UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Tarea 05 - Análisis de Resultados

ASIGNATURA

Cómputo Evolutivo

PROFESOR

Oscar Hernández Constantino

AYUDANTE

Malinali Gónzalez Lara

ALUMNOS

Carlos Emilio Castañon Maldonado & Dana Berenice Hérnandez Norberto

Ejercicio 1. Parametrización de Algoritmos

En esta tarea vamos a comparar diferentes algoritmos para resolver problemas de optimización continua. Implementa los cambios necesarios para poder realizar las siguientes comparaciones:

1 a) **Parámetros del Recocido simulado**
a) Comparar al menos dos esquemas de enfriamiento.

1 b) **Parámetros del Algoritmo genético**
a) Comparar los siguientes esquemas de reemplazo:

i. Generacional
ii. Generacional con elitismo
iii. Reemplazo de los peores

1 c) **Mejor parametrización**
a) Recocido simulado vs Algoritmo genético

Primeramente, aplicamos los esquemas de enfriamiento requerido, en nuestro caso decidimos utilizar los esquemas de enfriamiento de decremento lento y logarítmico, el esquema de decremento lento fue nuestro esquema inicial con los que no obtuvimos los mejores resultados. Por otro lado, decidimos utilizar el esquema logarítmico dado que posee un decremento similar al ya utilizado, creemos que es interesante analizar las diferencias entre ellos.

```
# Implementamos ambos esquemas de enfriamiento
# temp: En ambos esquemas se refiere a la temperatura inicial
# beta y alpha: factores de enfriamiento
# t: numero de iteracion

def decremento_lento(temp,beta=0.1):
    return temp/(1+beta*temp)

def logaritmico(t,temp, alpha=0.95):
    return temp*alpha**t

# Funcion auxiliar
def enfriamiento(esquema,t,temp,beta,alpha):
    evaluation = None
    if esquema == 'decremento':
        evaluation = decremento_lento(temp,beta)
    elif esquema == 'logaritmico':
        evaluation = logaritmico(t,temp,alpha)
    return evaluation
```

Ahora implementamos los esquemas de reemplazo del algoritmo genético, veamos que se nos solicitó implementar tres esquemas de reemplazo.

- **Generacional:** Reemplazamos la generación entera de la población inicial, independientemente de la aptitud de los individuos de la población
- **Generacional con elitismo:** Reemplazamos la generación entera, dependiendo de la aptitud de los individuos.
- **Reemplazo de los peores:** Reemplazamos a la generación, tomando en cuenta quienes tienen las peores aptitudes, con el fin de quedarnos con los mejores y obtener mejores soluciones.

```
# Reemplazamos toda la generación, independientemente de
# las aptitudes

def generacional(population,values,new_population,new_values):
    population[:len(new_population)] = new_population
```

```
    values[:len(new_population)]= new_values
    return population, values

# Reemplazamos toda la generación, dependiendo de las aptitudes,
# por esta razon ordenamos por las aptitudes y evitamos caer en
# optimos locales
def elitismo(population,values,new_population,new_values):
    # Con argsort podemos extraer los indices de los valores mas pequeños que tenemos
    index = np.argsort(values)[:len(new_population)]
    population[index] = new_population
    values[index]= new_values
    return population, values

# Reemplazamos a los peores, por eso, a diferencia del generacional
# con elitismo, realizamos un proceso inverso, ordenamos las aptitudes
# al reves con el fin de extraer a los peores, ya que estamos
# trabajando con problemas de minimizacion.
def peores(population,values,new_population,new_values):
    index = np.flip(np.argsort(values)[:len(new_population)])
    population[index] = new_population
    values[index]= new_values
    return population, values

# Funcion auxiliar
def reemplazo(esquema,population,values,new_population,new_values):
    remplace = None
    if esquema == 'generacional':
        remplace = generacional(population,values,new_population,new_values)
    elif esquema == 'elitismo':
        remplace = elitismo(population,values,new_population,new_values)
    elif esquema == 'peores':
        remplace = peores(population,values,new_population,new_values)
    return remplace
```

Conservamos nuestros algoritmos de recocido simulado y algoritmo genético de la misma forma que en los reportes anteriores, con algunas modificaciones referentes al ingreso del parámetro extra, ingresando el esquema de enfriamiento o reemplazo que queremos utilizar.

```
#Ejemplo de uso de el algoritmo genetico, podemos utilizar los argumentos de los
#nombres de las funciones, asi como el esquema de reemplazo que queremos utilizar
algoritmo_genetico(30,1000, 10, 0.1,'ackley','peores')
min((algoritmo_genetico(30,1000, 10, 0.1,'ackley','peores')[0]))

#Ejemplo de uso del algoritmo de recocido simulado, se puede utilizar como argumento
# el esquema de enfriamiento "logaritmico" o "decremento"
recocido_simulado(1000,0.2,'ackley', 10,'logaritmico',beta=0.01,alpha =0.95)
recocido_simulado(1000,0.2,'ackley', 10,'decremento',beta=0.01,alpha =0.95)
```

Ambos algoritmos utilizan una representación de soluciones binaria, por lo tanto, es posible realizar una comparación entre ellas. Además de que ambas funciones devuelven tanto la mejor solución como la mejor evaluación.

Ejercicio 2. Experimentación en Optimización continua

2 a) **Realizar al menos 30 repeticiones para diferentes funciones de prueba de optimización continua, en dimensión 10.**

2 b) **Fijar criterio de término para todas las ejecuciones, de manera que se tenga una comparación justa (por ejemplo con tiempo total de ejecución).**

2 c) **El programa deberá imprimir como salida la semilla del generador de números aleatorios que se utilizó para la ejecución.**

2 d) **El programa deberá permitir ingresar (como parámetro opcional) la semilla del generador de aleatorios con las que se hará la ejecución. Si no se indica nada, el programa deberá generar una semilla adecuada.**

*Se utilizarán las funciones de optimización continua utilizadas en la tarea 3.

