Universidad Tecnológica Nacional
Facultad Regional Rosario

Especialidad: Ing. en Sistemas de Información.

Asignatura: Algoritmos Genéticos

Comisión: 3EK03

Ciclo lectivo: 2025

Fecha: 12/06/2025

TRABAJO PRÁCTICO N°1:

Maximización de una función objetivo

Alumno	Correo	Legajo
Mondino, Juan Cruz	juancm.2000@hotmail.com	51922
Giampietro, Gustavo	gustgiam2001@gmail.com	50671
Mateo, Alexis	alexisjoelmateo@gmail.com	51191

Contenido

Enunciado

Hacer un programa que utilice un Algoritmo Genético Canónico para buscar un máximo de la función:

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

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

$$coef = 2^{30} - 1$$

Teniendo en cuenta los siguientes datos:

- Probabilidad de Crossover = 0,75.
- Probabilidad de Mutación = 0,05.
- Población Inicial: 10 individuos.
- Ciclos del programa: 20.
- Métodos de selección: Ruleta, Torneo y Elitismo.
- Método de Crossover: 1 Punto.
- Método de Mutación: invertida.
- Cantidad de individuos seleccionados por elitismo: 2.

El programa debe mostrar los siguientes datos para los diferentes métodos de selección:

1. El cromosoma correspondiente al valor máximo, el valor máximo, mínimo y promedio obtenido de cada población.
2. La impresión de las tablas de mínimos, promedios y máximos para 20, 100 y 200 corridas (para elitismo solamente los valores de 100 iteraciones del algoritmo).
3. Las gráficas de los valores Máximos, Mínimos y Promedios de la función objetivo por cada generación luego de correr el algoritmo genético 20, 100 y 200 iteraciones (para elitismo solamente las gráficas de 100 iteraciones del algoritmo).
4. Realizar comparaciones de las salidas corriendo el mismo programa en distintos ciclos de corridas y además realizar todos los cambios que considere oportunos en los parámetros de entrada de manera de enriquecer sus conclusiones.

Metodología de desarrollo

El programa consiste en ejecutarlo con la siguiente línea:

Python TP1_AG.py -c N -s M -e B

Donde:

- N es un número entero que representa la cantidad de ciclos, o generaciones, para las que uno quiere que se ejecute el programa.
- M es uno de entre dos valores, la letra “r” o la “t”. De esta forma se representa si se quiere efectuar el método de ruleta o el método de torneo respectivamente.
- Por último, B puede ser un valor entre 1 o 0, siendo 1 el que indique que se usará elitismo y siendo 0 para el caso contrario.

Con esta información, inicializa las variables del algoritmo, como la probabilidad de crossover, la probabilidad de mutación, la cantidad de individuos, la cantidad de genes por individuo, el coeficiente de normalización (*coef*) y, en caso de usarse torneo, cuántos competidores participan por ronda.

Luego se genera la población inicial: una lista de cromosomas binarios aleatorios. Cada cromosoma es una lista de 0s y 1s. A esta población se le calcula el fitness de cada cromosoma evaluando su valor decimal con la función objetivo enunciada anteriormente, y finalmente calcula su fitness relativo dentro de la población, es decir, su proporción respecto de la suma total de objetivos. Paralelamente, se calcula el máximo, mínimo, promedio y mejor individuo.

A partir de aquí, se inicia el ciclo evolutivo. El programa decide si usará elitismo o no. En caso afirmativo, se seleccionan los mejores individuos según fitness y se los aparta para preservarlos (los llamaremos elitistas). El resto de la nueva generación se obtiene por uno de los dos métodos de selección: ruleta, que asigna probabilidad proporcional al fitness, o torneo, donde se eligen aleatoriamente grupos de competidores y se selecciona el mejor de cada grupo.

Una vez seleccionados los individuos para la siguiente generación (sin contar a los elitistas), el algoritmo procede a evaluar la probabilidad de crossover. Si se termina dando, toma pares de individuos y aplica el operador de cruce en un punto, generando dos hijos. Luego, con otra probabilidad, analiza si debe aplicar una mutación a cada uno de los hijos, alterando aleatoriamente un gen de su cromosoma. Cuando se termina este proceso, si hay elitismo, los mejores individuos originales se reincorporan a la población sin modificaciones.

Con la nueva población formada, se vuelve a calcular el fitness y las estadísticas. Los valores máximos, mínimos, promedios y el mejor cromosoma de esa generación se almacenan para su posterior análisis. Este proceso se repite durante la cantidad de ciclos especificada por el usuario.

Una vez finalizados todos los ciclos, se generan los gráficos representando la evolución del valor máximo, mínimo y promedio de fitness a lo largo de las generaciones. Por último, se crea un archivo Excel, que contiene los datos obtenidos ciclo a ciclo, incluyendo el mejor cromosoma en formato binario y decimal.

El objetivo del programa es simular un algoritmo genético que permita analizar y comparar el comportamiento evolutivo de distintas estrategias de selección (ruleta y torneo), con y sin elitismo, observando cómo afectan a la convergencia de la población hacia la solución óptima

(el mayor número posible dentro del dominio [o; 2^{30}-1] que sea un máximo de la función objetivo), a través de la observación de las estadísticas en los gráficos.

Forma de trabajo abordada en equipo

Para realizar las corridas de la simulación nuestro grupo eligió programar en Python el algoritmo, para ello fue utilizado el editor de código fuente Visual Studio Code.

Como cada uno de nosotros trabajó desde su propia computadora fueron utilizadas herramientas como Google Drive, en donde fueron creadas carpetas y documentos compartidos para que cada uno pueda editar libremente, y GitHub, para poder codear sin necesidad de ir pasando el documento entre nosotros, además de aprovechar la seguridad que aporta este sistema de control de versiones online.

