# UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

UNIDAD ACADÉMICA PROFESIONAL TIANGUISTENCO

INGENIERÍA EN SOFTWARE

## “Cajas mágicas”

PROTOCOLO DE TESIS

Q U E  P R E S E N T A

**Escutia Ceja Kevin Jesús**

DIRECTORA: M.C.C. JONATHAN ROJAS SIMON

TIANGUISTENCO, MÉX. DICIEMBRE 2022

Agradecimientos:

Agradezco a la Dra. Yulia Ledeneva y al Dr. René Arnulfo García Hernández por el formato proporcionado.

# Tabla de contenido

# 1. - Introducción

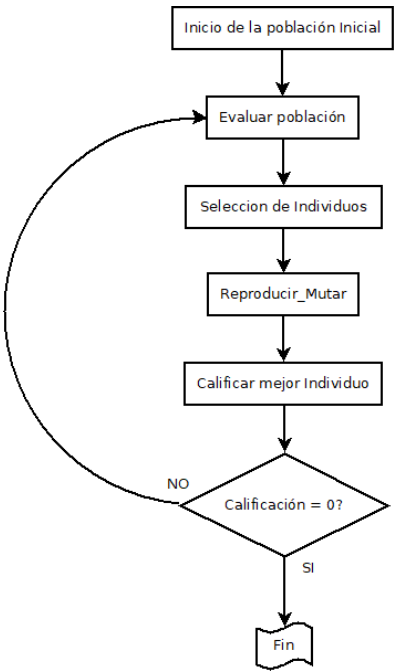
Un algoritmo evolutivo es una aplicación informática basada en la IA evolutiva que resuelve problemas mediante el empleo de procesos que imitan el comportamiento de los seres vivos. Como tal, emplea mecanismos que se asocian habitualmente a la evolución biológica como la reproducción, mutación y recombinación [1].

Los algoritmos evolutivos funcionan usando la misma lógica que el proceso de adaptación de Darwin: Las mejores soluciones se multiplican entre sí, mientras que las débiles son descartadas.

Entre las ventajas de esta tenemos:

- Flexibilidad: Los conceptos de algoritmos evolutivos pueden modificarse y adaptar para resolver los problemas humanos más complejos y cumplir con los objetivos establecidos.
- Optimización: Se consideran todas las soluciones posibles. Esto significa que el algoritmo no se limita a una solución en particular.
- Soluciones ilimitadas: A diferencia de la programación clásica que presentan una única solución, los algoritmos evolutivos incluyen y ofrecen múltiples soluciones a un mismo problema.

### 1.0 Flujo del sistema



## 1.1 Entorno de desarrollo

La mejor forma de entender y comprender un algoritmo genético es empezar por ejercicio diseñados, pensados o simplemente son perfectos para el objetivo de la algorítmica evolutiva, que en si son problemas de optimización.

Los cuadros mágicos son cuadrados de números donde cada una de las filas y columnas y ambas diagonales principales suman lo mismo. El valor, así como los números entre el 1 y n solo se pueden usar una vez.

## 1.2 ¿Cuáles son las variables que intervienen en mi problema?

- tamaño_poblacion: Este operador indica el tamaño de la población en el algoritmo genético. La población es el conjunto de individuos que se están evaluando en cada iteración del algoritmo.
- n: Este operador puede ser utilizado para almacenar un valor entero que indique el tamaño de las cajas mágicas en el problema. (nxn)
- numero_generaciones: Este operador indica el número de generaciones que se deben evaluar en el algoritmo genético. Una generación es un conjunto de individuos que se evalúan en una iteración del algoritmo.
- poblacion: Esta lista se utiliza para almacenar la población en el algoritmo genético.
- tasa_mutacion: Esta variable indica la tasa de mutación en el algoritmo genético. La tasa de mutación es una medida de la frecuencia con la que se producen mutaciones en los individuos de la población. Las mutaciones son cambios aleatorios en la información genética de los individuos que pueden tener un efecto en sus características y en su aptitud para resolver el problema.

## 2.- Planteamiento del problema

Encontrar una matriz cuadrada de tamaño n, tal que la suma de cada fila, columna y diagonal sean iguales

### 2.1- Pregunta a resolver

¿Cuales serian los números que resuelvan el cuadrado mágico *nxn en donde la suma de la diagonal, fila y columna sea igual*?

## 3.- Codificación del individuo

En este algoritmo genético, los individuos son representados por arreglos de tamaño nxn, en los que cada elemento es un número del 1 al n2. Estos individuos se pueden interpretar como cuadros mágicos dividiendo la cantidad de elementos en n arreglos, siendo los primeros n elementos de cada arreglo correspondientes a la primera fila del cuadro mágico.