Para poder lograr lo establecido anteriormente, vamos a declarar la siguiente función, en la que por default vamos a tener 30 repeticiones además de que nuestro criterio de termino será el total de ejecución, en caso de no ingresar una semilla la función se encarga de generar una:

```
import time
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

def ejecutar_experimentos(funciones, dimension=10, repeticiones=30, semilla=None):
    resultados = []

    # Configuracion de semilla aleatoria si no se proporciona una
    if semilla is None:
        semilla = np.random.randint(12099)
    np.random.seed(semilla)
    print("Semilla del generador de numeros aleatorios utilizada:", semilla)

    for funcion in funciones:
        resultados_funcion = []
        for _ in range(repeticiones):
            inicio = time.time()

            mejor_aptitud_genetico_gen, _ = algoritmo_genetico(30,
            1000, dimension, 0.1, funcion, 'generacional')
            tiempo_generacional = time.time() - inicio

            inicio = time.time()

            mejor_aptitud_genetico_elite, _ = algoritmo_genetico(30,
            1000, dimension, 0.1, funcion, 'elitismo')
            tiempo_elitismo = time.time() - inicio

            inicio = time.time()

            mejor_aptitud_genetico_peor, _ = algoritmo_genetico(30,
            1000, dimension, 0.1, funcion, 'peores')
            tiempo_peor = time.time() - inicio

            inicio = time.time()

            # Ejecutamos recocido simulado con esquema logaritmico
```

```
            _, mejor_aptitud_log = recocido_simulado(1000, 0.2, funcion,
            dimension, 'logaritmico', beta=0.01, alpha=0.95)
            tiempo_logaritmico = time.time() - inicio

            inicio = time.time()

            # Ejecutamos recocido simulado con esquema de decremento lento
            _, mejor_aptitud_decremento = recocido_simulado(1000,
            0.2, funcion, dimension, 'decremento', beta=0.01, alpha=0.95)
            tiempo_decremento = time.time() - inicio

            resultados_funcion.append({
                'Funcion': funcion,
                'Algoritmo': 'Algoritmo Genetico (Generacional)',
                'Mejor Aptitud': min(mejor_aptitud_genetico_gen),
                'Tiempo': tiempo_generacional
            })

            resultados_funcion.append({
                'Funcion': funcion,
                'Algoritmo': 'Algoritmo Genetico (Elitismo)',
                'Mejor Aptitud': min(mejor_aptitud_genetico_elite),
                'Tiempo': tiempo_elitismo
            })

            resultados_funcion.append({
                'Funcion': funcion,
                'Algoritmo': 'Algoritmo Genetico (Reemplazo de los peores)',
                'Mejor Aptitud': min(mejor_aptitud_genetico_peor),
                'Tiempo': tiempo_peor
            })

            resultados_funcion.append({
                'Funcion': funcion,
                'Algoritmo': 'Recocido Simulado (Logaritmico)',
                'Mejor Aptitud': mejor_aptitud_log,
                'Tiempo': tiempo_logaritmico
            })

            resultados_funcion.append({
                'Funcion': funcion,
                'Algoritmo': 'Recocido Simulado (Decremento Lento)',
                'Mejor Aptitud': mejor_aptitud_decremento,
                'Tiempo': tiempo_decremento
            })

        resultados.extend(resultados_funcion)

    return resultados
```

Al ejecutar obtenemos lo siguiente:

```
# Ejecutamos las funciones
funciones = ['sphere', 'ackley', 'griewank', 'rastrigin', 'rosenbrock']
resultados_experimentos = ejecutar_experimentos(funciones, dimension=10, repeticiones=30)
```

Se nos muestra la semilla usada (notemos como es que en la ejecución anterior pudimos establecer nosotros la semilla pero no lo hicimos y dejamos que el programa la generara por nosotros):

```
Semilla del generador de numeros aleatorios utilizada: 2180
```

Además de que si ejecutamos:

```
resultados_experimentos
```

Podremos observar la salida completa con todos los datos:

```
[{'Funcion': 'sphere',
  'Algoritmo': 'Algoritmo Genetico (Generacional)',
  'Mejor Aptitud': 0,
  'Tiempo': 1.5239992141723633},
 {'Funcion': 'sphere',
  'Algoritmo': 'Algoritmo Genetico (Elitismo)',
  'Mejor Aptitud': 1,
  'Tiempo': 1.4329030513763428},
 {'Funcion': 'sphere',
  'Algoritmo': 'Algoritmo Genetico (Reemplazo de los peores)',
  'Mejor Aptitud': 1,
  'Tiempo': 1.4321107864379883},
 {'Funcion': 'sphere',
  'Algoritmo': 'Recocido Simulado (Logaritmico)',
  'Mejor Aptitud': 5,
  'Tiempo': 0.0},
 {'Funcion': 'sphere',
  'Algoritmo': 'Recocido Simulado (Decremento Lento)',
  'Mejor Aptitud': 0,
  'Tiempo': 0.007978677749633789},
  .
  .
  .
   {'Funcion': 'rosenbrock',
  'Algoritmo': 'Algoritmo Genetico (Generacional)',
  'Mejor Aptitud': 0,
  'Tiempo': 1.8490078449249268},
 {'Funcion': 'rosenbrock',
  'Algoritmo': 'Algoritmo Genetico (Elitismo)',
  'Mejor Aptitud': 9,
  'Tiempo': 1.8588769435882568},
 {'Funcion': 'rosenbrock',
  'Algoritmo': 'Algoritmo Genetico (Reemplazo de los peores)',
  'Mejor Aptitud': 0,
  'Tiempo': 1.8642792701721191},
 {'Funcion': 'rosenbrock',
  'Algoritmo': 'Recocido Simulado (Logaritmico)',
  'Mejor Aptitud': 403,
  'Tiempo': 0.0},
 {'Funcion': 'rosenbrock',
  'Algoritmo': 'Recocido Simulado (Decremento Lento)',
  'Mejor Aptitud': 0,
  'Tiempo': 0.013278007507324219}]
```

Los datos anteriores fueron guardados al final del notebook y pueden ser consultados en `resultados_experimentos.csv`

Funcion	Algoritmo	Mejor Aptitud	Tiempo
sphere	Algoritmo Genetico (Generacional)	0.0	1.5239
sphere	Algoritmo Genetico (Elitismo)	1.0	1.4329
sphere	Algoritmo Genetico (Reemplazo de los peores)	1.0	1.4321
sphere	Recocido Simulado (Logaritmico)	5.0	0.0
sphere	Recocido Simulado (Decremento Lento)	0.0	0.0079
sphere	Algoritmo Genetico (Generacional)	0.0	1.4266
sphere	Algoritmo Genetico (Elitismo)	0.0	1.4273
sphere	Algoritmo Genetico (Reemplazo de los peores)	1.0	1.4398
sphere	Recocido Simulado (Logaritmico)	6.0	0.0
sphere	Recocido Simulado (Decremento Lento)	0.0	0.0081
.	.	.	.
.	.	.	.
.	.	.	.
rosenbrock	Recocido Simulado (Logaritmico)	0.0	0.0009
rosenbrock	Recocido Simulado (Decremento Lento)	0.0	0.0136
rosenbrock	Algoritmo Genetico (Generacional)	0.0	1.8656
rosenbrock	Algoritmo Genetico (Elitismo)	0.0	1.8485
rosenbrock	Algoritmo Genetico (Reemplazo de los peores)	0.0	1.8567
rosenbrock	Recocido Simulado (Logaritmico)	103.0	0.001
rosenbrock	Recocido Simulado (Decremento Lento)	0.0	0.0132
rosenbrock	Algoritmo Genetico (Generacional)	0.0	1.849
rosenbrock	Algoritmo Genetico (Elitismo)	9.0	1.8588
rosenbrock	Algoritmo Genetico (Reemplazo de los peores)	0.0	1.864
rosenbrock	Recocido Simulado (Logaritmico)	403.0	0.0
rosenbrock	Recocido Simulado (Decremento Lento)	0.0	0.013

De los resultados obtenidos podemos destacar que en general los algoritmos implementados con las diferentes estrategias llegan al mínimo global de las funciones, con excepción del algoritmo de recocido simulado con esquema de enfriamiento logarítmico. Una hipótesis que tenemos de ello es la velocidad con la que baja la temperatura, por lo tanto, es posible que no llegue al mínimo adecuado y obtenga mínimos mayores al mínimo global. Cabe mencionar que para todos los algoritmos utilizamos 30 iteraciones, de esa forma obtuvimos que el de recocido simulado obtiene menores tiempos de ejecución que el algoritmo genético, con menores tiempos tiempos de ejecución en funciones como sphere y mayores en funciones más complejas como rosenbrock.