Herramientas de programación

Como ya fue mencionado se utilizó Python para programar los algoritmos. Pero dentro de este lenguaje fueron utilizadas diferentes librerías con el fin de implementar las siguientes funciones:

- OS: Se utiliza para realizar operaciones sobre el sistema de archivos, como verificar si existe un archivo con un nombre determinado y eliminarlo en caso de ser necesario. También permite limpiar la pantalla del terminal, dependiendo del sistema operativo (Windows, Linux, etc.).
- RANDOM: Su utilización surge por la necesidad de tener un generador de números aleatorios confiable. Además, uno puede limitar el dominio de posibles números aleatorios o especificar si es entero o real.
- SYS: Se emplea para manejar argumentos desde la línea de comandos, lo que permite ejecutar el programa con distintos parámetros (por ejemplo, cantidad de ciclos, tipo de selección, uso o no de elitismo) sin modificar el código fuente.
- MATPLOTLIB.PYPLOT: Esta librería permite generar gráficos durante o después de la ejecución del programa. Se usa para visualizar la evolución de las métricas del algoritmo (como el fitness máximo, mínimo y promedio), lo cual facilita el análisis y comprensión de los resultados.
- PANDAS: Es una herramienta poderosa para la manipulación de datos en forma de tablas. Se utiliza para construir DataFrames y exportarlos a archivos Excel (.xlsx), lo que permite guardar los resultados del algoritmo en un formato claro y accesible.
- OPENPYXL: Es el motor que PANDAS usa por defecto para escribir archivos Excel. Es necesario tenerla instalada para que la función to_excel() de PANDAS funcione correctamente con archivos .xlsx.

Código

En este apartado se explicará parte por parte el código Python utilizado para correr las simulaciones:

En la parte superior del código se especifican las librerías que se utilizarán y deben ser instaladas para correr el programa.

```
import random
import os
import sys
import matplotlib.pyplot as plt
import pandas as pd
# Utiliza openpyxl tambien
```

Luego, se definieron dos funciones: la primera es utilizada para obtener un número aleatorio entre 0 y 1 de una manera más sencilla. La segunda, se la llama cuando se requiere obtener un cromosoma completo, uno solamente debe pasarle la cantidad de genes que debe tener dicho cromosoma y la función creará el cromosoma con la longitud especificada completo por ceros y unos aleatoriamente.

```
def aleatorio():
    return random.randint(0, 1)

def completoCromosoma(cantidad):
    cromosoma = [aleatorio() for _ in range(cantidad)]
    return cromosoma
```

Aquí se busca obtener una población de una cantidad de individuos o cromosomas especificada, ésta es la función que llama a la explicada anteriormente para que complete los genes de cada cromosoma.

```
def generarPoblacion(cantidadCromosomas, cantidadGenes):
    poblacion = []
    for i in range(cantidadCromosomas):
        cromosoma = completoCromosoma(cantidadGenes)
        poblacion.append(cromosoma)
    return poblacion
```

binarioADecimal es una función que utilizamos para convertir a un cromosoma, que se puede tomar como un número binario, a su correspondiente número decimal.

funcionObjetivo se llama para evaluar qué tan buena es una solución candidata. Toma un valor (o conjunto de valores) como parámetro y devuelve una medida numérica que representa su calidad o desempeño según el problema que se busca resolver.

```
def binarioADecimal(cromosoma):
    decimal = 0
    exponente=0
    for i in range(len(cromosoma)-1,-1,-1):
        if cromosoma[i] == 1:
            decimal = decimal + pow(2,exponente)
        exponente += 1
    return decimal

def funcionObjetivo(x):
    return (x / coef) ** 2
```

En la función crossover, se reciben dos parámetros que son los padres por combinar para obtener dos nuevos individuos de la población. Entonces, se elige un punto aleatorio dentro de cada individuo para “cortarlo” y unirlo a la otra parte del segundo individuo, obteniendo así los dos nuevos cromosomas de la nueva generación.

```
def crossover1Punto(padre, madre):
    puntoCorte = random.randint(1,len(padre)-1)
    h1 = padre[:puntoCorte] + madre[puntoCorte:]
    h2 = madre[:puntoCorte] + padre[puntoCorte:]
    return h1, h2
```

Esta es la función encargada de devolver las métricas de rendimiento de cada generación, sirve para monitorear el progreso.

```
def calculadorEstadisticos(poblacion):
    objetivos = [funcionObjetivo(binarioADecimal(ind)) for ind in poblacion]
    max_objetivos = max(objetivos)
    min_objetivos = min(objetivos)
    mejor_cromosoma = poblacion[objetivos.index(max_objetivos)]
    avg_objetivos = round((sum(objetivos)/len(objetivos)),4)
    return [max_objetivos,min_objetivos, avg_objetivos, mejor_cromosoma]
```

La medida devuelta por la funcionObjetivo es luego utilizada acá, donde se calcula el fitness relativo de cada individuo dividiendo su valor objetivo por la suma total de todos los objetivos. Esto normaliza los valores, de forma que el fitness de cada individuo representa su proporción de aptitud respecto al total, lo cual es clave para el mecanismo de selección.

```
def calculadorFitness(poblacion):
    fitness = []
    objetivos = []
    sumatoria = 0
    for individuo in poblacion:
        decimal = binarioADecimal(individuo)
        obj = funcionObjetivo(decimal)
        objetivos.append(obj)
        sumatoria += obj
    if sumatoria == 0:
        print ('La suma de los valores es igual a cero.', poblacion)
        exit()
    for i in range (len(poblacion)):
        peso = objetivos[i]/sumatoria
        fitness.append(round(peso,5))
    return fitness
```

La función operadorMutacion se utiliza para aplicar variaciones aleatorias a los individuos de la población. Su objetivo es mantener la evolución genética como es en la realidad (al menos lo más cercano posible) modificando ligeramente los cromosomas dependiendo de una probabilidad establecida.

```
def operadorMutacion(individuo):
    gen_a_mutar = random.randint(0,len(individuo)-1)
    individuo[gen_a_mutar] = 1 - individuo[gen_a_mutar]
    return individuo
```