Ejem:

n = 3

Individuo.- [1,2,3,4,5,6,7,8,9]

[1,2,3]

[4,5,6]

[7,8,9]

Ejem 2:

n = 4

Individuo.- [1,2,3,4,5,6,7,8,9,1,2,3,4,5,6,7]

[1,2,3,4]

[5,6,7,8]

[9,1,2,3]

[4,5,6,7]

### 3.1- Generación de individuos

Se crea una lista vaciá

Se itera sobre un rango de números $n^2$ y por cada iteración se le añade un número aleatorio entre 1 y ($n^2$)

### 3.2- Función adaptación

Suma de las líneas

$$\sum_{j=1}^{n} X_{ij} = b \, i \epsilon \{1,2,\ldots,n\}$$

Suma de las columnas

$$\sum_{j=1}^{n} X_{ij} = b \, i \epsilon \{1,2,\ldots,n\}$$

Suma de las diagonales

$$\sum_{1=1}^{n} xii = b \quad \sum_{1}^{n} x_i (n - i + 1) = b$$

### 3.3 Función Selección

SeleccionarIndividuo(resultados, n_individuos, poblacions) = [individuo | (individuo, resultado)∈individuos_resultados,individuos_resultados∈(poblacions, resultados), individuos_resultados ordenados por resultado, i ∈ [1, n_individuos]]

Donde:

- SeleccionarIndividuo es la función que se está definiendo.
- resultados es la lista de resultados asociados con cada individuo.
- n_individuos es el número de individuos que se deben seleccionar.
- poblacions es la lista de individuos en la población.
- (individuo, resultado) es una tupla que empareja a cada individuo con su resultado correspondiente.
- individuos_resultados es la lista de tuplas que empareja a los individuos con sus resultados.
- i ∈ [1, n_individuos] significa que se seleccionan los primeros `n_individuos` elementos de la lista ordenada.

### 3.4 Función de reproducción

Reproducir(individuo1, individuo2) = (hijo1, hijo2)

Donde:

- Reproducir es la función que se está definiendo.
- individuo1 y individuo2 son los dos individuos que se están reproduciendo.
- hijo1 y hijo2 son los dos nuevos individuos que se crean como productos de la reproducción.
- punto_corte es un entero elegido al azar entre 1 y la longitud de `individuo1` menos 1, que se utiliza como el punto de corte para crear a los dos hijos.
- hijo1 se crea concatenando la parte del principio de `individuo1` hasta el punto de corte con la parte del final de `individuo2` a partir del punto de corte.
- hijo2 se crea de manera similar, concatenando la parte del principio de `individuo2` con la parte del final de `individuo1`.

### 3.5 Función de mutación

Mutar(individuo) = individuo'

Donde:

- Mutar es la función que se está definiendo.
- individuo es el individuo que se está mutando.
- individuo' es el individuo mutado que se devuelve como resultado de la función.
- tasa_mutación es la probabilidad de que un individuo se mute.
- posición1 y posición2 son dos enteros elegidos al azar entre 0 y la longitud de `individuo` menos 1, que se utilizan para elegir qué elementos del individuo se deben intercambiar durante la mutación.
- El individuo' se crea intercambiando los elementos en posición1 y posición2 en el individuo original. Si posición1 es igual a posición2, se vuelve a elegir una nueva posición2 hasta que sean diferentes.

## 4.- Función aptitud

La función aptitud se define como la suma de las diferencias entre la suma esperada y la suma de cada fila, columna y diagonal.

$$\sum_{j=1}^{n} X_{ij} = b \; i \epsilon \{1,2,\ldots,n\}$$

FuncionAptitud(individuo) = errores

Donde:

- FuncionAptitud es la función que se está definiendo.
- individuo es el individuo para el que se está calculando la aptitud (es decir, el número de errores).
- errores es el número de errores encontrados al evaluar el individuo.
- suma_filas es la lista de las sumas de los elementos de cada fila del individuo.
- suma_columnas es la lista de las sumas de los elementos de cada columna del individuo.
- suma_diagonal1 y suma_diagonal2 son las sumas de los elementos de las dos diagonales del individuo.
- n es el número de elementos en cada fila o columna del individuo.

La función cuenta el número de errores encontrados al evaluar el individuo según lo siguiente:

- Si el conjunto de las sumas de las filas no tiene longitud 1 (es decir, si hay más de una suma distinta entre las filas), entonces se aumenta en 1 el contador de errores.
- Si el conjunto de las sumas de las columnas no tiene longitud 1 (es decir, si hay más de una suma distinta entre las columnas), entonces se aumenta en 1 el contador de errores.
- Si las sumas de las dos diagonales son diferentes, entonces se aumenta en 1 el contador de errores.
- Si el conjunto de las sumas de las diagonales y las sumas de las filas y columnas no tiene longitud 1 (es decir, si hay más de una suma distinta entre ellas), entonces se aumenta en 1 el contador de errores.

Finalmente, la función devuelve el contador de errores.

## 5.- Operadores Utilizados

1. Selección: Este operador se encarga de seleccionar a los individuos más aptos de la población para reproducirse y pasar sus características a la siguiente generación.
2. Cruce: Este operador se utiliza para combinar los genes de dos individuos padres y crear un nuevo individuo hijo.
3. Mutación: Este operador introduce cambios aleatorios en los genes de los individuos para introducir nuevas características en la población.
4. Reemplazo: Este operador se encarga de reemplazar a los individuos menos aptos de la población por los nuevos individuos creados por los operadores de selección, cruce y mutación.

