Comparativa de frameworks para cómputo evolutivo implementados en Python

Guillermo Ramírez Solís
Depto. de ingeniería Electrónica
Universidad de Guanajuato
Salamanca, Gto.
Email: g.ramirezsolis@ugto.mx

Felipe Trujillo-Romero
Depto. de ingeniería Electrónica
Universidad de Guanajuato
Salamanca, Gto.
Email: fdj.trujillo@ugto.mx

Carlos Hugo García Capulín
Depto. de ingeniería Electrónica
Universidad de Guanajuato
Salamanca, Gto.
Email: carlosg@ugto.mx

***Resumen*—Cada vez es más generalizado la utilización de algoritmos de inteligencia artificial y en particular aquellos que están clasificados dentro del cómputo evolutivo. Por esa razón, en el presente artículo se realiza una comparativa de cuatro diferentes librerías o frameworks orientados al desarrollo de algoritmos de computo evolutivo. Las librerías que se compararon fueron: 1) DEAP, 2) PyGAD, 3) GAFT y 4) MEALPY. Todas estas librerías están implementadas usando Python como lenguaje de programación. La métrica principal utilizada para realizar la comparativa entre los frameworks fue la velocidad de ejecución de un problema clásico de optimización. Sin embargo, también se consideró el uso tanto de memoria como del procesador; así como la facilidad de instalación, de implementación de código, y el soporte que ofrecen.**

***Palabras claves*—Computo evolutivo, Algoritmos genéticos, Python, framework, funciones de benchmark.**

I. Introducción

Los algoritmos genéticos (GA) son una técnica de búsqueda y optimización computacional inspirada en los principios de la evolución biológica. El desarrollo de esta idea se remonta a la década de 1960, cuando John Holland llevó a cabo una investigación innovadora en este campo. Holland [1] fue pionero en la utilización de principios evolutivos, inspirándose en la genética y la selección natural, para abordar los desafíos de optimización y búsqueda dentro del campo de la informática.

Los algoritmos genéticos comenzaron a utilizarse en diversos campos a partir de la década de 1980, desde la programación hasta el diseño de ingeniería. Esto permitió la generación de nuevas aplicaciones, además de demostrar la adaptabilidad y la eficiencia de este tipo de algoritmos para resolver diferentes problemas. Sin embargo, no fue hasta los años 1990 y principios de los 2000, donde los algoritmos genéticos experimentaron una gran popularidad donde se enfocó en el desarrollo de variaciones y mejoras, como por los algoritmos genéticos paralelos, híbridos, así como al desarrollo de la aplicaciones en problemas de optimización más complejos. En la actualidad, los algoritmos genéticos se han convertido en una herramienta esencial en la optimización, abordando una amplia gama de problemas en campos como finanzas, biología, robótica y más.

Ahora bien, el desarrollo de un algoritmo genético requiere de una serie de pasos generales los cuales se deben de realizar para cualquier implementación que se desee realizar y que ene este caso se van a aplicar con las bibliotecas que se evaluaran. Los pasos del algoritmo genético son:

- **Inicialización:** En la cual se define la representación de los individuos así como el tamaño de la población y los parámetros iniciales del algoritmo.
- **Creación de la Población Inicial:** Se genera una población genética inicial de individuos utilizando la representación genética definida.
- **Evaluación de la Aptitud:** Se evalúa la aptitud de cada individuo en la población utilizando la función de aptitud definida para el problema.
- **Ciclo Evolutivo:** Se aplican los operadores de selección, cruce y mutación para generar nuevos individuos. Además se evalúa la aptitud y se actualiza la población con los individuos generados.
- **Criterio de Terminación:** Se repite el ciclo evolutivo hasta que se cumpla un criterio de terminación, como un número máximo de generaciones o una convergencia aceptable.

Revisemos a continuación algunos trabajos en donde se ejemplifican diferentes aplicaciones que utilizan algoritmos genéticos. Comencemos con el trabajo realizado por Zhi y Liu [2] quienes presentaron un modelo robusto de reconocimiento facial que integra análisis de componentes principales, algoritmo genético y máquina de vectores de soporte. En este trabajo [2] se utiliza un algoritmo genético para optimizar la estrategia de búsqueda obteniendo una tasa de precisión máxima que alcanza el 99 %. En [3], Malekli et al. han utilizado los algoritmos genéticos para optimizar la selección de características y mejorar el rendimiento de un clasificador que determina el estado de la enfermedad de los pacientes con cáncer de pulmón. Esta metodología presenta un nivel de precisión del 100 % en clasificación de la enfermedad. Por su parte, Kordos et al. [4], han optimizado la colocación discreta de productos y de las rutas de preparación de pedidos en un almacén mediante el uso de algoritmos genéticos. Dicha optimización de las rutas de preparación de pedidos permitió reducir los tiempos totales de preparación de pedidos al 54 %. En [5], se propuso un sistema para reducir los casos de fraude con tarjetas de crédito, mediante la construcción de un clasificador de detección de fraude más preciso, el cual

hace uso de un método de muestreo basado en la agrupación de K-means y algoritmos genéticos.