Implementa el método de selección por ruleta, donde cada individuo tiene una probabilidad de ser elegido proporcional a su valor de fitness. Se usa para formar la nueva generación favoreciendo a los más aptos, pero manteniendo algo de aleatoriedad.
Con los dos for es creado el arregloRuleta con cantidades repetidas proporcionales al fitness de cada individuo, se multiplica por 100000 para volver todos los números reales que puede haber a enteros y diferenciar mejor entre los individuos con menor fitness, y finalmente se realiza una selección aleatoria sobre el arreglo ruleta.

```
# Ruleta
def seleccionRuleta(poblacion, fitnessValores, cantidad):
    arregloRuleta = []
    seleccionados = []
    for i in range(len(fitnessValores)):
        fitnessValores[i] = fitnessValores[i]*100000
        fitnessValores[i] = int(fitnessValores[i])
        for j in range(fitnessValores[i]):
            arregloRuleta.append(poblacion[i])
    for i in range (cantidad):
            nro = random.randint(0,99999)
            seleccionados.append(arregloRuleta[nro])
    return seleccionados
```

Realiza la selección por torneo, donde se eligen varios competidores al azar y gana el que tiene mayor fitness. Se repite hasta completar la cantidad de individuos requeridos. Es un método de selección que introduce competencia directa entre cromosomas.

```
# Torneo
def seleccionTorneo(poblacion, fitnessValores, cantidadIndividuos, cantidadCompetidores):
    ganadores = []
    for j in range(cantidadIndividuos):
        competidores = []
        fitness_competidores = []
        for i in range(cantidadCompetidores):
            c=random.randint(0,len(poblacion)-1)
            competidores.append(poblacion[c])
            fitness_competidores.append(fitnessValores[c])
        ganador = competidores[fitness_competidores.index(max(fitness_competidores))]
        ganadores.append(ganador)
    return ganadores
```

```
# Elitismo
def ciclos_con_elitismo(ciclos, prob_crossover, prob_mutacion,cantidadIndividuos, cantidadGenes, metodo_seleccion,
cantidadElitismo, cantidadCompetidores=None):
    maximos=[]
    minimos=[]
    promedios=[]
    mejores=[]
    pob = generarPoblacion(cantidadIndividuos,cantidadGenes)
    fit = calculadorFitness(pob)
    rta = calculadorEstadisticos(pob)
    for j in range (ciclos):
        #De la poblacion me quedo con los de elite.
        elitistas = []
        fit_ordenados = sorted(fit, reverse=True)
        for i in range(cantidadElitismo):
            indice = fit.index(fit_ordenados[i])
            elitistas.append(pob[indice])
        if metodo_seleccion == 'r':
            pob = seleccionRuleta(pob,fit, cantidadIndividuos-cantidadElitismo)
        else:
            pob = seleccionTorneo(pob, fit, cantidadIndividuos-cantidadElitismo, cantidadCompetidores)
        for i in range (0,len(pob),2):
            padre = pob[i]
            madre = pob[i+1]
            if random.random() < prob_crossover :
                hijo1, hijo2 = crossover1Punto(padre,madre)
                pob[i], pob[i+1] = hijo1, hijo2
            if random.random() < prob_mutacion:
                pob[i] = operadorMutacion(pob[i])
            if random.random() < prob_mutacion:
                pob[i+1] = operadorMutacion(pob[i+1])
        pob = pob + elitistas
        fit = calculadorFitness(pob)
        rta = calculadorEstadisticos(pob)
        #GUARDAR VALORES NECESARIOS PARA LA GRAFICA
        maximos.append(rta[0])
        minimos.append(rta[1])
        promedios.append(rta[2])
        mejores.append(rta[3])
    return maximos, minimos, promedios, mejores
```

Ejecuta el ciclo evolutivo del algoritmo genético con elitismo, es decir, conservando a los mejores individuos de cada generación. En cada iteración se hace selección, cruzamiento, mutación y se agregan los elitistas a la nueva población. Además, al final se almacenan los valores de las métricas medidas de cada iteración.

```
# Sin elitismo
def ciclos_sin_elitismo(ciclos, prob_crossover, prob_mutacion, cantidadIndividuos, cant_genes, metodo_seleccion,
cantidadCompetidores=None):
    maximos=[]
    minimos=[]
    promedios=[]
    mejores=[]
    pob = generarPoblacion(cantidadIndividuos,cant_genes)
    fit = calculadorFitness(pob)
    rta = calculadorEstadisticos(pob)
    maximos.append(rta[0])
    minimos.append(rta[1])
    promedios.append(rta[2])
    mejores.append(rta[3])
    for j in range (ciclos):
        if metodo_seleccion == 'r':
            pob = seleccionRuleta(pob,fit, cantidadIndividuos)
        else:
            pob = seleccionTorneo(pob, fit, cantidadIndividuos, cantidadCompetidores)
        for i in range (0,len(pob),2):
            padre = pob[i]
            madre = pob[i+1]
            if random.random() < prob_crossover :
                hijo1, hijo2 = crossover1Punto(padre,madre)
                pob[i], pob[i+1] = hijo1, hijo2
            if random.random() < prob_mutacion:
                pob[i] = operadorMutacion(pob[i])
            if random.random() < prob_mutacion:
                pob[i+1] = operadorMutacion(pob[i+1])
        fit = calculadorFitness(pob)
        rta = calculadorEstadisticos(pob)
        #GUARDAR VALORES NECESARIOS PARA LA GRAFICA
        maximos.append(rta[0])
        minimos.append(rta[1])
        promedios.append(rta[2])
        mejores.append(rta[3])
    return maximos, minimos, promedios, mejores
```

Ejecuta el ciclo evolutivo sin conservar explícitamente los mejores individuos. En cada generación se hace selección, cruzamiento y mutación.