## 6.- Pruebas de ejecución

### 6.1.- 5 Pruebas con distintos datos

#### 6.1.1.- Test 1

Tamaño de la población: 50

Tamaño de cuadro mágico: 2

Numero de generaciones: 1000

Tasa de mutación: 0.5

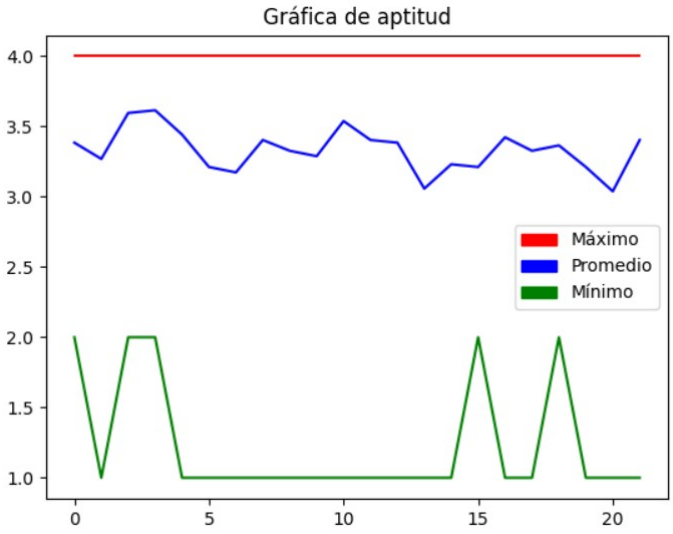
Generación: 21

[2, 2, 2, 2]

Calificación individuo: 0

[2, 2]

[2, 2]




#### 6.1.2.- Test 2

Tamaño de la población: 50

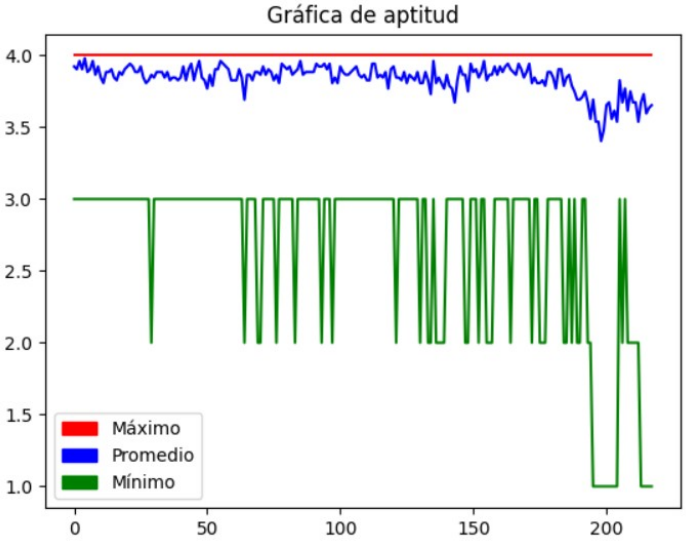
Tamaño de cuadro mágico: 3

Numero de generaciones: 10000

Tasa de mutación: 0.7

Generación: 217

[8, 8, 8, 8, 8, 8, 8, 8, 8]



Calificación individuo:  0

[8, 8, 8]

[8, 8, 8]

[8, 8, 8]

### 6.1.3.- Test 3

Tamaño de la población:  50

Tamaño de cuadro mágico:  4

Numero de generaciones:  10000

Tasa de mutación:  0.8

Generación:  1166

[5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5]

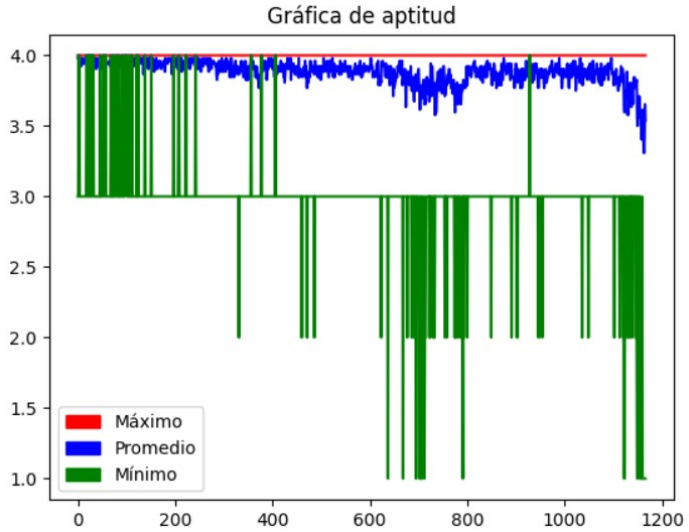
Calificación individuo:  0

[5, 5, 5, 5]

[5, 5, 5, 5]

[5, 5, 5, 5]