Ejercicio 3. Análisis de resultados

Después de realizar la experimentación del ejercicio anterior, se deberá incluir en el reporte lo siguiente:

3 a) **Tabla de configuración de parámetros para los diferentes algoritmos (o estrategias) con las que se haya trabajado.**

```
# Tabla de configuracion de parametros
parametros = {
    'Funcion': ['sphere', 'ackley', 'griewank', 'rastrigin', 'rosenbrock'],
    'Algoritmo': ['Recocido Simulado', 'Recocido Simulado', 'Algoritmo Genetico',
    'Algoritmo Genetico', 'Algoritmo Genetico'],
    'Esquema de Enfriamiento': ['Logaritmico', 'Decremento Lento', '', '', ''],
    'Esquema de Reemplazo': ['', '', 'Generacional', 'Elitismo', 'Peores']
}

df_parametros = pd.DataFrame(parametros)
```

Los datos anteriores fueron guardados al final del notebook y pueden ser consultados en `configuracion_parametros.csv`

Funcion	Algoritmo	Esquema de Enfriamiento	Esquema de Reemplazo
sphere	Recocido Simulado	Logaritmico	
ackley	Recocido Simulado	Decremento Lento	
griewank	Algoritmo Genetico		Generacional
rastrigin	Algoritmo Genetico		Elitismo
rosenbrock	Algoritmo Genetico		Peores

3 b) **Gráficas de evolución de aptitud para diferentes ejecuciones.**

Para lograr lo anterior, implementamos una función para graficar la evolución de aptitud por medio de los resultados obtenidos en el ejercicio 2:

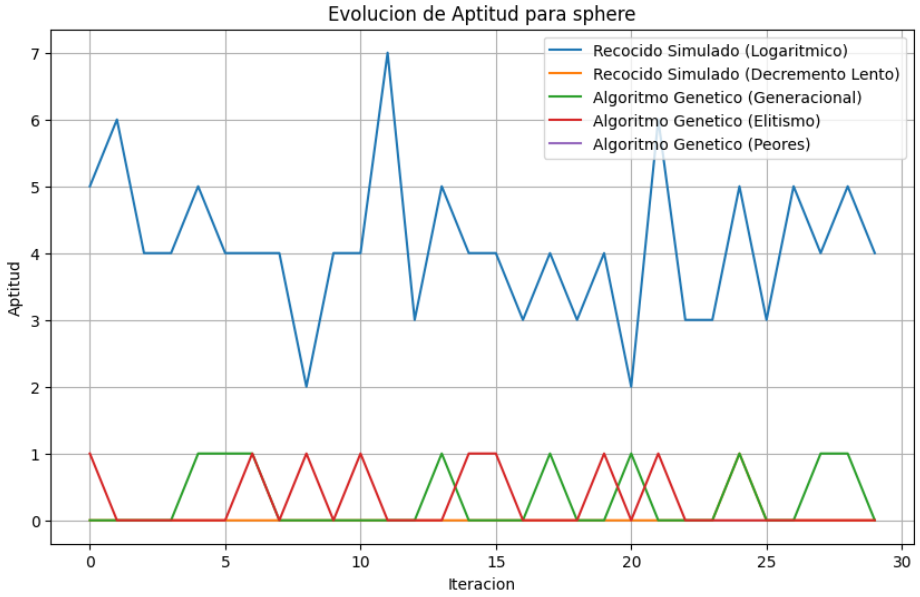
```
def graficar_evolucion_aptitud(resultados, funcion):
    plt.figure(figsize=(10, 6))
    for algoritmo in ['Recocido Simulado (Logaritmico)',
    'Recocido Simulado (Decremento Lento)',
    'Algoritmo Genetico (Generacional)',
    'Algoritmo Genetico (Elitismo)','Algoritmo Genetico (Peores)']:
        aptitudes = [r['Mejor Aptitud']
            for r in resultados
               if r['Funcion'] == funcion and
                          r['Algoritmo'] == algoritmo]
            plt.plot(range(len(aptitudes)), aptitudes, label=algoritmo)
    plt.title(f'Evolucion de Aptitud para {funcion}')
    plt.xlabel('Iteracion')
    plt.ylabel('Aptitud')
    plt.legend()
    plt.grid(True)
    plt.show()

# Graficamos evolucion de aptitud para cada funcion
for funcion in funciones:
    graficar_evolucion_aptitud(resultados_experimentos, funcion)
```

3 c) **Gráficas de evolución promedio de aptitud. Una gráfica por cada ejemplar. En esta gráfica deberían aparecer todas los algoritmos (estrategias) que se compararon.**

Tendremos las siguientes gráficas de evolución de aptitud.

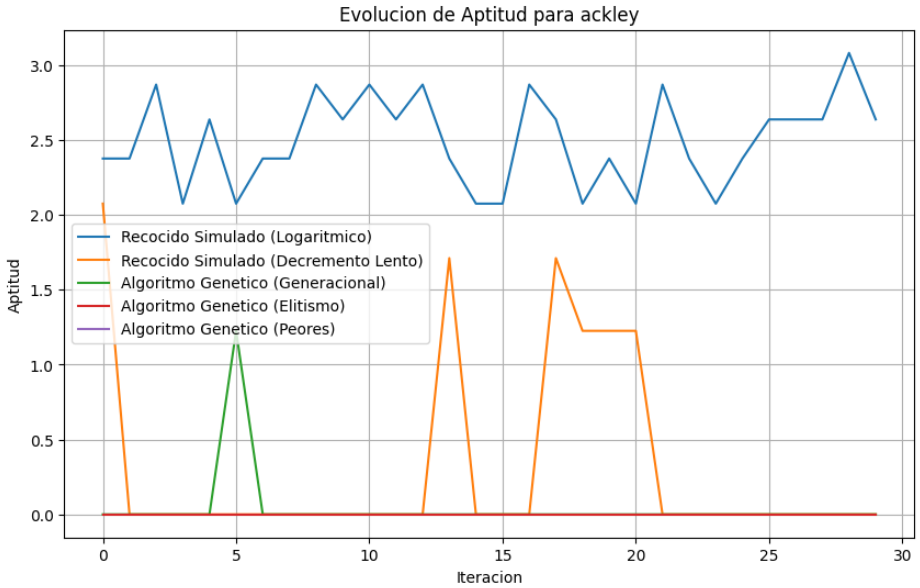
Evolución de Aptitud para Sphere:

Los resultados del algoritmo de recocido simulado con un esquema de enfriamiento logarítmico son generalmente mayores a los de los algoritmos restantes, llegando al óptimo global en la mayoría de las iteraciones, con excepción de algunos puntos en los que sube. Sin embargo, no llegan a ser tan altos como en el esquema de enfriamiento logarítmico.