Genera un archivo Excel con los datos de cada generación: máximos, mínimos, promedio de aptitud y el mejor cromosoma. Se utiliza para registrar los resultados y poder analizarlos o compararlos luego.

```
def crear_tabla(maximos, minimos, promedios, mejores, metodo_seleccion, elitismo_Bool):
    cadenas = [''.join(str(num) for num in cromosoma) for cromosoma in mejores]
    decimales = [str(binarioADecimal(cromosoma)) for cromosoma in mejores]
    nombreMetodo = ''
    nombreElitismo = ''
    nombreCantidadCiclos = str(len(maximos)-1)
    if metodo_seleccion == 'r':
        nombreMetodo = '_Ruleta'
    else:
        nombreMetodo = '_Torneo'
    if elitismo_Bool == 1:
        nombreElitismo = '_Elitismo'
    df_nuevo = pd.DataFrame({
        'Corrida': range(len(maximos)),
        'Max': maximos,
        'Min': minimos,
        'AVG': promedios,
        'Decimal': decimales,
        'Mejor Cromosoma': cadenas,
    })
    archivo_excel = 'VALORES_' + nombreCantidadCiclos + 'Ciclos' + nombreMetodo + nombreElitismo + '.xlsx'
    if os.path.exists(archivo_excel):
        os.remove(archivo_excel)
        df_nuevo.to_excel(archivo_excel, index=False)
    else:
        df_nuevo.to_excel(archivo_excel, index=False)
```

Crea tres gráficos (máximos, mínimos y promedios de aptitud por ciclo) para visualizar la evolución del rendimiento del algoritmo a lo largo de las generaciones.

```
# GRAFICOS
def generar_grafico(maximos, minimos, promedios, mejores, titulo, ciclo):
    x = list(range(len(maximos)))
    fig, axs = plt.subplots(1, 3, figsize=(15, 4))
    axs[0].plot(x, maximos, marker='o', linestyle='-', color='b', linewidth=0.7, markersize=2)
    axs[0].set_title('Máximos')
    axs[0].set_xlabel('CORRIDA')
    axs[0].set_ylabel('APTITUD')
    axs[0].set_ylim(0, 1.2)
    axs[0].set_xlim(0, ciclo + 2)
    axs[0].grid(True)
    axs[1].plot(x, minimos, marker='o', linestyle='-', color='g', linewidth=0.7, markersize=2)
    axs[1].set_title('Mínimos')
    axs[1].set_xlabel('CORRIDA')
    axs[1].set_ylim(0, 1.2)
    axs[1].set_xlim(0, ciclo + 2)
    axs[1].grid(True)
    axs[2].plot(x, promedios, marker='o', linestyle='-', color='r', linewidth=0.7, markersize=2)
    axs[2].set_title('Promedios')
    axs[2].set_xlabel('CORRIDA')
    axs[2].set_ylim(0, 1.2)
    axs[2].set_xlim(0, ciclo + 2)
    axs[2].grid(True)
    fig.suptitle(titulo, fontsize=14)
    plt.tight_layout(rect=[0, 0, 1, 0.95])
    plt.show()
```

```
# PROGRAMA PRINCIPAL
probCrossover = 0.75
probMutacion = 0.05
cantidadIndividuos = 10
cantidadElitismo = 2
cantidadCompetidores = int(cantidadIndividuos * 0.4)
cantidadGenes = 30
coef = (2 ** cantidadGenes) – 1
maximosPorCiclo = []
minimosPorCiclo = []
promediosPorCiclo = []

if len(sys.argv) != 7 or sys.argv[1] != "-c" or sys.argv[3] != "-s" or sys.argv[5] != "-e":
    print("Uso: python TP1_AG.py -c <ciclos> -s <seleccion: r-ruleta t-torneo> -e <elitismo: 1-si 0-no>")
    sys.exit(1)
if int(sys.argv[2]) < 0 or (int(sys.argv[6]) != 0 and int(sys.argv[6]) != 1) or (sys.argv[4] != "r" and sys.argv[4] != "t"):
    print("Error: python TP1_AG.py -c <ciclos> -s <seleccion: r-ruleta t-torneo> -e <elitismo: 1-si 0-no>")
    sys.exit(1)

ciclosPrograma = int(sys.argv[2])

if int(sys.argv[6]) == 1:
    maximosPorCiclo, minimosPorCiclo, promediosPorCiclo, mejores = ciclos_con_elitismo(ciclosPrograma,probCrossover, probMutacion, cantidadIndividuos,
cantidadGenes, sys.argv[4], cantidadElitismo=cantidadElitismo, cantidadCompetidores = cantidadCompetidores)
    if sys.argv[4] == 'r':
        titulo = 'Seleccion RULETA ELITISTA - de '+ str(ciclosPrograma) + ' ciclos'
    else:
        titulo = 'Seleccion TORNEO ELITISTA - de '+ str(ciclosPrograma) + ' ciclos'
    generar_grafico(maximosPorCiclo, minimosPorCiclo, promediosPorCiclo, mejores, titulo, ciclosPrograma)
    crear_tabla(maximosPorCiclo, minimosPorCiclo, promediosPorCiclo, mejores, sys.argv[4], int(sys.argv[6]))
else:
    maximosPorCiclo, minimosPorCiclo, promediosPorCiclo, mejores = ciclos_sin_elitismo(ciclosPrograma,probCrossover, probMutacion, cantidadIndividuos,
cantidadGenes, sys.argv[4], cantidadCompetidores = cantidadCompetidores)
    if sys.argv[4] == 'r':
        titulo = 'Seleccion RULETA - de '+ str(ciclosPrograma) + ' ciclos'
    else:
        titulo = 'Seleccion TORNEO - de '+ str(ciclosPrograma) + ' ciclos'
    generar_grafico(maximosPorCiclo, minimosPorCiclo, promediosPorCiclo, mejores, titulo, ciclosPrograma)
    crear_tabla(maximosPorCiclo, minimosPorCiclo, promediosPorCiclo, mejores, sys.argv[4], int(sys.argv[6]))
```

Por último, en este bloque define los parámetros generales del algoritmo genético, como la probabilidad de cruzamiento (probCrossover), de mutación (probMutacion), la cantidad de individuos por generación y la cantidad de genes por cromosoma. También toma los argumentos de entrada desde la consola, que indican la cantidad de ciclos a ejecutar, el tipo de selección (r para ruleta o t para torneo) y si se aplica elitismo (1 sí, 0 no).

