adjuntó una animación GIF que retrata mejor este proceso.

Se dejan adjuntos al entregable cinco ejecuciones de cada modo de personalización como evidencia en una carpeta externa al proyecto.

Obtención y Procesamiento de Datos

Se realizaron cinco ejecuciones con una población base del documento con las indicaciones del examen y después se ejecutaron otras cinco veces con una población aleatoria realizada por el programa y comparar el rendimiento del algoritmo.

En el primer caso, la primera ejecución ya había conseguido el máximo para toda su población en la quinta generación, la segunda y tercera ejecución lo consiguieron

en la sexta generación, la tercera ejecución. La cuarta ejecución estuvo muy cerca de conseguir el récord en la cuarta generación, pero los operadores del algoritmo genético llevaron a esta ejecución a llegar al máximo en la sexta generación y por último la quinta ejecución llegó a la cúspide en la quinta generación.

Sin embargo, no fue hasta el segundo caso en el que se descubrió que podía mejorar el caso anterior, pero personalmente, deduzco que se estancó en un punto al concentrarse todos los puntos en una misma representación; el cruzamiento no causaba efecto.

Para este caso, la primera ejecución logró la cúspide de la población en la séptima generación. La segunda ejecución lo consiguió en la sexta generación. Con el récord de progresión más lento, la tercera ejecución llegó al óptimo en toda la población en la octava generación. La cuarta lo consigue en la sexta generación y la última llega a la cúspide del óptimo en la séptima generación.

Conclusión:

Con el desarrollo de este examen puedo aprender diferentes perspectivas y aplicaciones de un algoritmo genético. Es muy interesante la forma en la que se modela el problema para usar número binarios que puedan realizar la expresión de número de un intervalo dado y un gran planteamiento de las fórmulas que nos permiten hacer estas representaciones.

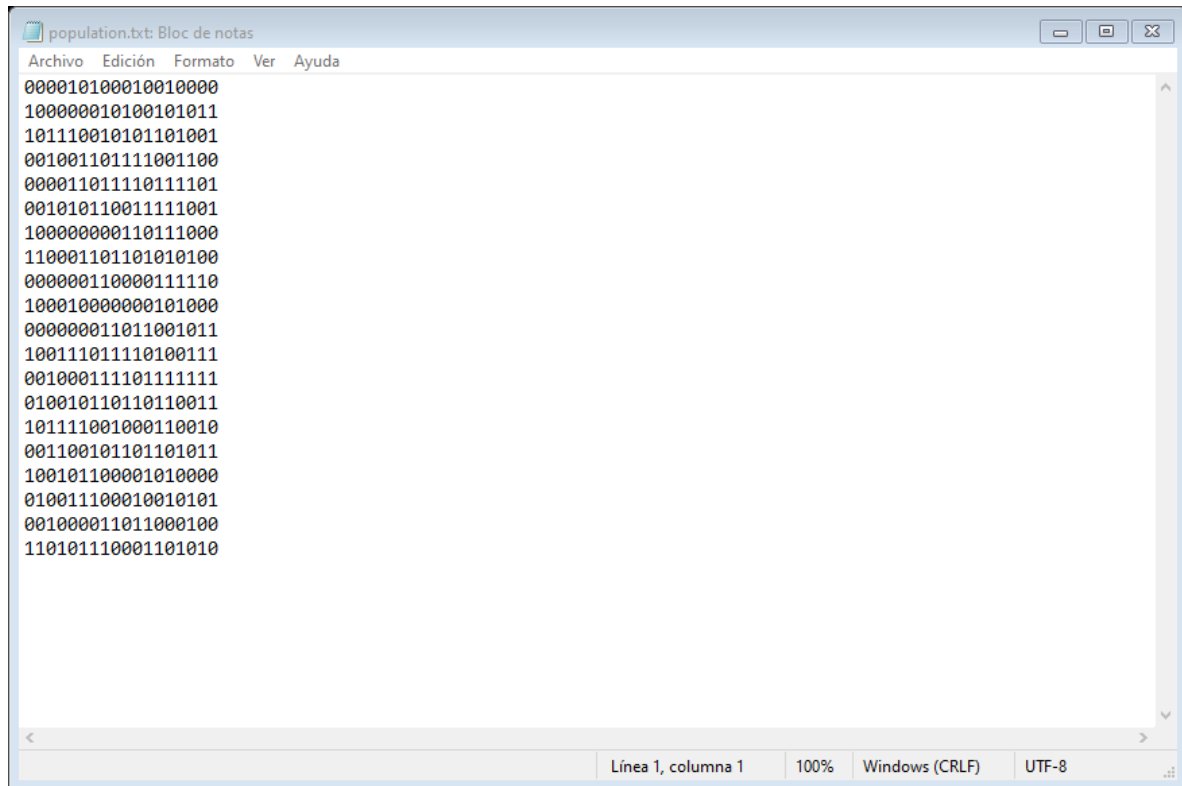
Concluyo que el algoritmo genético es una herramienta muy útil para optimizar procesos, que aunque no consiga siempre óptimo definitivo, consigue muy buenos resultados, pues también podríamos probar e investigar cuáles son los parámetros de probabilidad de cruzamiento, mutación, elitismo y otras variables que mejoran el rendimiento del algoritmo para un problema en específico.