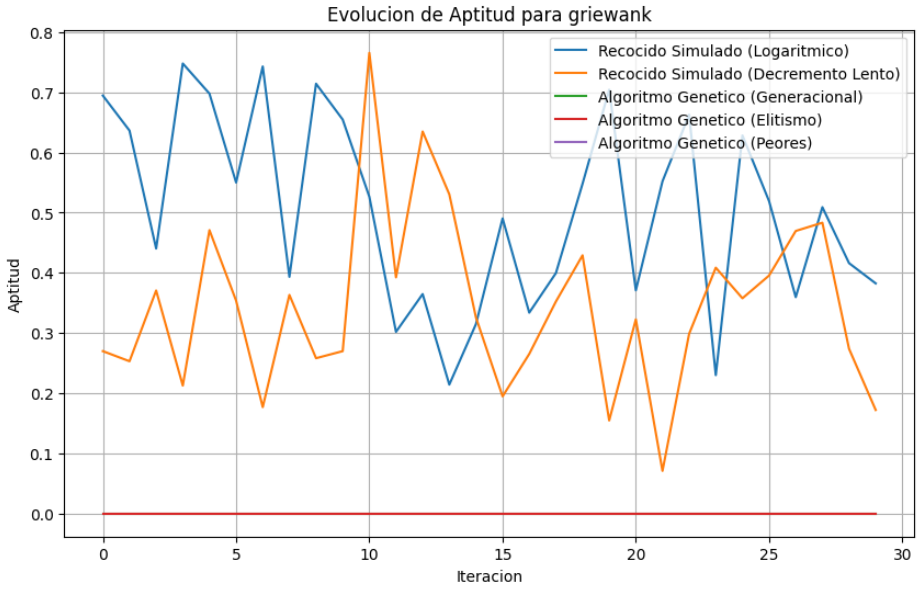


Evolución de Aptitud para Ackley

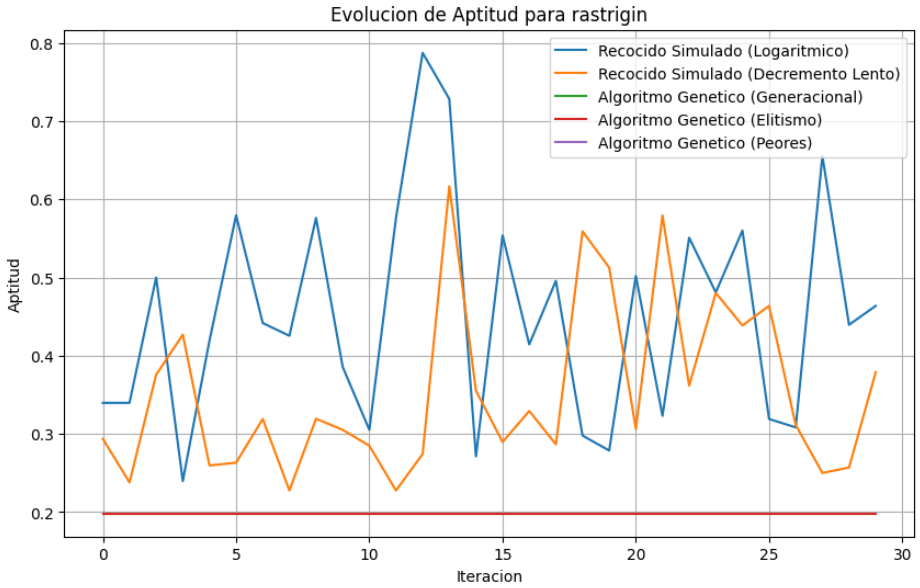
En este caso la gráfica de evolución de aptitud de recocido simulado con esquema de enfriamiento logarítmico de igual forma es mayor que el de otros algoritmos, sin embargo, posee una menor volatilidad en él y una mayor volatilidad en el algoritmo genético con reemplazo generacional y recocido simulado con esquema de enfriamiento de decremento lento.



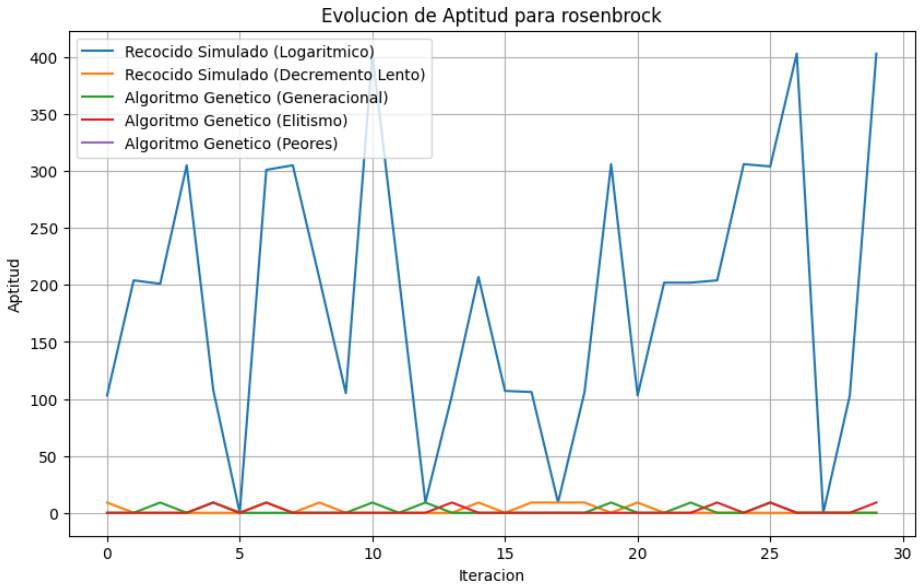
Evolución de Aptitud para Griewank:
Para esta función tenemos que ambos algoritmos de recocido simulado poseen una gran volatilidad, además de no llegar al mínimo global en general. Por otra parte, tenemos que nos algoritmos genéticos en general llegan al mínimo global. Creemos que esto es debido al ruido que posee la función.