Luego, según los argumentos recibidos, elige si ejecutar el ciclo con o sin elitismo, usando las funciones ciclos_con_elitismo o ciclos_sin_elitismo. Al finalizar, genera un gráfico con la evolución de los valores máximos, mínimos y promedios de aptitud, y exporta una tabla Excel con los resultados por ciclo para su análisis.

En resumen, este bloque es el punto de entrada del programa, controla el flujo principal del algoritmo genético y gestiona la visualización y guardado de resultados.

Salidas por pantalla

Los datos obtenidos en cada generación fueron almacenados en archivos Excel. Como algunas de las corridas se realizaron para gran cantidad de generaciones, decidimos subir estos archivos a una carpeta de drive y compartir el link. Esto debido a que examinar directamente esa cantidad de valores numéricos podría ser tedioso y no le aportaría tanto a la conclusión, así como lo hacen los gráficos de la siguiente sección.

Link:

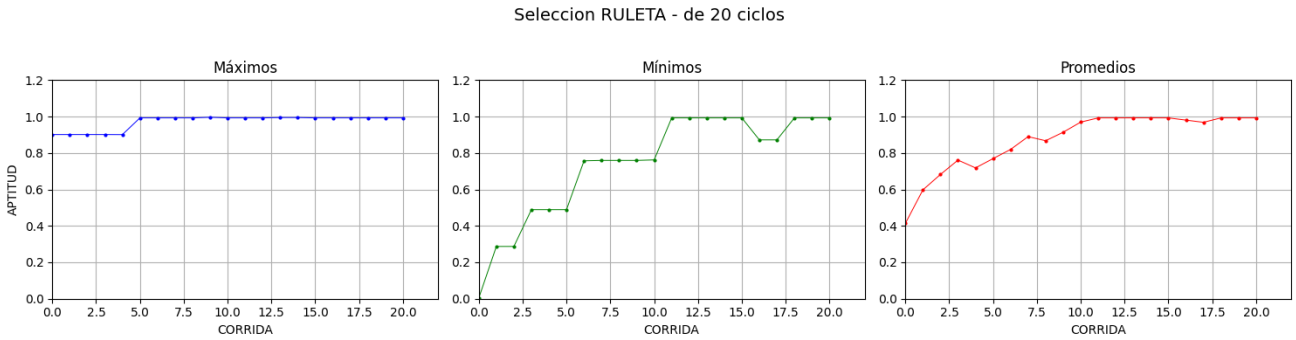
https://drive.google.com/drive/folders/1Th12fWT9TuVWHyZnt1ePVbe9SKPeT2lQ?usp=sharing

Gráficas

Sin elitismo:

Ruleta:

Para 20 corridas:

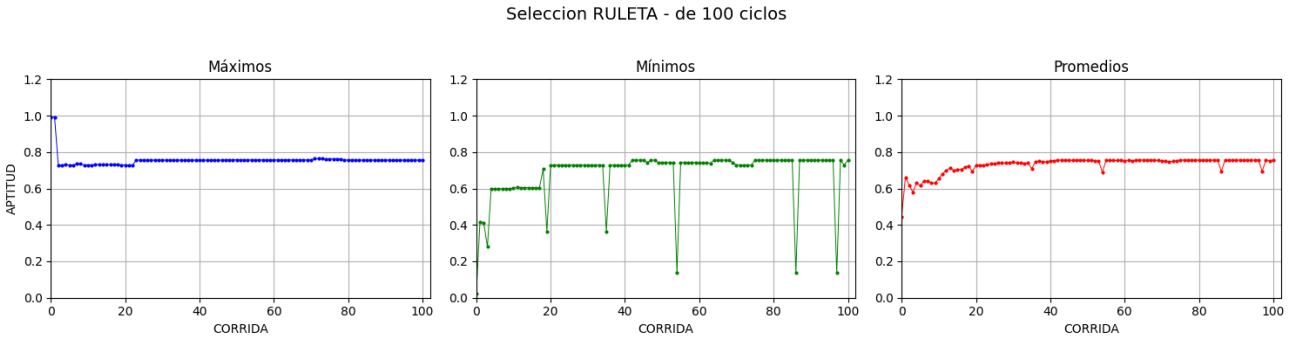


Como se puede observar en la imagen, es difícil que varíe la gráfica de los máximos a no ser que sufran una mutación, cosa que no sucede en este caso, ya que al tener más probabilidades de salir en la ruleta, se mantienen aunque pasen por el crossover. Esto se da debido a que la parte de mayor peso del número binario es la que indica que tan grande el número es, y la parte de menor peso no modifica tanto el número.

En la gráfica de los mínimos se puede observar que es mayoritariamente creciente, los números más altos tienen mayor probabilidad de ser los que pasen por el crossover y generen la nueva población, es por ello que los valores mínimos son cada vez más altos a no ser que salga sorteado un valor con probabilidad baja o que algún cromosoma sufra una mutación en algún gen que represente un bit de peso.

Por último, en la gráfica de promedios se observa una tendencia a seguir la forma de la gráfica de mínimos, ya que la de máximos no se mueve tanto y que como los mínimos son cada vez más grandes implica que los valores de la población en general son más grandes.