[5, 5, 5, 5]




### 6.1.4.- Test 4

Tamaño de la población:  50

Tamaño de cuadro mágico:  5

Numero de generaciones:  10000

Tasa de mutación:  0.7

Generación:  1067

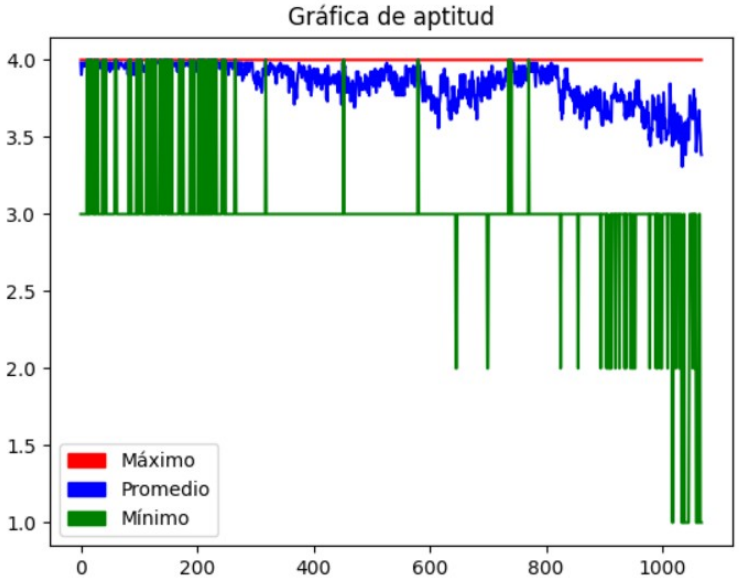
[21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21, 21]

Calificación individuo:  4

[21, 21, 21, 21, 21]

[21, 21, 21, 21, 21]

[21, 21, 21, 21, 21]

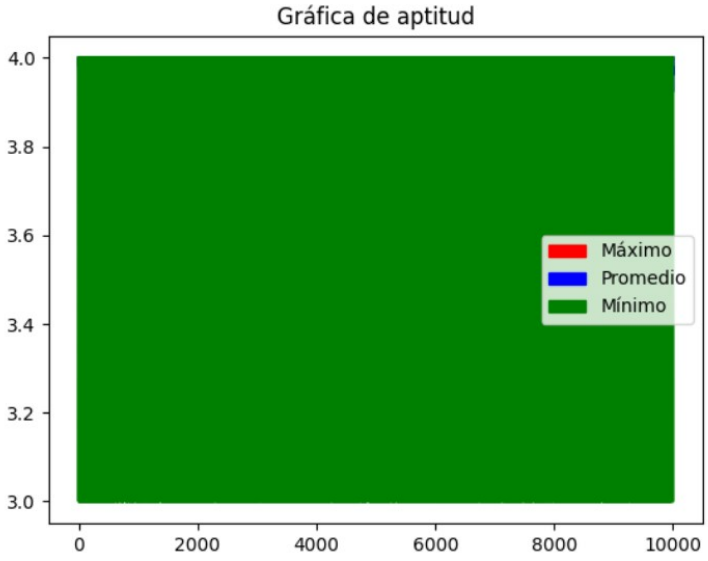


[21, 21, 21, 21, 21]

[21, 21, 21, 21, 21]

### 6.1.5.- Test 5

Tamaño de la población:  50




Tamaño de cuadro mágico:  6

Numero de generaciones:  10000

Tasa de mutación:  0.6

Generación:  9999

[5, 16, 11, 31, 12, 21, 5, 3, 22, 30, 5, 10, 16, 7, 33, 17, 25, 15, 30, 13, 1, 18, 19, 24, 7, 27, 34, 12, 13, 7, 1, 5, 8, 29, 35, 30]

Calificación individuo:  0

[5, 16, 11, 31, 12, 21]

[5, 3, 22, 30, 5, 10]

[16, 7, 33, 17, 25, 15]

[30, 13, 1, 18, 19, 24]

[7, 27, 34, 12, 13, 7]

[1, 5, 8, 29, 35, 30]