Referencias:

- Imageio Development Team. (2020). *Imageio Documentation*. Septiembre 14, 2021, de Read the Docs Sitio web: <https://imageio.readthedocs.io/en/stable/>
- Luna, F. (2021). *Instrucciones: Primera Evaluación Parcial*. Aguascalientes: Editorial Universidad Autónoma de Aguascalientes.
- Matplotlib Development Team. (2012). *Matplotlib Documentation*. Septiembre 13, 2021, de Matplotlib Sitio web: <https://matplotlib.org/>
- Numpy Development Team. (2020). *Numpy Documentation*. Septiembre 11, 2021, de Numpy Sitio web: <https://numpy.org/doc/>
- Python Software Foundation. (2021). *Python Documentation*. Septiembre 11, 2021, de Python Sitio web: <https://docs.python.org/3/>

Anexos:

Anexo 1: Archivo population.txt



```
population.txt: Bloc de notas
Archivo  Edición  Formato  Ver  Ayuda
000010100010010000
100000010100101011
101110010101101001
001001101111001100
000011011110111101
001010110011111001
100000000110111000
110001101101010100
000000110000111110
10001000000101000
000000011011001011
100111011110100111
001000111101111111
010010110110110011
101111001000110010
001100101101101011
100101100001010000
010011100010010101
001000011011000100
110101110001101010
```

Línea 1, columna 1 100% Windows (CRLF) UTF-8

Anexo 2: Código en Python de main.py

```
### Portada
'''
        Centro de Ciencias Básicas
    Departamento de Ciencias de la Computación
        Metaheurísticas I
            7º "A"

        Primera Evaluación Parcial

        Profesor: Francisco Javier Luna Rosas

        Alumno: Joel Alejandro Espinoza Sánchez

Fecha de Entrega: Aguascalientes, Ags., 15 de septiembre de 2021
'''

### Descripción
'''
Archivo: main
Función: Supervisa el flujo principal del programa

Se utilizarán las siguientes variables:
    set_customization: Modo de personalización del programa
        Si está en 0, usará los parámetros del archivo
        Si está en 1, permitirá edición libre ignorando el archivo
    qe: Cantidad de elementos por individuo
    qi: Cantidad de individuos
    pc: Probabilidad de cruzamiento
    pm: Probabilidad de mutación
    qelitism: Cantidad de individuos a heredar a la siguiente generación por
    elitismo
    qg: Cantidad de generaciones
    x_min: El intervalo inferior de x
    x_max: El intervalo superior de x
    accuracy: La precisión requerida
    ii: Iterador de los individuos
    random_num: Número aleatorio
    rand_pick: Número aleatorio
    ic: Iterador principal de cruzamiento
    jc: Iterador secundario de cruzamiento
    ig: Iterador de generaciones
    PEC: PreviousExecutionCounter

Con las variables anteriores, se utilizarán las siguientes estructuras:
    sample: En búsqueda de la mayor similitud con el programa en C aquí sample y
    results se fusionan en la misma estructura
'''

### Importación de archivos
import random

### Configuración del programa
stream = open("set_program.py", encoding="utf-8")
```

```

read_file = stream.read()
exec(read_file)

# Ya se tienen las variables
set_customization, qe, qi, pc, pm, qelitism, qg, x_min, x_max, accuracy, sample =
set_values()
ig = 0
finals = []

%% Preparación y exportación del primer trozo del archivo: La portada del archivo
stream = open("exporting_functions.py", encoding="utf-8")
read_file = stream.read()
exec(read_file)

PEC = PreviousExecutionCounter()
FirstExport(PEC + 1, set_customization, qe, qi, pc, pm, qelitism, qg, x_min,
x_max, accuracy, sample)

%% Ejecución del Algoritmo Genético Simple

# Cargamos el programa de functions
stream = open("functions.py", encoding="utf-8")
read_file = stream.read()
exec(read_file)

while ig <= qg:
    # Cargamos la población al archivo
    ShowPopulation(PEC + 1, qi, sample, ig)

    # Evaluamos a todas las muestras junto con sus evaluaciones acumuladas
    sample = evaluate(qe, x_min, x_max, sample, 0)
    finals.append([sample[1], sample[2]])
    Evaluation(PEC + 1, qi, sample)

    # Ordenamos
    sample = JoulSort(sample, qi)
    sample = evaluate(qe, x_min, x_max, sample, 1)
    Sorting(PEC + 1, qi, sample)

    # Creamos una nueva muestra vacía
    newSample = newSampleCreator(qi)
    ShowNullNewSample(PEC + 1, qi, newSample)

    # Buscamos los qelitism mejores
    ii = 0
    while ii < qelitism:
        newSample[0][ii] = sample[0][ii]
        newSample[1][ii] = sample[1][ii]
        newSample[2][ii] = sample[2][ii]
        newSample[3][ii] = sample[3][ii]
        ii = ii + 1
    ShowNewSampleWithElit(PEC + 1, qi, newSample, qelitism, pc)

    # Aplicamos cruzamiento
    while ii < qi:

```

```

random_num = random.randint(0,99)

if random_num < pc:
    CrossingTitle(PEC + 1, random_num, pc, 1)

    # Seleccionamos por ruleta
    # Padre 1
    random_pick = random.uniform(0, sample[3][ii])

    for ic in range(qi):
        if random_pick < sample[3][ic]:
            c1 = sample[0][ic]
            RouletteTitle(PEC + 1, random_pick, c1, 1)
            break

    # Padre 2
    random_pick = random.uniform(0, sample[3][ii])

    for ic in range(qi):
        if random_pick < sample[3][ic]:
            c2 = sample[0][ic]
            RouletteTitle(PEC + 1, random_pick, c2, 2)
            break

    # Seleccionamos aleatoriamente un punto de corte
    random_cut = random.randint(1, qe - 1)
    CutTitle(PEC + 1, random_cut, ii, c1, c2, qe)
    newSample[0][ii] = c1[:random_cut] + c2[random_cut:]
    try:
        newSample[0][ii + 1] = c2[:random_cut] + c1[random_cut:]
    except:
        pass

else:
    CrossingTitle(PEC + 1, random_num, pc, 0)

    # Seleccionamos por ruleta
    # Padre 1
    random_pick = random.uniform(0, sample[3][ii])

    for ic in range(qi):
        if random_pick < sample[3][ic]:
            c1 = sample[0][ic]
            RouletteTitle(PEC + 1, random_pick, c1, 1)
            break

    # Padre 2
    random_pick = random.uniform(0, sample[3][ii])

    for ic in range(qi):
        if random_pick < sample[3][ic]:
            c2 = sample[0][ic]
            RouletteTitle(PEC + 1, random_pick, c2, 2)
            break

```

```

        # No se activó el cruzamiento
        NoCutTitle(PEC + 1, ii, c1, c2, qe)
        newSample[0][ii] = c1
        try:
            newSample[0][ii + 1] = c2
        except:
            pass

    ii = ii + 2

ShowPostCrossing(PEC + 1, qi, newSample, pm)

# Aplicamos mutación
ii = 0
while ii < qi:
    random_num = random.randint(0,99)

    if random_num < pm:
        random_cut = random.randint(0, qe - 1)
        MutationTitle(PEC + 1, random_num, ii, pm, 1, random_cut)

        if newSample[0][random_cut] == '1':
            newSample[0][random_cut] = '0'
        if newSample[0][random_cut] == '0':
            newSample[0][random_cut] = '1'

    else:
        MutationTitle(PEC + 1, random_num, ii, pm, 0, random_cut)

    ii = ii + 1

# Transformamos newSample en sample y se repetiría el procedimiento
sample = newSample
ig = ig + 1

%% Exportación de resultados a formatos visibles
sample = evaluate(qe, x_min, x_max, sample, 1)
#finals.append([sample[1], sample[2]])
FinalResults(PEC + 1, qi, sample)
CreateGraph(PEC + 1, x_min, x_max, finals, qg)
CreateGIF(PEC + 1, x_min, x_max, finals, qg)

```

Anexo 3: Código en Python de set_program.py

```
### Portada
'''
        Centro de Ciencias Básicas
    Departamento de Ciencias de la Computación
        Metaheurísticas I
            7º "A"

        Primera Evaluación Parcial

        Profesor: Francisco Javier Luna Rosas

        Alumno: Joel Alejandro Espinoza Sánchez

Fecha de Entrega: Aguascalientes, Ags., 15 de septiembre de 2021
'''

### Descripción
'''
Archivo: set_program
Función: Obtiene los valores con los que se calibrará el Algoritmo Genético

Se utilizarán las siguientes variables:
    values_file: Variable que guardará el contenido del archivo "values.txt"
    population_file: Variable que guardará el contenido del archivo
"population.txt"
    line_counter: Iterador de archivo para contar los individuos dentro de
"population.txt"
    length_aux: Guarda la primera línea de "population.txt"
    length_counter: Guarda un número de identificación para obtener el número de
elementos por individuo
    set_customization: Modo de personalización del programa
        Si está en 0, usará los parámetros del archivo
        Si está en 1, permitirá edición libre ignorando el archivo
    qe: Cantidad de elementos por individuo
    qi: Cantidad de individuos
    pc: Probabilidad de cruzamiento
    pm: Probabilidad de mutación
    qelitism: Cantidad de individuos a heredar a la siguiente generación por
elitismo
    qg: Cantidad de generaciones
    x_min: El intervalo inferior de x
    x_max: El intervalo superior de x
    accuracy: La precisión requerida
    ii: Iterador para los individuos
    ie: Iterador para los elementos de cada individuo
    individual: Variable usada para la construcción aleatoria de individuos
    gen: Número 0 o 1 insertado a individual
    L:
    input_aux:

Con las variables anteriores, se utilizarán las siguientes estructuras:
    sample: Matriz donde se guarda la población
'''
```

```

#%%
import random
import math

#%%
def set_values():
    sample = [[], [], [], []]
    values_file = open("files/values.txt", "r")

    # Se obtiene el modo de personalización
    set_customization = int(get_value(values_file.readline()))

    # La personalización se hará parcialmente por values.txt junto con
    population.txt
    if set_customization == 0:
        # Se ignoran dos renglones en values.txt
        pc = float(get_value(values_file.readline()))
        pc = float(get_value(values_file.readline()))

        # Se abre population.txt
        population_file = open("files/population.txt", "r")

        # Se obtendrá qe contando la longitud de un renglón
        length_aux = population_file.readline()
        length_counter = length_aux.find("\n")
        if length_counter != -1:
            qe = length_counter
        else:
            qe = len(length_aux)

        # Reset: Se cierra y se abre el archivo nuevamente para lectura completa
        population_file.close()
        population_file = open("files/population.txt", "r")

        # Se obtendrá qi contando la cantidad de renglones
        qi = 0
        for line_counter in population_file:
            if line_counter != "\n":
                qi = qi + 1

        # Se formará la población del archivo population.txt
        length_counter = line_counter.find("\n")
        if length_counter != -1:
            sample[0].append(line_counter[:length_counter])
        else:
            sample[0].append(line_counter)

        population_file.close()

    # La personalización se hará enteramente por values.txt
    if set_customization == 1:
        qe = int(get_value(values_file.readline()))
        qi = int(get_value(values_file.readline()))