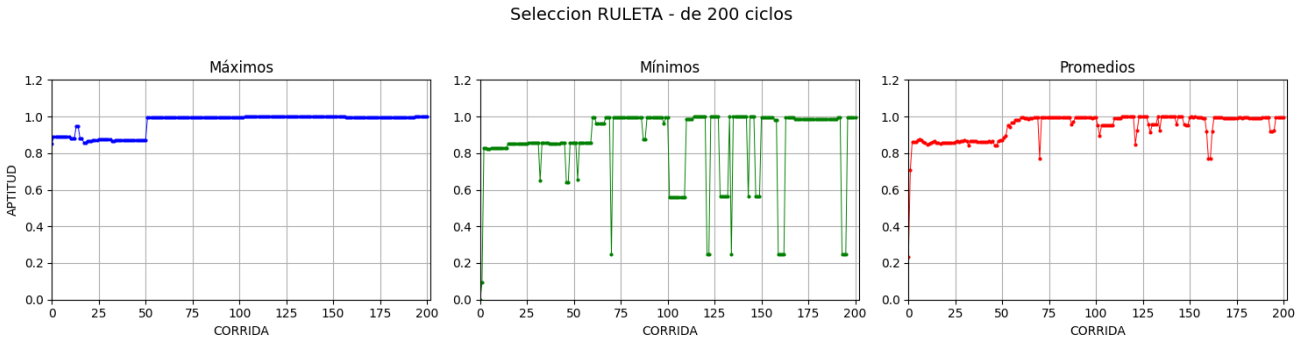
Para 100 corridas:



A diferencia de los gráficos anteriores, en este caso se puede observar que la gráfica de los máximos sufre una caída abrupta cerca de las primeras iteraciones (alrededor de la tercera), esto puede ser dado por una mutación en algún gen que represente algún bit de peso (esta caso no me convence del todo porque la caída es muy abrupta, tiene que haber mutación en varios cromosomas o solamente había un solo cromosoma con un valor tan alto a esa altura y justo sufre la mutación) o por no haber salidos sorteados los cromosomas con mayor probabilidad en la ruleta. Luego, no se observan mayores cambios en los máximos.

Los mínimos y los promedios tienden a crecer, estabilizándose pasando la corrida número veinte, pero luego se observan picos hacia abajo en alguna que otra iteración aisladas (comparadas con las aledañas), lo que me lleva a pensar una vez más en mutaciones que generaron valores mínimos que no superan el siguiente sorteo, es por esto que se observan picos y no mesetas.

Para 200 corridas:



Al igual que en la gráfica anterior, se pueden observar picos en las curvas, pero al tratarse esta vez de una mayor cantidad de generaciones, hay también más oportunidades de que estas variaciones ocurran a lo largo de la simulación. A diferencia del caso anterior, aquí se observan mutaciones que no solo introducen variabilidad negativa, sino que también generan mejoras en la población, como se aprecia en al menos dos ocasiones en el gráfico de máximos. La mutación ocurrida cerca de la decimoquinta generación produjo un aumento momentáneo en el máximo, aunque su efecto desapareció en apenas dos iteraciones. En cambio, la mutación surgida hacia la quincuagésima tirada sí logró mantenerse, evidenciando que el algoritmo es capaz de encontrar una solución óptima, pero sin garantía de preservarla. La presencia de oscilaciones y falta de estabilidad inicial refuerzan la idea de

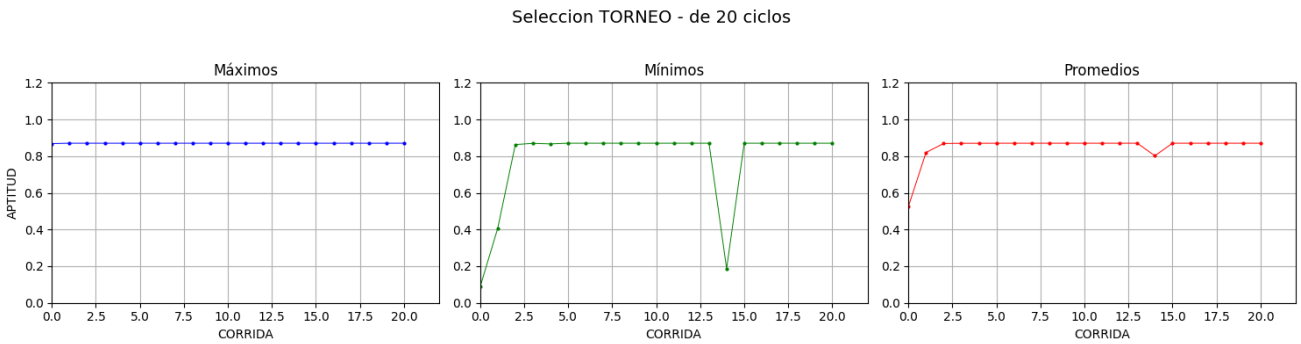
que, sin elitismo, las mejores soluciones pueden perderse fácilmente por efectos del azar (mutaciones o decisiones de selección desfavorables).

La curva de mínimos es donde se manifiesta con mayor claridad esta inestabilidad. Aunque en general los valores tienden a mejorar con el tiempo, aparecen múltiples descensos bruscos y profundos en distintas etapas del proceso, incluso hacia el final. Esto sugiere que, sin un mecanismo de conservación como el elitismo, los individuos de baja aptitud pueden seguir ingresando a la población, ya sea por mutación o recombinación aleatoria, y no hay forma de impedir su permanencia temporal. A diferencia de los máximos, esta curva no muestra una estabilización clara.

En cuanto a la aptitud promedio, se observa una mejora sostenida a lo largo de las generaciones, con un salto notable que coincide con el ascenso en la curva de máximos. A partir de ese punto, el promedio se mantiene en valores elevados, aunque con caídas esporádicas que reflejan la presencia ocasional de individuos poco aptos. Estas oscilaciones coinciden en general con los descensos en la curva de mínimos, y muestran cómo incluso una pequeña porción de la población puede seguir afectando el rendimiento general cuando no se protege explícitamente la calidad de las soluciones ya obtenidas.