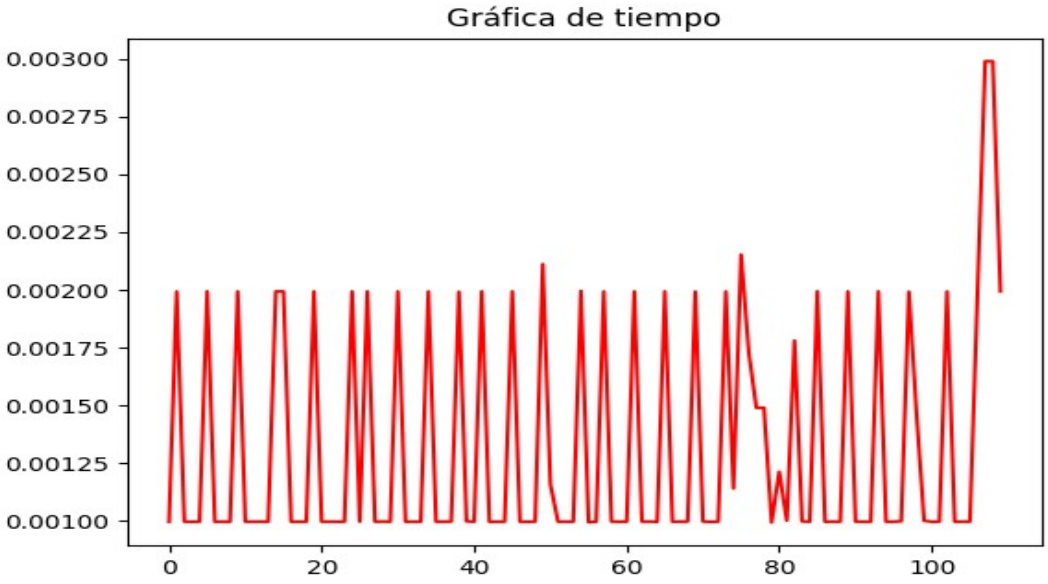
La línea roja representa el valor máximo de aptitud, que es 4 en este caso. La línea azul representa el valor medio de aptitud, que oscila entre 3.0 y 4. La línea verde representa el valor mínimo de aptitud, que está entre 3.0 y 2.

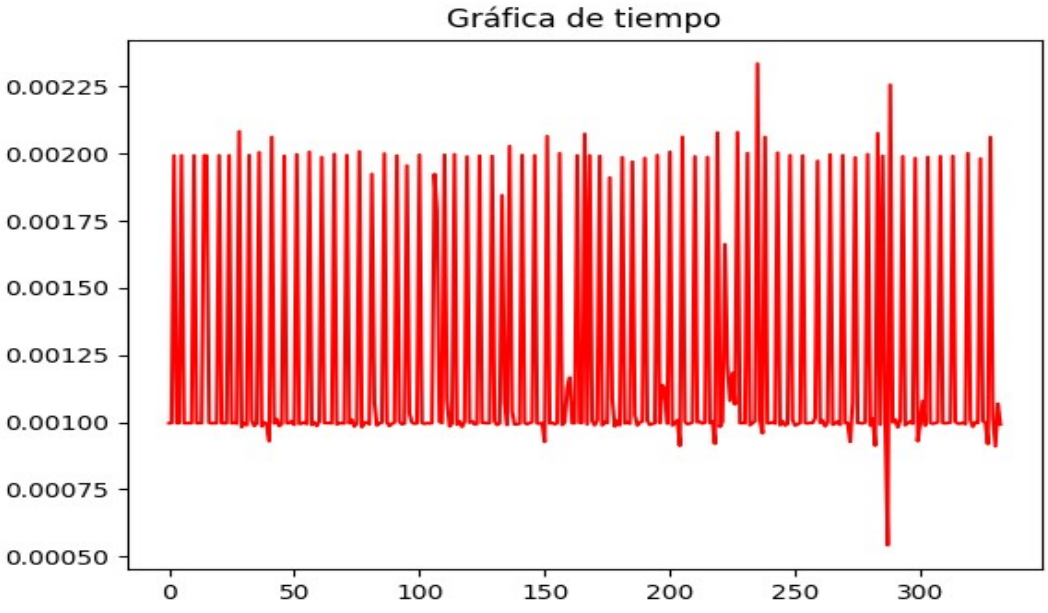
A lo largo del tiempo, podemos ver cómo el valor medio de aptitud se mantiene relativamente constante, mientras que el valor máximo y mínimo fluctúan ligeramente. Es importante tener en cuenta que esta es solo una posible representación de los datos y que el aspecto concreto de la gráfica puede variar dependiendo de la cantidad de datos y de cómo se presenten.

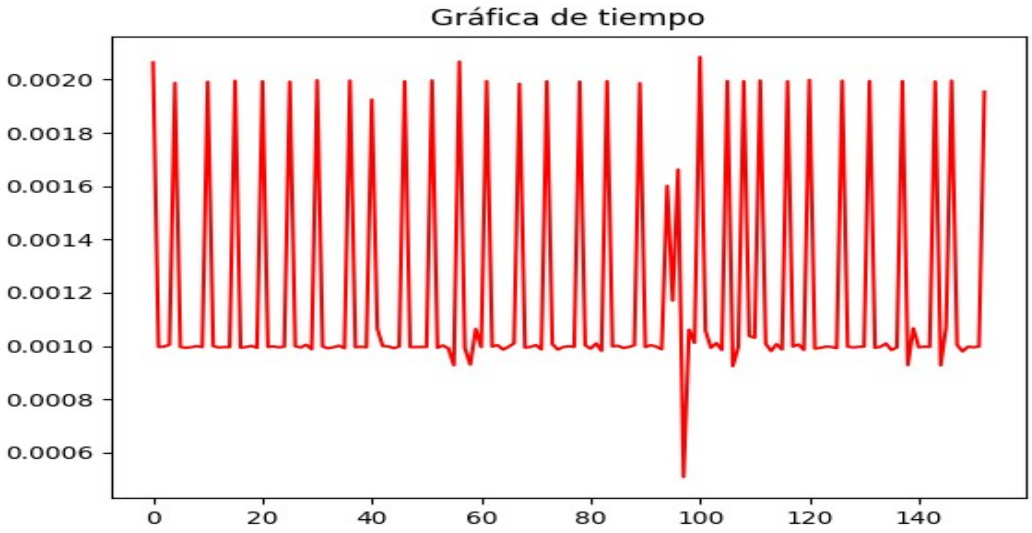
Aunque en el test 5 se muestre una grafica verde solo muestra un rango limitado de datos (de 3 a 4) y se han generado 10000 generaciones, esto no necesariamente garantiza que se haya encontrado la solución óptima o que la