```

```

pc = float(get_value(values_file.readline()))
pm = float(get_value(values_file.readline()))
qelitism = int(get_value(values_file.readline()))
qg = int(get_value(values_file.readline()))
x_min = float(get_value(values_file.readline()))
x_max = float(get_value(values_file.readline()))
accuracy = float(get_value(values_file.readline()))

values_file.close()

# Se sugerirá un qe según los valores de x_min, x_max y accuracy
L = round(math.log((1 + ((x_max - x_min)/(accuracy))), 2) + 0.00000001)

# Lo anterior implica que si L y qe no son iguales, se pedirá al usuario
elegir
if (qe != L) and (set_customization == 1):
    print("Para conseguir una precisión de " + str(accuracy) + " es
recomendable usar una cantidad de elementos por individuo de " + str(L) + " y
usted especificó una cantidad de " + str(qe))
    print("¿Desea cambiar el valor?")
    print("0: No, haré los cálculos con el valor " + str(qe) + " que es el
que especifiqué en el archivo")
    print("1: Acepto la recomendación y tomaré " + str(L) + " para hacer los
cálculos con este valor")
    input_aux = int(input())
    if input_aux == 1:
        qe = L

# Ahora se construirá una población aleatoria
if set_customization == 1:
    for ii in range(qi):
        individual = ""
        for ie in range(qe):
            gen = random.randint(0, 1)
            individual = individual + str(gen)
        sample[0].append(individual)

    return set_customization, qe, qi, pc, pm, qelitism, qg, x_min, x_max,
accuracy, sample

# Función para obtener los valores de cada renglón del archivo values.txt
def get_value(line):
    initial_position = line.find(":")
    if line[initial_position + 1] == " ":
        initial_position = initial_position + 1
    final_position = line.find("\n")
    return line[initial_position + 1:final_position]

```

Anexo 4: Código en Python de exporting_functions.py

```
### Portada
'''
        Centro de Ciencias Básicas
    Departamento de Ciencias de la Computación
        Metaheurísticas I
        7° "A"

        Primera Evaluación Parcial

        Profesor: Francisco Javier Luna Rosas

        Alumno: Joel Alejandro Espinoza Sánchez

Fecha de Entrega: Aguascalientes, Ags., 15 de septiembre de 2021
'''

### Descripción
'''
Archivo: exporting_functions
Función: Se encarga de exportar el procedimiento a archivos txt e imagen

Se utilizarán las siguientes variables:
    iPEC: Iterador para PreviousExecutionCounter
    total: Total de ejecuciones existentes en la carpeta
    iFE: Iterador para FirstExport
    iE: Iterador para Evaluation
'''

### Importación de archivos
import os
import matplotlib.pyplot as plt
import numpy as np
import imageio

### Determina el número de ejecuciones que ya hay en la carpeta
def PreviousExecutionCounter():
    for iPEC in os.walk("executions"):
        total = len(iPEC[2])

    total = int(total/3)
    return total

### Comienza un nuevo archivo para la nueva ejecución
def FirstExport(exec_num, set_customization, qe, qi, pc, pm, qelitism, qg, x_min,
x_max, accuracy, sample):
    file = open("executions/execution " + str(exec_num) + ".txt", 'a')

    file.write("                Centro de Ciencias Básicas\n")
    file.write("        Departamento de Ciencias de la Computación\n")
    file.write("                Metaheurísticas I\n")
    file.write("                7° "A"\n")
    file.write("\n")
    file.write("                Primera Evaluación Parcial\n")
```



```

file.write("\n")
file.write("                Profesor: Francisco Javier Luna Rosas\n")
file.write("\n")
file.write("                Alumno: Joel Alejandro Espinoza Sánchez\n")
file.write("Fecha de Entrega: Aguascalientes, Ags., 15 de septiembre de
2021\n")

```

```

file.write("_____
_\n")
file.write("\n")
file.write("Ejecución número " + str(exec_num) + "\n")
file.write("Algoritmo configurado con las siguientes variables:\n")
file.write("                Modo de personalización del programa: " +
str(set_customization) + "\n")
file.write("    Cantidad de elementos por individuo: " + str(qe) + "\n")
file.write("    Cantidad de individuos: " + str(qi) + "\n")
file.write("    Probabilidad de cruzamiento: " + str(pc) + "%\n")
file.write("    Probabilidad de mutación: " + str(pm) + "%\n")
file.write("    Cantidad de individuos a heredar por elitismo: " + str(qelitism)
+ "\n")
file.write("    Cantidad de generaciones: " + str(qg) + "\n")
file.write("    Intervalo inferior en x: " + str(x_min) + "\n")
file.write("    Intervalo superior en x: " + str(x_max) + "\n")
file.write("    Precisión: " + str(accuracy) + "\n")

```

```

file.write("_____
_\n")
file.write("\n")

file.close()
return

```

```

### Se muestra la población
def ShowPopulation(exec_num, qi, sample, ig):
    file = open("executions/execution " + str(exec_num) + ".txt", 'a')

    file.write("POBLACIÓN " + str(ig) + "\n")
    file.write("La población " + str(ig) + " es la siguiente:\n")
    file.write("\n")

    for iFE in range(qi):
        file.write(sample[0][iFE] + "\n")