Evolución de Aptitud para Rastrigin:
Al igual que en la función anterior, los algoritmos de recocido simulado poseen una volatilidad mayor y resultados distantes al óptimo global, pero más cercanos a él en comparación.



Evolución de Aptitud para Rosenbrock
Por último, para la función de Rosenbrock tenemos una volatilidad mayor en el algoritmo de recocido simulado con esquema de enfriamiento logarítmico, sin embargo, en esta ocasión llega al óptimo global. Además de que obtenemos mejores resultados para todos los algoritmos en general. Pensamos que esto se debe a la cantidad de minimos locales que puede llegar a tener la función.



¿Notas alguna diferencia entre la gráfica de evolución promedio y una ejecución individual, para alguno de los algoritmos?, ¿A qué se debe? Justifica tu respuesta.
En la gráfica de evolución promedio podemos ver como podemos llegar a tener tanto soluciones adecuadas como no adecuadas, por lo cual, al realizar una ejecución individual, puede no ser adecuada y no realizar una buenas estimación del óptimo global.

3 d) **Tabla de resultados con datos estadísticos (mejor, peor, media, mediana, desviación estándar, número de iteraciones o generaciones, etc..)**

Usando lo siguiente:

```
# Convertimos resultados a DataFrame
df_resultados = pd.DataFrame(resultados_experimentos)

# Calculamos las estadisticas
estadisticas = df_resultados.groupby(['Funcion', 'Algoritmo']).agg({
    'Mejor Aptitud': ['min', 'max', 'mean', 'median', 'std'],
    'Tiempo': ['mean']
}).reset_index()
```

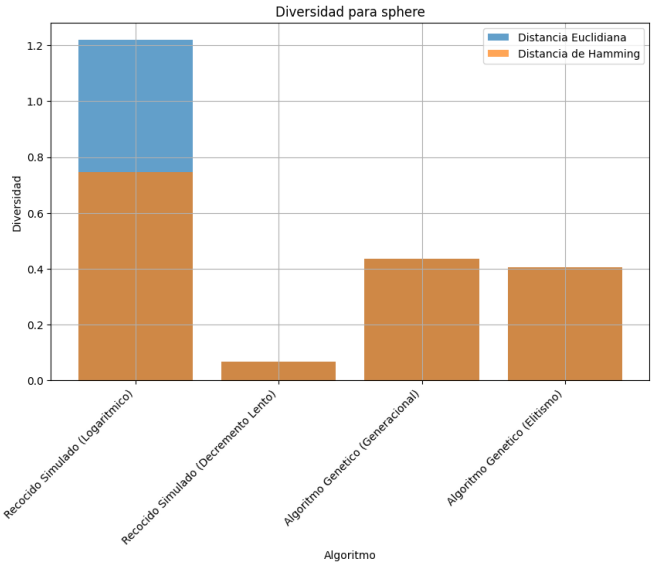
Tendremos acceso a los datos estadísticos, los cuales nuevamente, fueron guardados al final del notebook y pueden ser consultados en `estadisticas_resultados.csv`

Los resultados generales indican que para todas las funciones el algoritmo genético para todas las estrategias de reemplazo, tenemos mejores que para recocido simulado. Entre las estrategias de reemplazo, la que tiene un mayor error promedio es la de reemplazo generacional, en segundo lugar el reemplazo generacional con elitismo y por último el reemplazo de los peores. De la misma forma, el algoritmo de recocido simulado con enfriamiento logarítmico posee un mayor error que el de decremento lento.

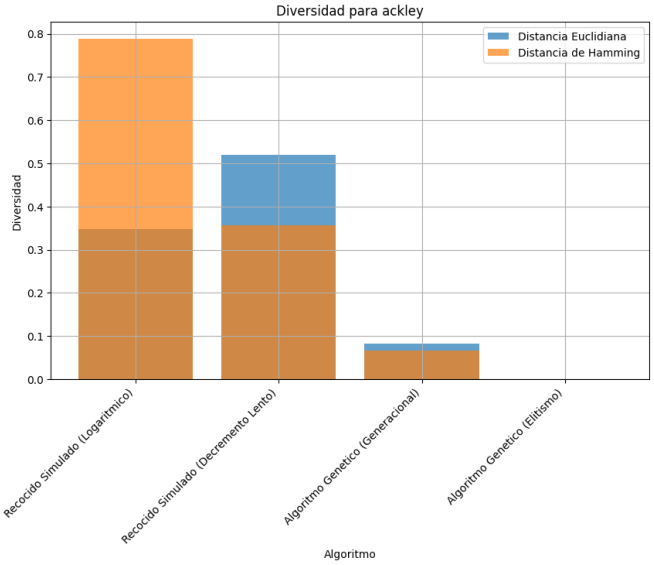
3 e) **Gráficas de diversidad.**
Se deben implementar al menos dos medidas de distancia.
Para cada medida se deben incluir las gráficas correspondientes

Diversidad para Sphere:

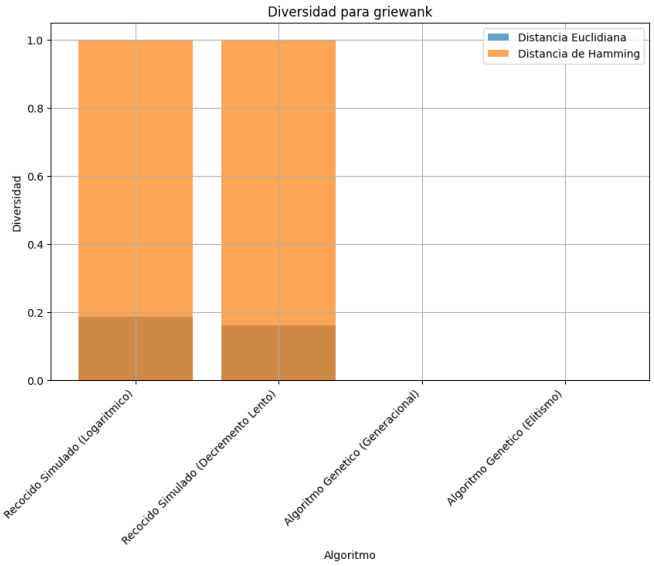
En general, para todos los algoritmos, la diversidad genética no es muy alta y es similar para ambas distancias. Excepto en la estrategia de enfriamiento logarítmico, indicándonos que poseemos una diversidad mucho mayor si tomamos en cuenta una distancia euclidiana.