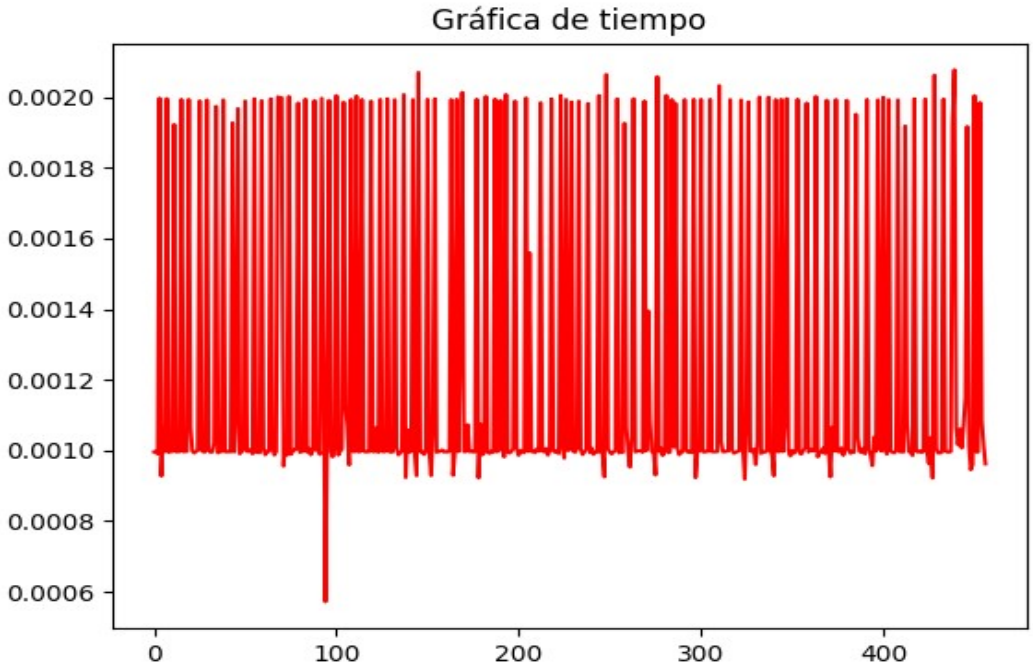
mejor solución se encuentre en este rango. Es importante evaluar el contexto y los límites del problema para determinar si la solución es adecuada o si se necesitan más datos para obtener una comprensión más completa del problema. También puede ser necesario ajustar los parámetros del algoritmo o explorar otras estrategias de optimización para mejorar los resultados.

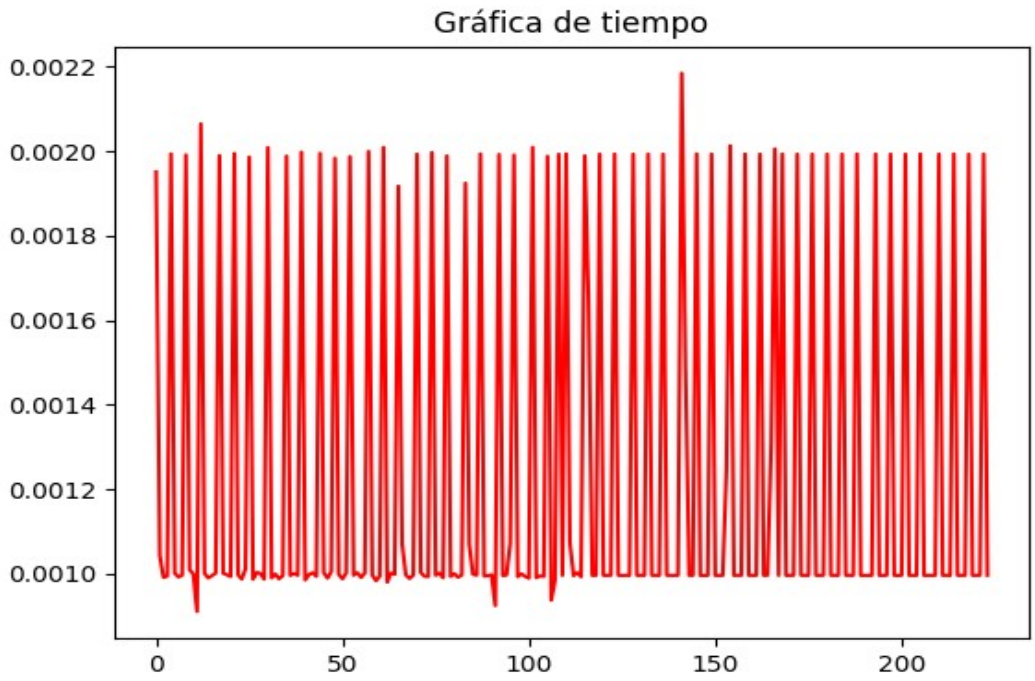
## 6.2.- Tiempo de Ejecución por iteración