```

```

file.write("_____
_\n")
file.write("\n")

file.close()
return

```

```

### Se realiza una evaluación de cada miembro (la población no está ordenada)
def Evaluation(exec_num, qi, sample):
    file = open("executions/execution " + str(exec_num) + ".txt", 'a')

```

```

file.write("EVALUACIÓN DE LOS INDIVIDUOS\n")
file.write("Al evaluar a los individuos, tenemos los siguientes datos:\n")
file.write("\n")
file.write(" Cadena binaria      \t\tx          \tf(x)          f(x)
acumulada\n")

for iE in range(qi):
    file.write(str(sample[0][iE]) + "      \t" + str(sample[1][iE]) + "      \t"
+ str(sample[2][iE]) + "      \t" + str(sample[3][iE]) + "\n")

file.write("_\n")
_ = \n")
    file.write("\n")

file.close()
return

### Se ordena a la población para hacer elitismo
def Sorting(exec_num, qi, sample):
    file = open("executions/execution " + str(exec_num) + ".txt", 'a')

    file.write("ORDENAMIENTO DE LOS INDIVIDUOS\n")
    file.write("Al ordenar la población de forma descendente (en función de f(x))
obtenemos lo siguiente:\n")
    file.write("\n")
    file.write(" Cadena binaria      \t\tx          \tf(x)          f(x)
acumulada\n")

    for iE in range(qi):
        file.write(str(sample[0][iE]) + "      \t" + str(sample[1][iE]) + "      \t"
+ str(sample[2][iE]) + "      \t" + str(sample[3][iE]) + "\n")

file.write("_\n")
_ = \n")
    file.write("\n")

file.close()
return

### Se prepara la estructura para llevar ahí la nueva población
def ShowNullNewSample(exec_num, qi, sample):
    file = open("executions/execution " + str(exec_num) + ".txt", 'a')

    file.write("INICIALIZACIÓN DE LA NUEVA POBLACIÓN\n")
    file.write("La estructura para la nueva población ha sido creada:\n")
    file.write("\n")
    file.write("Cadena binaria      x      f(x)      f(x) acumulada\n")

    for iE in range(qi):
        file.write("      " + str(sample[0][iE]) + "\t\t" + str(sample[1][iE])
+ "      \t" + str(sample[2][iE]) + "      \t      " + str(sample[3][iE]) + "\n")

```

[illegible]

```

%% Se reporta la selección por ruleta
def RouletteTitle(exec_num, random_pick, cross, parent):
    file = open("executions/execution " + str(exec_num) + ".txt", 'a')

    if parent == 1:
        file.write("      Aplicamos selección por ruleta para el primer padre
obteniendo " + str(random_pick) + "\n")
    if parent == 2:
        file.write("      Aplicamos selección por ruleta para el segundo padre
obteniendo " + str(random_pick) + "\n")
    file.write("      Por lo tanto, el cromosoma a escoger será " + str(cross)
+ "\n")
    file.write("\n")

    file.close()
    return

%% Se reporta el punto de corte
def CutTitle(exec_num, random_cut, ii, c1, c2, qe):
    file = open("executions/execution " + str(exec_num) + ".txt", 'a')
    file.write("      Se ha seleccionado aleatoriamente un punto de corte entre
el gen " + str(random_cut) + " y " + str(random_cut + 1) + "\n")
    file.write("      Serán los cromosomas " + str(ii + 1) + " y " + str(ii + 2)
+ " de la nueva generación\n")
    file.write("\n")

    file.write("      ")
    for i in range(qe):
        if i < random_cut:
            file.write("(" + str(c1[i]) + ")")
        else:
            file.write "[" + str(c2[i]) + "]"
    file.write("\n")

    file.write("      ")
    for i in range(qe):
        if i < random_cut:
            file.write "[" + str(c2[i]) + "]"
        else:
            file.write("(" + str(c1[i]) + ")")
    file.write("\n")

    file.write("\n")

    file.close()
    return

%% Se reporta si no hubo punto de corte
def NoCutTitle(exec_num, ii, c1, c2, qe):
    file = open("executions/execution " + str(exec_num) + ".txt", 'a')
    file.write("      No hay punto de corte\n")
    file.write("      Serán los cromosomas " + str(ii + 1) + " y " + str(ii + 2)
+ " de la nueva generación\n")
    file.write("\n")

```

```

        file.write("      ")
        for i in range(qe):
            file.write("(" + str(c1[i]) + ")")
        file.write("\n")

        file.write("      ")
        for i in range(qe):
            file.write("[ " + str(c2[i]) + "]")
        file.write("\n")

        file.write("\n")

        file.close()
        return

### Se muestra a la población al pasar el cruzamiento
def ShowPostCrossing(exec_num, qi, sample, pm):
    file = open("executions/execution " + str(exec_num) + ".txt", 'a')

    file.write("_____
    _\n")
    file.write("\n")
    file.write("CRUZAMIENTO FINALIZADO\n")
    file.write("La nueva generación ha pasado la etapa de cruzamiento
    exitosamente:\n")

    for iE in range(qi):
        file.write(str(sample[0][iE]) + "\n")

    file.write("_____
    _\n")
    file.write("\n")

    file.write("EJECUCIÓN DE MUTACIONES\n")
    file.write("Se realizará la ejecución de mutaciones con una probabilidad del
    " + str(int(pm)) + "%:\n")
    file.write("\n")

    file.close()
    return

### Se reporta la activación de mutación
def MutationTitle(exec_num, random_num, ii, pm, activated, random_cut):
    file = open("executions/execution " + str(exec_num) + ".txt", 'a')

    if activated == 1:
        file.write("  La mutación se ha activado para el cromosoma " + str(ii +
        1) + " (Probabilidad: " + str(random_num) + " < " + str(int(pm)) + ")\n")
        file.write("  El gen a mutar será el gen número " + str(random_cut + 1)
        + "\n")
    if activated == 0:

```



```
    file.write("Otra buena forma de visualización del algoritmo es con la  
ejecución visual adjunta a este procedimiento\n")
```

```
    file.close()  
    return
```

```
### Finalmente se exportará un gráfico de evolución del algoritmo
```

```
def CreateGraph(exec_num, x_min, x_max, finals, qg):  
    # Se hace un espacio dividido en 100 separaciones  
    x = np.linspace(-1,21,100)
```

```
    # Se indica la función  
    y = x/(1+(x**2))
```

```
    ...  
    # Fijando al centro los ejes  
    fig = plt.figure()  
    ax = fig.add_subplot(1, 1, 1)  
    ax.spines['left'].set_position(0)  
    ax.spines['bottom'].set_position('zero')  
    ax.spines['right'].set_color('none')  
    ax.spines['top'].set_color('none')  
    ax.xaxis.set_ticks_position('bottom')  
    ax.yaxis.set_ticks_position('left')  
    ...
```

```
    plt.xlim(-1, 21)  
    plt.ylim(0, 0.6)
```

```
    # Dibujamos la función  
    plt.plot(x,y, 'blue')
```

```
    # Distribuimos los colores  
    cmap = plt.get_cmap("RdYlGn")  
    colors = [cmap(i) for i in np.linspace(0,1,qg)]  
    for i, color in enumerate(colors, start = 0):  
        plt.scatter(finals[i][0], finals[i][1], color = color)
```

```
    # show the plot  
    plt.show()
```

```
    plt.savefig("executions/execution " + str(exec_num) + ".jpg")  
    return
```

```
### Finalmente se exportará un gráfico de evolución del algoritmo
```

```
def CreateGIF(exec_num, x_min, x_max, finals, qg):  
    # Se hace un espacio dividido en 100 separaciones  
    x = np.linspace(-1,21,100)
```

```
    # Se indica la función  
    y = x/(1+(x**2))
```

```
    plt.xlim(-1, 21)  
    plt.ylim(0, 0.6)
```

```
    # Dibujamos la función
```

```

plt.plot(x,y, 'blue')

frames = []

# Distribuimos los colores
cmap = plt.get_cmap("RdYlGn")
colors = [cmap(i) for i in np.linspace(0,1,len(finals))]
for i in range(len(colors)):
    plt.scatter(finals[i][0], finals[i][1], color = colors[i])
    plt.title(label = "Generación " + str(i))
    plt.show()
    plt.savefig("temporal/" + str(i) + ".jpg")
    frames.append("temporal/" + str(i) + ".jpg")
    plt.clf()

# Se hace un espacio dividido en 100 separaciones
x = np.linspace(-1,21,100)

# Se indica la función
y = x/(1+(x**2))

plt.xlim(-1, 21)
plt.ylim(0, 0.6)

# Dibujamos la función
plt.plot(x,y, 'blue')
plt.close()

...

# Obtenemos en una lista las imágenes
images = os.listdir("temporal")
frames = []
for i in range(len(finals)):
    frames.append("temporal/" + images[i])
...

# Construimos el GIF
with imageio.get_writer("executions/execution " + str(exec_num) + ".gif", mode
= "I", duration = 0.25) as writer:
    for filename in frames:
        image = imageio.imread(filename)
        writer.append_data(image)

# Eliminamos temporal
for i in range(len(frames)):
    os.remove(frames[i])