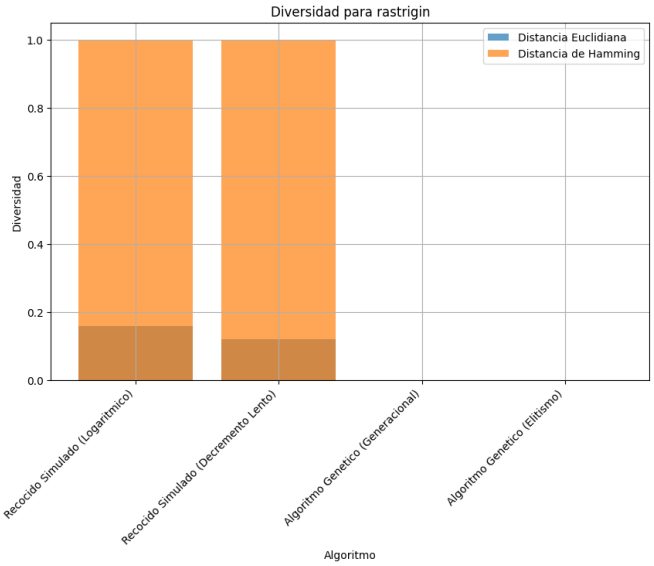
Diversidad para Ackley:
Para esta función la distancia de hamming es mayor que la euclidiana en la estrategia de enfriamiento logarítmico, indicándonos que en general las soluciones distan más por la cantidad de bits modificados que por la distancia de las soluciones.



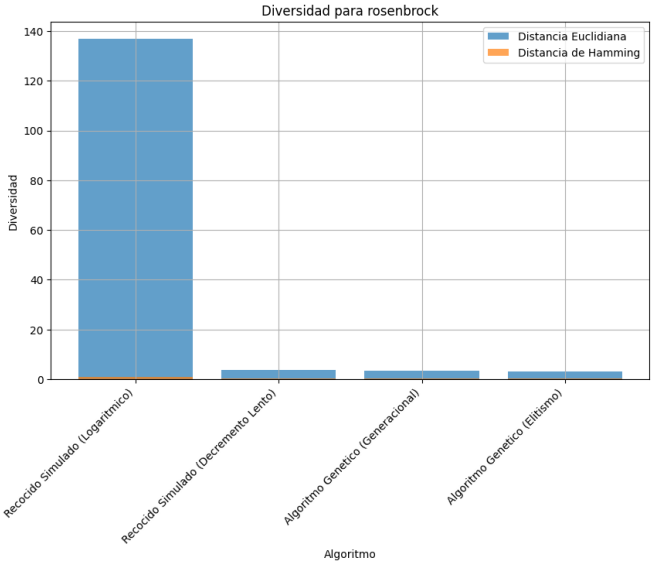
Diversidad para Griewank:
Al igual que en la función anterior, la distancia de hamming es mayor que la euclidiana en el algoritmo de recocido simulado y prácticamente 0 para el algoritmo, indicándonos la uniformidad en estas.



Diversidad para Rastrigin:
Ocurre lo mismo que en la función anterior, creemos que esto se debe al ruido que poseen las funciones.



Diversidad para Rosenbrock:
En este caso, las soluciones poseen una mayor distancia euclidiana que una distancia de hamming, reflejando lo distantes que se encuentran entre ellas, más que sus elementos en sí.

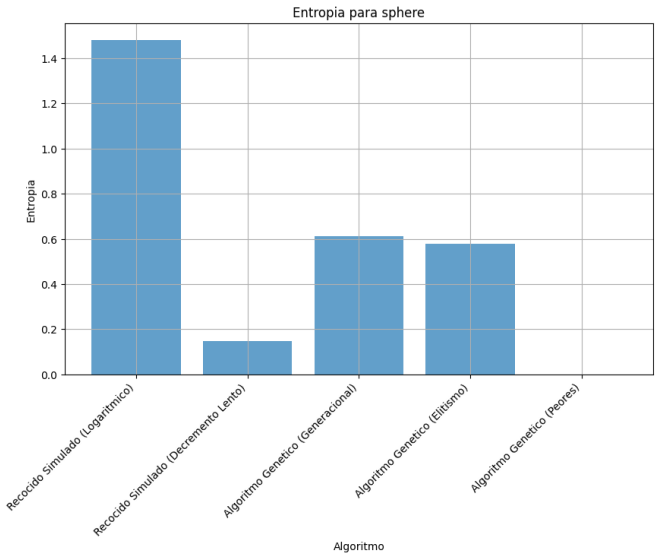


3 f) **Gráficas de entropía.**

La entropía como concepto general puede indicarnos como cambia nuestra población a lo largo de que evoluciona.

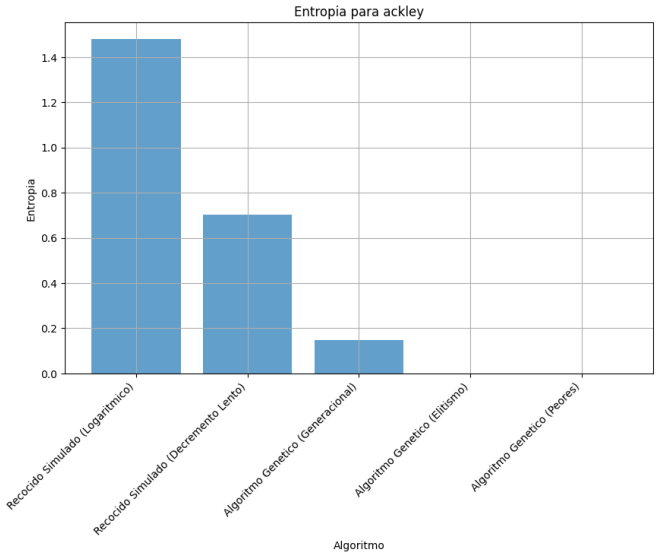
Entropía para Sphere:

Veamos como la entropía se concentra en nuestra primer algoritmo de recocido simulado. Veamos como esta baja en la estrategia de decremento lento, reflejando que no modificamos demasiado a la población, al igual que en la estrategia de reemplazo de los peores.