Torneo:

Para 20 corridas:

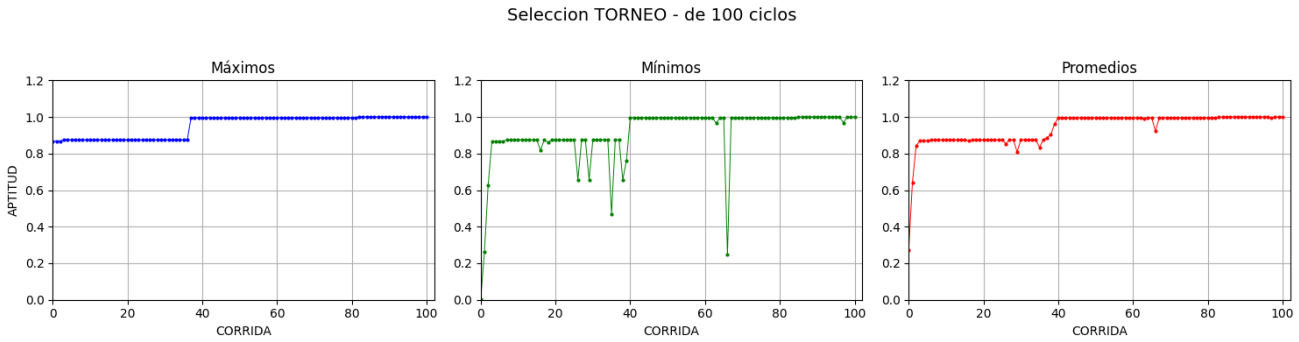


La aptitud máxima se mantiene constante y baja (alrededor de 0.52). Esto indica que el algoritmo no logra mejorar la mejor solución durante las veinte generaciones.

La aptitud mínima comienza cerca del cero y sube rápidamente en las primeras dos corridas, estabilizándose luego cerca de 0.85. Esto sugiere una mejora inicial del conjunto poblacional, pero luego se estanca (obviando una posible mutación en la decimocuarta generación).

El promedio de aptitud aumenta rápidamente al principio y luego se estabiliza cerca de 0.48. Esto es coherente con el comportamiento de los mínimos y refleja que la población general mejora al principio, pero no logra avanzar mucho más allá.

Para 100 corridas:

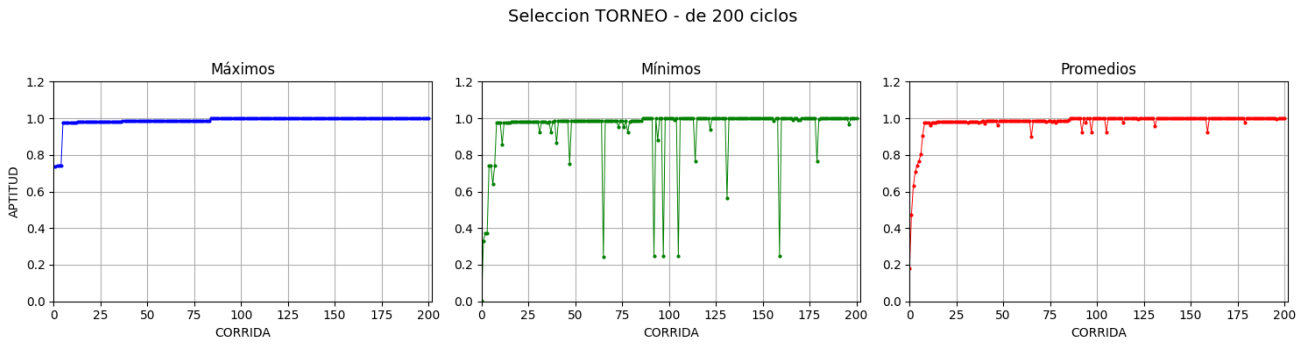


La aptitud máxima comienza cerca de 0.88 y da un salto claro hacia 1.0 alrededor de la trigésima quinta generación, manteniéndose en ese valor. Esto indica que el algoritmo eventualmente encuentra una solución óptima o muy cercana. La mejora no es continua, sino que ocurre en un salto puntual.

La curva de los mínimos es más inestable, aunque oscila cerca de 0.9, hay varias caídas abruptas (hasta 0.2 cerca de la sexagésima corrida).

La media muestra una tendencia general ascendente, aunque con oscilaciones. Al igual que los máximos, el promedio mejora drásticamente alrededor de la trigésima quinta corrida, coincidiendo con la aparición del individuo más apto. Después de eso, se mantiene alto (cerca de 1.0), aunque con pequeños descensos temporales debido a los bajos mínimos.

Para 200 corridas:



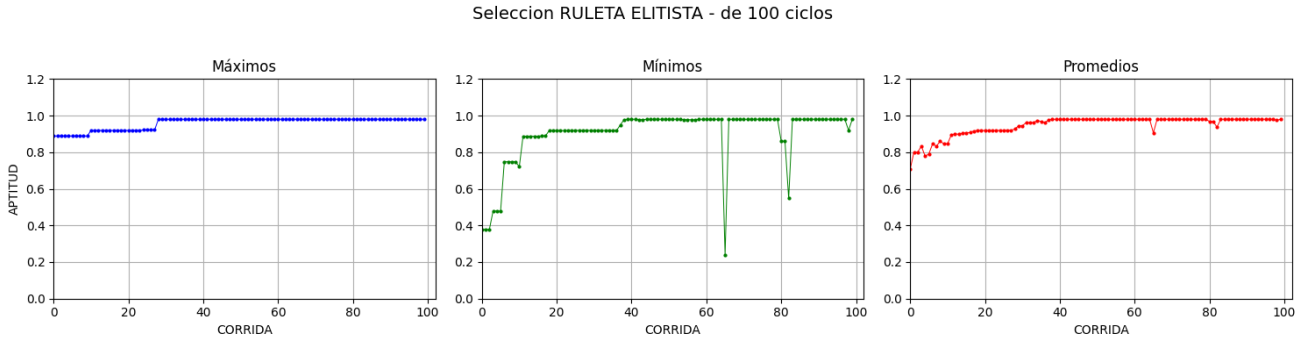
Una vez más, luego del gran salto para obtener un máximo alrededor de la octava generación, es muy difícil que varíe éste ya que queda poco margen de mejora.

Como se observó anteriormente, a mayor número de generaciones es más probable que sucedan mutaciones (aún más con el número de probabilidad con el que se trabajó), viéndose reflejadas como picos en las gráficas. Aún así no son de mayor inconveniente porque no sobreviven más de una generación.