```


Anexo 5: Código en Python de functions.py

```
### Portada
'''
        Centro de Ciencias Básicas
    Departamento de Ciencias de la Computación
        Metaheurísticas I
            7º "A"

        Primera Evaluación Parcial

        Profesor: Francisco Javier Luna Rosas

        Alumno: Joel Alejandro Espinoza Sánchez

Fecha de Entrega: Aguascalientes, Ags., 15 de septiembre de 2021
'''

### Descripción
'''
Archivo: functions
Función: Realiza otras funciones de apoyo para el Algoritmo Genético

Se utilizarán las siguientes variables:
    qi:
    qe:
    x_min:
    x_max:
    ii:
    x:
    bin_ent:
    overwrite: Variable para sobrescribir o crear espacios en sample
        Si overwrite = 0 no se sobrescribe, se crean espacios
        Si overwrite = 1 se sobrescriben los espacios
    ijs: Iterador de JoulSort
    auxijs: Auxiliar Iterador de JoulSort
    auxjs: Auxiliar de JoulSort
    nullCreator:
    nullNewSample:

Con variables anteriores, se utilizarán las siguientes estructuras:
    sample:
'''

### Funciones para evaluación de muestras
def evaluate(qe, x_min, x_max, sample, overwrite = 0):
    for ii in range(len(sample[0])):
        bin_ent = int(sample[0][ii], 2)
        x = x_min + (bin_ent * ((x_max - x_min) / ((2**qe) - 1)))
        if overwrite == 0:
            sample[1].append(x)
            sample[2].append(f(x))
            if ii == 0:
                sample[3].append(f(x))
        else:
```

```

        sample[3].append(sample[3][ii - 1] + f(x))
    if overwrite == 1:
        sample[1][ii] = x
        sample[2][ii] = f(x)
        if ii == 0:
            sample[3][ii] = f(x)
        else:
            sample[3][ii] = sample[3][ii - 1] + f(x)
    return sample

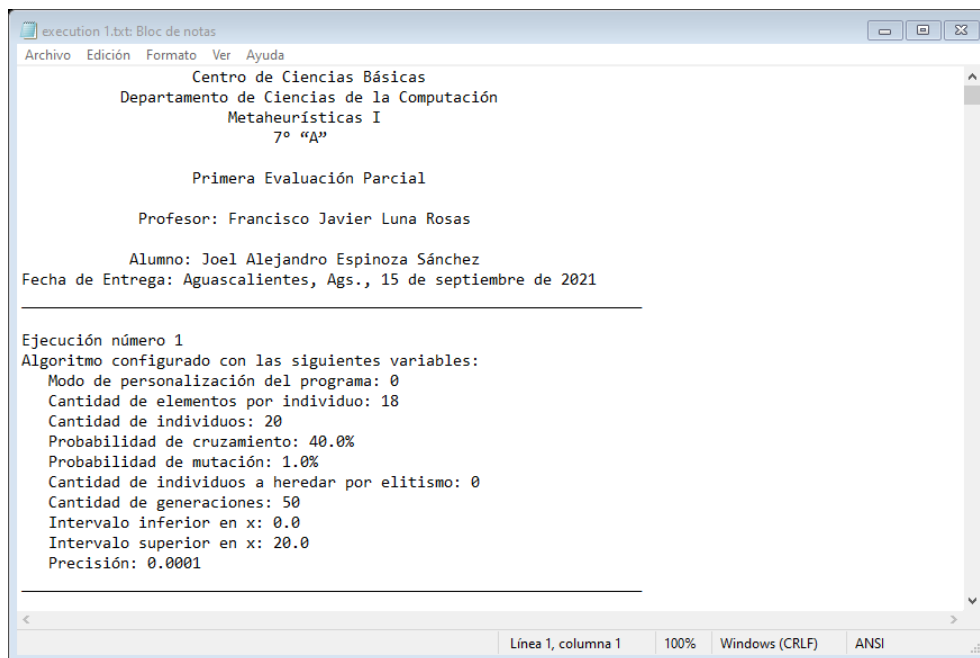
def f(x):
    return (x)/(1+(x**2))

### Funciones para ordenar en la preparación para el elitismo
def JoulSort(sample, qi, order_by = 2):
    ijs = 0
    while ijs < qi - 1:
        if sample[2][ijs] < sample[2][ijs + 1]:
            for auxijs in range(4):
                auxjs = sample[auxijs][ijs]
                sample[auxijs][ijs] = sample[auxijs][ijs + 1]
                sample[auxijs][ijs + 1] = auxjs
            ijs = 0
            continue
        ijs = ijs + 1
    return sample

### Funciones para crear la nueva muestra vacía
def newSampleCreator(qi):
    nullCreator = []
    for ii in range(qi):
        nullCreator.append(0)
    nullNewSample = []
    nullNewSample.append(nullCreator[:])
    nullNewSample.append(nullCreator[:])
    nullNewSample.append(nullCreator[:])
    nullNewSample.append(nullCreator[:])
    return nullNewSample