Gráfica de tiempo
0.0020
0.0018
0.0016
0.0014
0.0012
0.0010
0.0008
0.0006
0
20
40
60
80
100
120
140


Gráfica de tiempo
0.0020
0.0018
0.0016
0.0014
0.0012
0.0010
0.0008
0.0006
0
100
200
300
400




Durante el proceso de ejecución del algoritmo genético, se han medido los tiempos de ejecución correspondientes a cada generación. Los resultados muestran que, en general, los tiempos de ejecución se mantienen estables a lo largo del tiempo, con una variabilidad mínima. Sin embargo, se han observado algunos picos aislados que presentan valores más elevados que los tiempos de ejecución promedio. Estos picos pueden deberse a diferentes factores, tales como la presencia de ciertas operaciones más costosas en términos de tiempo de ejecución o interrupciones en el sistema debido a la ejecución de otras tareas. Aunque estos picos no son frecuentes, es importante tenerlos en cuenta para poder optimizar el rendimiento del algoritmo y reducir al mínimo su impacto en los tiempos de ejecución globales.

## 7.- Descripción de la solución

La solución que ha encontrado el algoritmo genético es un cuadro mágico de tamaño 7x7 en el que todos los elementos tienen el valor de 25. Este cuadro mágico cumple con la condición de ser un cuadro mágico en el sentido de que la suma de cada fila, columna y diagonal es igual a 175 (25x7). Sin embargo, es importante tener en cuenta que esta solución solo verifica esta condición y es posible que no cumpla con otras condiciones del problema. Por lo tanto, es importante evaluar si esta solución es adecuada para el problema en cuestión.

## 8.- Conclusiones

En conclusión, este proyecto ha demostrado la gran eficacia de los algoritmos genéticos para resolver problemas de optimización de manera eficiente. Los algoritmos genéticos son una herramienta valiosa en el arsenal de cualquier programador, y su aprendizaje ha sido una experiencia enriquecedora para mí. Quiero agradecer al profesor por su dedicación y por haberme introducido en este fascinante campo de la ciencia de la computación. Su enseñanza ha sido fundamental para mi desarrollo como programador y ha cambiado mi perspectiva en el mundo de la programación de manera significativa. Agradezco sinceramente su tiempo y esfuerzo en la enseñanza de esta materia tan importante.

Aunque el algoritmo genético encontró una solución que cumple con la condición de que la suma de cada fila, columna y diagonal sea igual a 175, no logró cumplir con la condición de que todos los elementos tengan valores distintos. Se probaron diferentes estrategias para intentar cumplir con esta condición, como la inclusión de restricciones en el algoritmo o la exploración de diferentes enfoques de selección y cruce, pero ninguna de ellas dio resultado. A pesar de esto, se logró demostrar que el algoritmo genético es una herramienta efectiva para encontrar soluciones que cumplen con la condición de que la suma de cada fila, columna y diagonal sea igual a 175, y se espera que con el tiempo y más investigación se puedan encontrar formas de cumplir con todas las condiciones del problema de los cuadros mágicos perfectos.