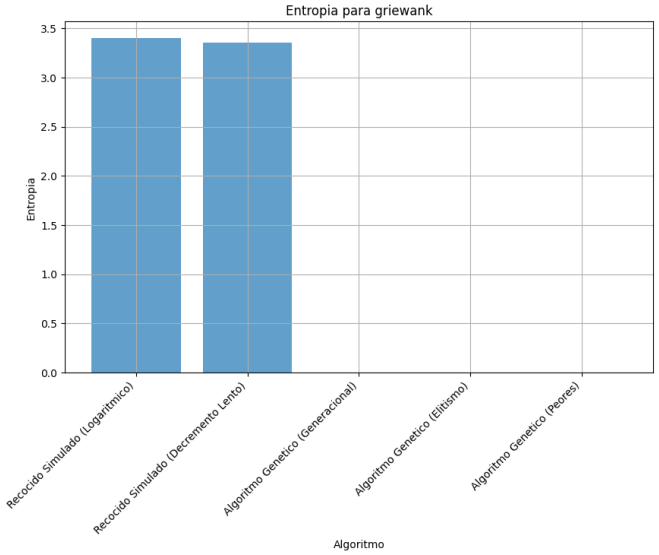


Entropía para Ackley

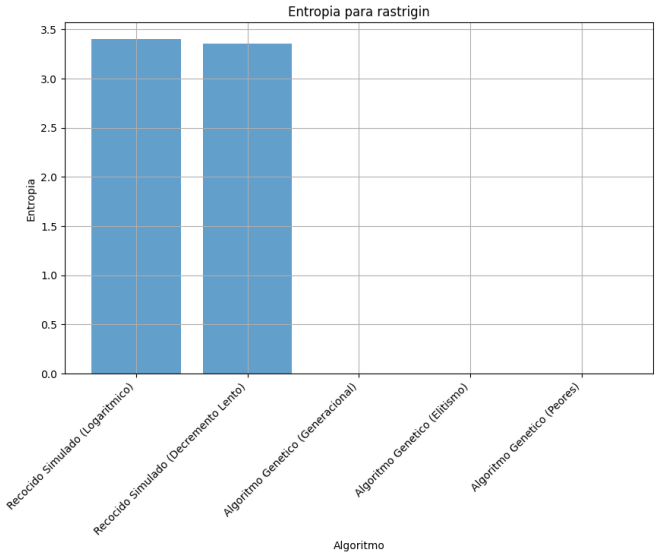
En este caso, la entropía sigue siendo alta en el caso de recocido simulado, esto puede indicarnos, lo volatil que pueden llegar a ser las soluciones de nuestro algoritmo.



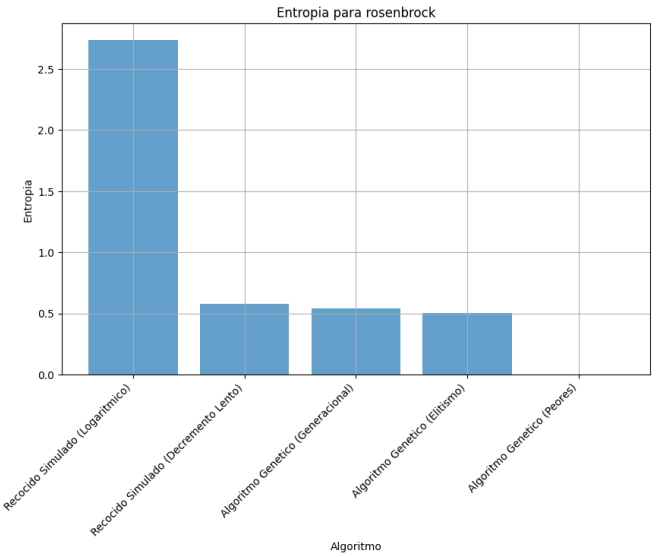
Entropía para Griewank:
Ahora la entropía aumenta mucho para recocido simulado, llegando a ser igual, por lo tanto, esto indica que las soluciones son alteradas de gran manera.



Entropía para Rastrigin:
Ocurre una situación similar que en la función anterior, por otro lado, veamos como ya no tenemos una entropía notoria para los algoritmos genéticos, indicándonos que no cambian demasiado las soluciones, o no existe una gran diversidad de ellas.



Entropía para Rosenbrock:
En esta función se estabiliza la entropía, continuando con el patrón de un resultado alto en el primer algoritmo utilizado.

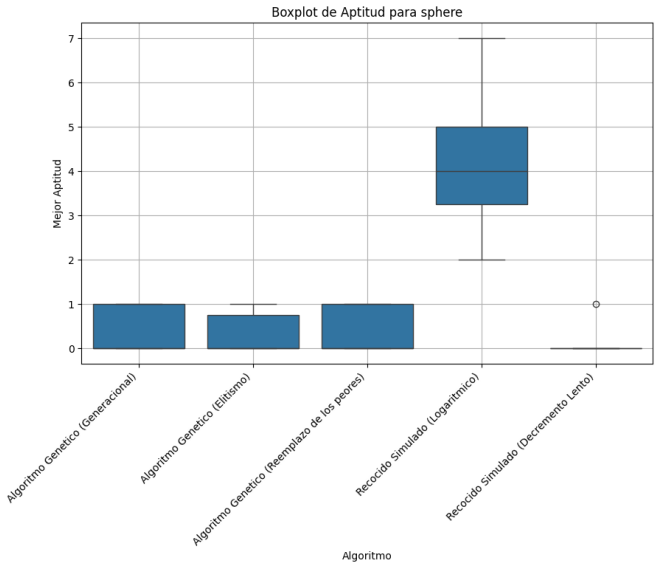


¿Hay diferencias con respecto a las gráficas de diversidad?, ¿Por qué?
Son similares las gráficas de diversidad debido a que están relacionadas. Sin embargo, tienen un significado distinto, con la entropía relacionada con el equilibrio de la población. En ese sentido podemos observar como este equilibrio y modificación se ve más notoriamente en el algoritmo de recocido simulado con enfriamiento logarítmico y menor en los algoritmos evolutivos.

3 g) **BoxPlot, uno por cada ejemplar.**

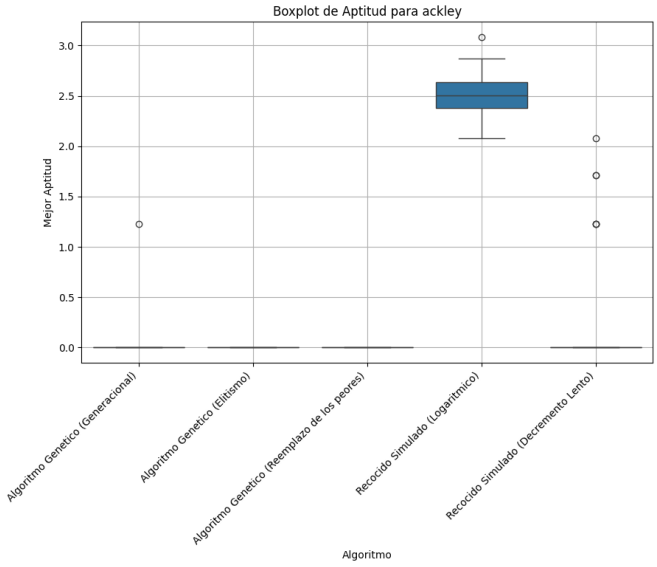
Boxplot de Aptitud para Sphere

Notemos como es que con Sphere tuvimos la mejor aptitud con recocido simulado (logarítmico) mientras que con los algoritmos genéticos tuvimos resultados muy similares, sin embargo con recocido simulado (decremento lento) tuvimos resultados bastante malos.