```

Anexo 6: Algunas capturas de pantalla del resultado de una ejecución en formato de texto (tomando como base la ejecución 1 incluida en el entregable)



```
execution 1.txt: Bloc de notas
Archivo Edición Formato Ver Ayuda

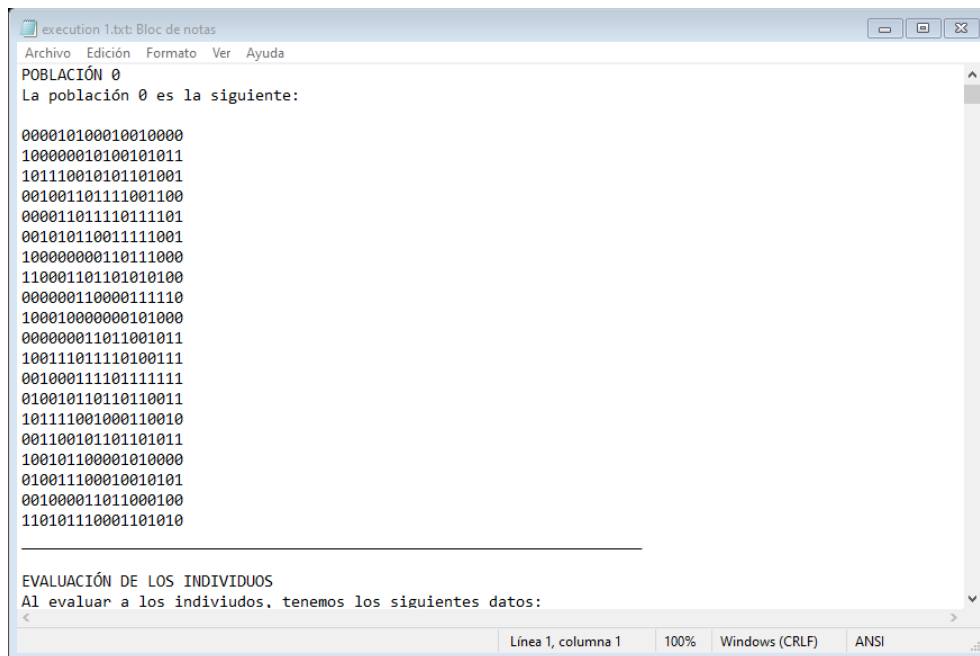
Centro de Ciencias Básicas
Departamento de Ciencias de la Computación
Metaheurísticas I
7º "A"

Primera Evaluación Parcial

Profesor: Francisco Javier Luna Rosas

Alumno: Joel Alejandro Espinoza Sánchez
Fecha de Entrega: Aguascalientes, Ags., 15 de septiembre de 2021

Ejecución número 1
Algoritmo configurado con las siguientes variables:
Modo de personalización del programa: 0
Cantidad de elementos por individuo: 18
Cantidad de individuos: 20
Probabilidad de cruzamiento: 40.0%
Probabilidad de mutación: 1.0%
Cantidad de individuos a heredar por elitismo: 0
Cantidad de generaciones: 50
Intervalo inferior en x: 0.0
Intervalo superior en x: 20.0
Precisión: 0.0001
```



```
execution 1.txt: Bloc de notas
Archivo Edición Formato Ver Ayuda

POBLACIÓN 0
La población 0 es la siguiente:

000010100010010000
100000010100101011
101110010101101001
001001101111001100
000011011110111101
001010110011111001
100000000110111000
110001101101010100
000000110000111110
100010000000101000
000000011011001011
100111011110100111
001000111101111111
010010110110110011
101111001000110010
001100101101101011
100101100001010000
010011100010010101
001000011011000100
110101110001101010

EVALUACIÓN DE LOS INDIVIDUOS
Al evaluar a los individuos, tenemos los siguientes datos:
```

```
execution 1.txt: Bloc de notas
Archivo Edición Formato Ver Ayuda

EVALUACIÓN DE LOS INDIVIDUOS
Al evaluar a los individuos, tenemos los siguientes datos:

Cadena binaria      x      f(x)      f(x) acumulada
000010100010010000 0.7922393502782832 0.48674018731881935 0.48674018731881935
100000010100101011 10.100975421811759 0.09803944865776479 0.5847796359765841
101110010101101001 14.48072235382978 0.06872956304739561 0.6535091990239797
001001101111001100 3.0429193226597695 0.2965993896898081 0.9501085887137879
000011011110111101 1.0886424585054721 0.4982020655859669 1.4483106542997548
001010110011111001 3.3783850798991386 0.2721544253128324 1.720465079612587
100000000110111000 10.03360761113018 0.09868480139484212 1.8191498810074291
110001101101010100 15.533811698195258 0.06411001320082109 1.8832598942082501
000000110000111110 0.23910613672690095 0.22617530022308982 2.10943519443134
10001000000101000 10.628092300767138 0.0932645936637194 2.2026997880950594
000000011011001011 0.13267567701597982 0.13038060863948173 2.333080396734541
100111011110100111 12.337006900813677 0.08052785144307553 2.4136082481776167
001000111101111111 2.80266877238759 0.3165086263576767 2.7301168745352933
010010110110110011 5.892585344640139 0.16495416362924842 2.8950710381645415
101111001000110010 14.730433389409596 0.06757523802084509 2.9626462761853865
001100101101101011 3.973022358025963 0.23670208236043294 3.1993483585458193
100101100001010000 11.724898242562265 0.08467266559446128 3.2840210241402805
010011100010010101 6.105141087116573 0.1595166653827065 3.443537689522987
001000011011000100 2.6321511541410603 0.3319977566619838 3.775535446184971
110101110001101010 16.805026264290863 0.059296041179834606 3.8348314873648053

Línea 1, columna 1 100% Windows (CRLF) ANSI
```

```
execution 1.txt: Bloc de notas
Archivo Edición Formato Ver Ayuda

RESULTADOS FINALES
El algoritmo genético ha terminado con los siguientes resultados:

Cadena binaria      x      f(x)      f(x) acumulada
000011011110111101 1.0886424585054721 0.4982020655859669 0.4982020655859669
000011011110111101 1.0886424585054721 0.4982020655859669 0.9964041311719338
000011011110111101 1.0886424585054721 0.4982020655859669 1.4946061967579007
000011011110111101 1.0886424585054721 0.4982020655859669 1.9928082623438677
000011011110111101 1.0886424585054721 0.4982020655859669 2.4910103279298346
000011011110111101 1.0886424585054721 0.4982020655859669 2.9892123935158015
000011011110111101 1.0886424585054721 0.4982020655859669 3.4874144591017684
000011011110111101 1.0886424585054721 0.4982020655859669 3.9856165246877353
000011011110111101 1.0886424585054721 0.4982020655859669 4.483818590273702
000011011110111101 1.0886424585054721 0.4982020655859669 4.982020655859669
000011011110111101 1.0886424585054721 0.4982020655859669 5.480222721445636
000011011110111101 1.0886424585054721 0.4982020655859669 5.978424787031603
000011011110111101 1.0886424585054721 0.4982020655859669 6.47662685261757
000011011110111101 1.0886424585054721 0.4982020655859669 6.974828918203537
000011011110111101 1.0886424585054721 0.4982020655859669 7.473030983789504
000011011110111101 1.0886424585054721 0.4982020655859669 7.971233049375471
000011011110111101 1.0886424585054721 0.4982020655859669 8.469435114961438
000011011110111101 1.0886424585054721 0.4982020655859669 8.967637180547406
000011011110111101 1.0886424585054721 0.4982020655859669 9.465839246133374
000011011110111101 1.0886424585054721 0.4982020655859669 9.964041311719342

Línea 1, columna 1 100% Windows (CRLF) ANSI
```