Con elitismo:

Para ambos casos se decidió realizar una corrida de cien generaciones.

Ruleta:

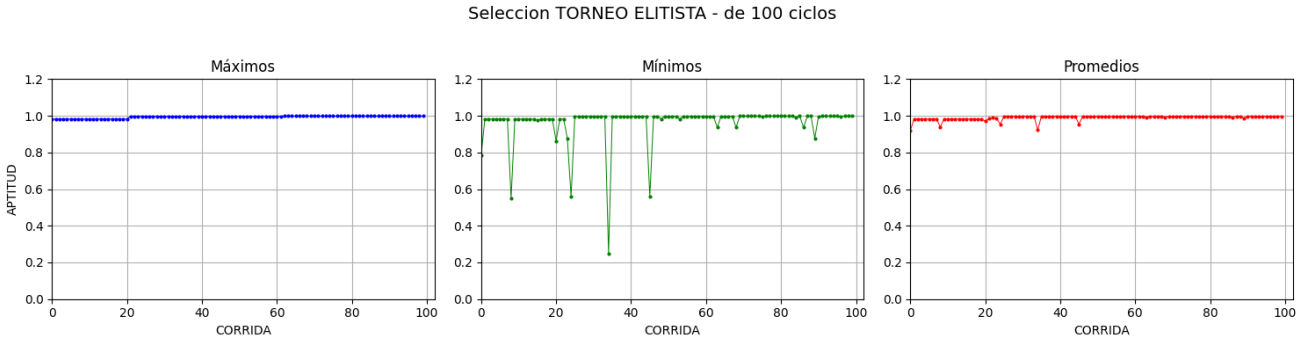


En esta simulación se observa un comportamiento notablemente más estable y eficiente en comparación con las ejecuciones sin elitismo. La gráfica de máximos muestra una evolución más clara: comienza en un valor relativamente bajo y luego asciende progresivamente hasta alcanzar el valor óptimo (fitness ≈ 1.0), que se mantiene sin sobresaltos durante el resto de las generaciones. Esto indica que, una vez alcanzada una solución óptima, el elitismo asegura su preservación, evitando que el mejor individuo se pierda por mutación o azar.

La curva de mínimos muestra una tendencia general ascendente, aunque presenta algunos pequeños descensos puntuales. Estos picos pueden atribuirse a mutaciones que generan individuos de baja aptitud que, al no ser seleccionados ni protegidos por elitismo, no perduran más de una generación. Aun así, el elitismo protege la calidad global, lo que se refleja en una curva que rápidamente vuelve a subir.

El gráfico del promedio sigue un camino similar al de los mínimos: muestra una mejora progresiva con leves oscilaciones, estabilizándose en valores altos una vez que el mejor individuo ha sido incorporado consistentemente en la población. Esto refuerza la idea de que el elitismo contribuye a mejorar la población completa, no solo a conservar los mejores.

Torneo:



Este segundo gráfico, correspondiente al método de torneo con elitismo, también muestra una evolución clara hacia la convergencia óptima. La curva de máximos comienza en un valor intermedio y asciende rápidamente, estabilizándose en 1.0 (óptimo). Al igual que en el caso anterior, una vez alcanzado el valor máximo, el elitismo evita su pérdida, consolidando una población cada vez más apta.

La curva de mínimos, si bien algo más irregular que en el caso de ruleta, también sigue una tendencia ascendente. Se observan algunos picos breves hacia abajo, señal de la aparición

esporádica de mutaciones o combinaciones que producen malos individuos. Sin embargo, al contar con elitismo, estos individuos no afectan la permanencia de los mejores, lo que evita que estos valores bajos se perpetúen.

La curva del promedio crece con una inclinación constante, lo que indica una mejora colectiva sostenida de la población. Las oscilaciones son suaves, lo que refleja la efectividad del torneo (en este caso con elitismo) para preservar buenas soluciones y explorar nuevas al mismo tiempo.

Conclusiones

A lo largo del desarrollo del trabajo práctico, se observaron diferencias marcadas en el comportamiento de los algoritmos genéticos según el método de selección utilizado y la presencia o ausencia de elitismo. La selección por ruleta, si bien es sencilla de implementar y basa sus decisiones en una lógica probabilística proporcional al fitness, mostró una fuerte dependencia de las mutaciones para producir mejoras significativas. En varias simulaciones, los valores máximos se mantuvieron estancados hasta que una mutación afortunada introdujo una solución de mayor calidad, lo que evidencia una presión selectiva débil y una convergencia más lenta hacia el óptimo.

En contraste, la selección por torneo arrojó resultados más dinámicos, ya que favorece de manera directa a los individuos con mayor fitness dentro de subconjuntos aleatorios de la población. Esto permitió una mejora más rápida de los valores máximos, aunque también introdujo una mayor variabilidad en los mínimos. La falta de elitismo en ambos métodos permitió que individuos de alta calidad se pierdan ocasionalmente.

La lentitud de convergencia del método de ruleta se explica porque incluso los peores individuos conservan una probabilidad no nula de ser seleccionados, mientras que los mejores pueden quedar fuera del proceso de cruce. Por su parte, el método de torneo también presenta vulnerabilidades, si se enfrentan dos individuos de baja calidad en una misma competencia, uno de ellos avanzará, y si los mejores compiten entre sí, solo uno continuará. Esto puede derivar en un deterioro de la calidad de la población. En este contexto, la incorporación de elitismo se reveló como un factor clave para estabilizar la evolución del algoritmo, ya que garantiza la conservación de las mejores soluciones y favorece tanto la convergencia como la mejora general de la población.

En resumen, mientras que la ruleta tiende a avanzar de forma más lenta y dependiente del azar, el torneo permite mejoras más rápidas pero con menor estabilidad. El elitismo actúa como un mecanismo de protección que mitiga las debilidades de ambos métodos, permitiendo una evolución más sólida y sostenida hacia la solución óptima.