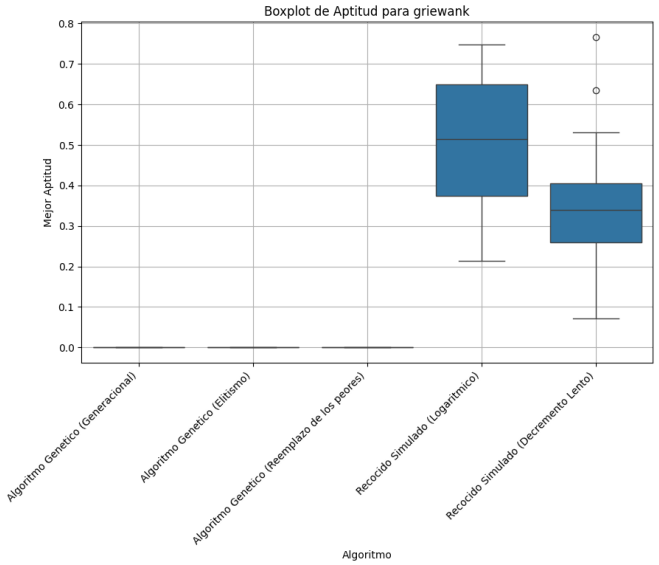
Boxplot de Aptitud para Ackley

Notemos como es que con Ackley tuvimos malos desempeños de aptitud en todas menos en una, en recocido simulado (logarítmico) tuvimos resultados de aptitud bastante buenos además de que tuvimos un mejor desempeño que con Sphere.



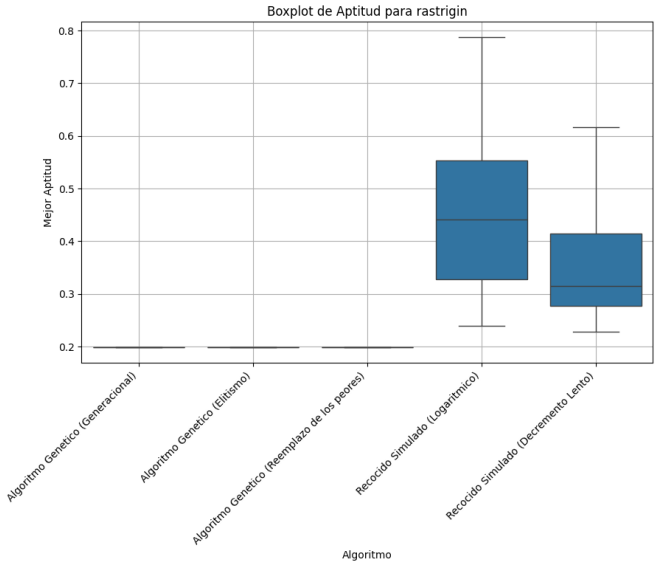
Boxplot de Aptitud para Griewank

Como podemos observar tuvimos una aptitud bastante mal con los algoritmos genéticos, sin embargo tuvimos desempeños aceptables de parte de recocido simulado (decremento lento) y unos resultados bastante buenos de recocido simulado (logarítmico).



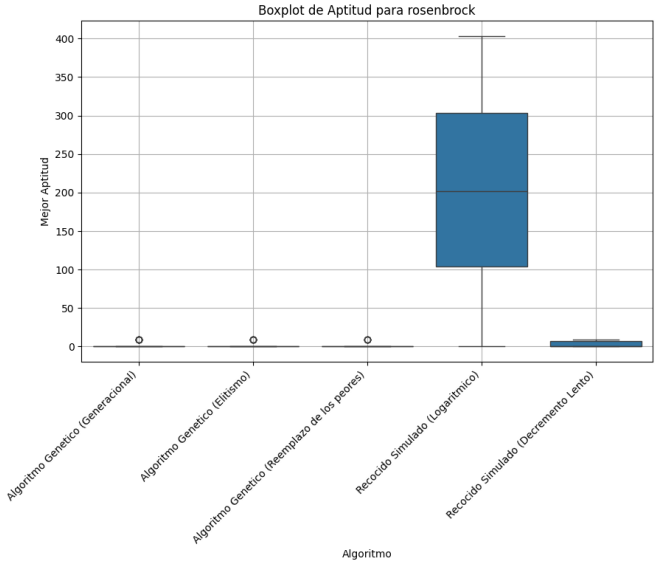
Boxplot de Aptitud para Rastrigin

Observemos como nuevamente tuvimos este comportamiento nada favorecedor en los algoritmos genéticos y con los únicos que tuvimos una aptitud aceptable fueron recocido simulado (decremento lento) y recocido simulado (logarítmico), este ultimo teniendo los mejores resultados otra vez.



Observemos como es que con Rosenbrock tuvimos resultados de aptitud realmente con todas, e incluso la única que nos brindo resultados que fue recocido simulado (logarítmico) no se salvo de esto y nos dio aptitudes en cierta forma aceptables (mas no buenas o realmente malas como las demás).

Boxplot de Aptitud para Rosenbrock



Conclusiones:

Con base en las gráficas y resultados estadísticos obtenidos anteriormente, podemos concluir que los resultados y óptimos globales más desfavorecedores son los del algoritmo de recocido simulado con decremento logarítmico, con soluciones más volátiles en cuanto a su diversidad, entropía y evolución de aptitud. En comparación con el algoritmo genético con reemplazo de los peores, quienes nos permiten obtener soluciones muy cercanas al óptimo global, con una diversidad baja y volatilidad baja de la evolución de aptitud.

Consideraciones Generales

➢ El reporte debe considerar el pseudocódigo de todos los componentes utilizados, una justificación de los parámetros seleccionados para la ejecución del algoritmo, comentarios de los resultados obtenidos (tabla y gráficas) así como conclusiones generales.

El entregable para esta tarea deberá ser un archivo zip con la siguiente estructura:

```
+ # cuenta / <--- Nombre de la carpeta
    - src / <--- Carpeta con el código fuente de su implementación
    - output / <--- Carpeta con gráficas y cualquier otro archivo
                    generado
    - README.txt <--- Archivo con instrucciones para compilar y ejecutar
                      se debe incluir el comando para ejecutar un ejemplo
                      de cada inciso
    - makefile <-- [Opcional]
    - reporteT5.pdf <--- Reporte de la tarea
    - ejecuciones.csv <-- [Opcional] Hoja de cálculo con la información de las
                                     ejecuciones realizadas.
```

El reporte (reporteT5.pdf) deberá incluir al menos:

- Nombre completo
- Título y número de Tarea
- Respuestas a los ejercicios planteados
- Comentarios / Conclusiones

El formato para el reporte es libre. Pueden usar Word, LaTeX, o cualquier otro procesador; es obligatorio que se incluya el archivo pdf. No hay extensión mínima ni máxima, aunque se sugiere considerar un reporte entre 5-10 páginas, pero deben incluir las respuestas / comentarios que se piden en cada uno de los ejercicios.

Referencias

- https://www.sfu.ca/ssurjano/index.html
- https://patentimages.storage.googleapis.com/a3/d7/f2/0343f5f2c0cf50/US2632058.pdf