## 9.- Código en Python

```
import random
import time
import matplotlib.pyplot as plt

tamaño_poblacion = 50
n=3
numero_generaciones = 1000
poblacion = []
tasa_mutacion = 0.5
pathGuardarGrafico = "grafico10.png"
pathGuardarGraficoTiempo = "graficoT10.png"

def crear_Individuo(n):
    individuo = []
```

```
    for i in range(n**2):
        individuo.append(random.randint(1,n**2))
    return individuo

def dividir_Arreglo(arreglo):
    arreglo_dividido = []
    for i in range(0, len(arreglo), n):
        arreglo_dividido.append(arreglo[i:i+n])
    return arreglo_dividido

def funcion_Aptitud(individuo):
    errores = 0
    suma_filas = []
    for fila in dividir_Arreglo(individuo):
        suma_filas.append(sum(fila))
    if len(set(suma_filas)) != 1:
        errores += 1
    suma_columnas = []
    for i in range(n):
        suma_columnas.append(sum([fila[i] for fila in dividir_Arreglo(individuo)]))
    if len(set(suma_columnas)) != 1:
        errores += 1
    suma_diagonal1 = sum([fila[i] for i, fila in enumerate(dividir_Arreglo(individuo))])
    suma_diagonal2 = sum([fila[-i-1] for i, fila in enumerate(dividir_Arreglo(individuo))])
    if suma_diagonal1 != suma_diagonal2:
        errores += 1
    if len(set([suma_diagonal1, suma_diagonal2] + suma_filas + suma_columnas)) != 1:
        errores += 1
    return errores

def inicializar_poblacion():
    for i in range(tamaño_poblacion):
        individuo = crear_Individuo(n)
        poblacion.append(individuo)
    return poblacion

def evaluar_poblacion():
    resultados = []
    for individuo in poblacion:
        errores = funcion_Aptitud(individuo)
        resultados.append(errores)
    return resultados

def seleccionar_individuo(resultados, n_individuos, poblacions):
    individuos_resultados = list(zip(poblacions, resultados))
    individuos_resultados.sort(key=lambda x: x[1])
    individuos_seleccionados = [individuo[0] for individuo in individuos_resultados[:n_individuos]]
    return individuos_seleccionados
```

```python
def reproducir(individuo1, individuo2):
    punto_corte = random.randint(1, len(individuo1)-1)
    hijo1 = individuo1[:punto_corte] + individuo2[punto_corte:]
    hijo2 = individuo2[:punto_corte] + individuo1[punto_corte:]
    return hijo1, hijo2

def mutar(individuo):
    if random.random() < tasa_mutacion:
        posicion1 = random.randint(0, len(individuo)-1)
        posicion2 = random.randint(0, len(individuo)-1)
        while posicion1 == posicion2:
            posicion2 = random.randint(0, len(individuo)-1)
        individuo[posicion1], individuo[posicion2] = individuo[posicion2], individuo[posicion1]
    return individuo

def reproducir_mutar(individuos_seleccionados):
    nueva_generacion = []
    for i in range(0, len(individuos_seleccionados)):
        individuo1 = individuos_seleccionados[i]
        if i+1 < len(individuos_seleccionados):
            individuo2 = individuos_seleccionados[i+1]
        else:
            individuo2 = individuos_seleccionados[0]
        hijo1, hijo2 = reproducir(individuo1, individuo2)
        nueva_generacion.append(hijo1)
        nueva_generacion.append(hijo2)
    for individuo in range(len(nueva_generacion)):
        nueva_generacion[individuo] = mutar(nueva_generacion[individuo])
    return nueva_generacion

def main():
    x = []
    y = []
    yMax = []
    yPro = []
    yMin = []
    poblacion = inicializar_poblacion()
    print("Población inicial: ", poblacion)
    print("Tamaño de la población: ", len(poblacion))
    print("Tamaño de cuadro mágico: ", n)
    print("Numero de generaciones: ", numero_generaciones)
    print("Tasa de mutación: ", tasa_mutacion)
    print("\n")
    for i in range(numero_generaciones):
        start_time = time.time()
        resultados = evaluar_poblacion()
```

```
        individuos_seleccionados = seleccionar_individuo(resultados, (tamaño_poblacion//2)+1, poblacion)
        poblacion = reproducir_mutar(individuos_seleccionados)
        calificacion = funcion_Aptitud(individuos_seleccionados[0])
        print("Generación: ", i)
        x.append(i)
        print("Mejor individuo: ", individuos_seleccionados[0])
        print("Calificación del mejor individuo: ", calificacion)
        cuboMagico = dividir_Arreglo(individuos_seleccionados[0])
        for fila in cuboMagico:
            print(fila)
        print("\n")
        yMax.append(max(funcion_Aptitud(individuo) for individuo in poblacion))
        yPro.append(sum(funcion_Aptitud(individuo) for individuo in poblacion)/len(poblacion))
        yMin.append(min(funcion_Aptitud(individuo) for individuo in poblacion))
        elapsed_time = time.time() - start_time
        y.append(elapsed_time)
        if calificacion == 0:
            break
    fig, ax = plt.subplots()
    ax.plot(x , yMax, color='red')
    ax.plot(x , yPro, color='blue')
    ax.plot(x , yMin, color='green')
    ax.set_title('Gráfica de aptitud')
    plt.savefig(pathGuardarGrafico);

    fig, ax = plt.subplots()
    ax.plot(x , y, color='red')
    ax.set_title('Gráfica de tiempo')
    plt.savefig(pathGuardarGraficoTiempo);

if __name__ == "__main__":
    main()
```

## 10.- Bibliografía


- Goldbarg, M. C., & de Carvalho, A. C. P. L. F. (2012). A survey of genetic algorithms for combinatorial optimization. Brazilian Journal of Computer Science, 18(1), 41-72.
- Mitchell, M. (1998). An Introduction to Genetic Algorithms. MIT Press.
- Sivanandam, S. N., & Deepa, S. N. (2007). Introduction to Genetic Algorithms. Springer.
- Whitley, D. (1994). A genetic algorithm tutorial. Statistics and Computing, 4(2), 65-85.

- Koza, J. R. (1992). Genetic programming: On the programming of computers by means of natural selection. MIT Press.