Un campo en el que los algoritmos genéticos son muy usados en dentro de la robótica tanto en robótica de manipuladores como en robótica móvil. Por ejemplo, Ji et al. [6] determinan la velocidad al caminar más rápida y la eficiencia energética más baja de los robots bípedos mediante el uso de empuje de tobillo, optimizando la trayectoria del robot mediante algoritmos genéticos. En [7], se propuso un enfoque para la síntesis de la marcha de un robot humanoide mediante algoritmos evolutivos; dicha síntesis se orienta hacia dos escenarios que son el caminar y subir escaleras. Por otra parte, Lamini et al. [8] implementan un operador de cruce mejorado para resolver problemas de planificación de rutas utilizando la optimización mediante algoritmos genéticos en un entorno estático para un robot móvil. También Liu et al. [9] han desarrollado un método para optimizar la trayectoria del robot móvil pero basado en el gemelo digital de este; en donde el robot se entrena en un entorno de simulación el cual proporciona la información para el movimiento real del robot, y en función de los datos reales devueltos, se ajusta la trayectoria del robot virtual mejorando así el desempeño del robot real. En [10] se implementa un planificador de rutas para un robot móvil mediante la mejora un algoritmo genético usando el algoritmo bio-inspirado de optimización del lobo gris [11].

Los trabajos antes mencionados solo son una muestra de lo que se puede encontrar en la literatura acerca del uso de los algoritmos genéticos en diferentes áreas de estudio. Por lo que, tanto los algoritmos genéticos como sus variantes representan una valiosa contribución al conjunto de herramientas de optimización y permiten ofrecer soluciones efectivas para problemas complejos. Sin embargo, es necesario tener un entorno de programación que permita la implementación de diferentes aplicaciones usando algoritmos evolutivos. En este rubro se han desarrollado diversas librerías, frameworks o códigos específicos para implementar software con algoritmos genéticos. Por esa razón, en el presente artículo se evaluaron tres de estas librerías con la finalidad de determinar cual de ellas permite un desarrollo más eficiente y amigable. Las librerías evaluadas fueron DEAP [12], PyGAD [13], GAFT [14], y MEALPY [15].

La estructura del artículo es como se describe a continuación. En la sección II, se presenta una breve descripción de los frameworks evaluados. Mientras que la evaluación de dichas librerías se realiza en la sección III. Los resultados obtenidos de este análisis se muestra en la sección IV. Finalmente, en la sección V se realiza una discusión y las conclusiones de los resultados obtenidos.

II. Librerías de Algoritmos evolutivos

En esta sección se comentarán de manera breve cada uno de las librerías que se consideraron para esta evaluación, mencionando la descripción de cada uno de ellos.

A. DEAP (Distributed Evolutionary Algorithms in Python)

DEAP [12] (Distributed Evolutionary Algorithms in Python) es una biblioteca de algoritmos evolutivos distribuidos implementada en Python. Este framework es de código abierto la cuál, de acuerdo con el desarrollador, esta diseñada para facilitar la implementación y ejecución de algoritmos evolutivos distribuidos. DEAP proporciona herramientas y estructuras de datos que permiten la creación de algoritmos evolutivos escalables y paralelizables, lo que significa que pueden aprovechar la potencia de los sistemas distribuidos y paralelos para acelerar la búsqueda de soluciones óptimas en problemas complejos. La biblioteca incluye funcionalidades para representar individuos, definir operadores genéticos, evaluar la aptitud, aplicar estrategias de selección y ejecutar algoritmos evolutivos de forma distribuida para abordar problemas a gran escala.

Además, DEAP es una biblioteca ampliamente utilizada para implementar algoritmos evolutivos en Python. Para instalar DEAP, se deben de ejecutar el siguiente comando:

```
pip install deap
```

B. PyGAD (Python Genetic Algorithm Library)

PyGAD [13] (Python Genetic Algorithm Library) es una biblioteca realizada en Python la cuál es de código abierto que, según el desarrollador, proporciona un marco sencillo y flexible para implementar algoritmos genéticos (AG). PyGAD ofrece una selección completa de parámetros que brindan al usuario un control considerable sobre todos los aspectos de su ciclo de vida del programa. Esto abarca varios factores, como el tamaño de la población, el rango de valores genéticos, el tipo de datos de los genes, la metodología de selección de padres, la técnica de cruce y la estrategia de mutación. Además, PyGAD ofrece a los usuarios la flexibilidad de personalizar la función de fitness. La biblioteca admite el entrenamiento de modelos de aprendizaje profundo, que se pueden desarrollar utilizando PyGAD u otros marcos como Keras y PyTorch.

Dado que PyGAD puede ser una biblioteca eficiente para el desarrollo de algoritmos genéticos en Python, si alguien quiere probarla se puede instalar utilizando el comando `pip` de la siguiente manera:

```
pip install pygad
```

C. GAFT (Genetic Algorithm Framework in Python)

GAFT (Genetic Algorithm Framework in Python) es una biblioteca Python de código abierto que proporciona un marco versátil y extensible para implementar algoritmos genéticos (AG). Los algoritmos genéticos son algoritmos de optimización y búsqueda inspirados en los principios de la selección natural. GAFT ofrece diversas características, como la representación personalizable de individuos, varios operadores genéticos, funciones flexibles de evaluación de la aptitud y diversas estrategias de selección. Esta biblioteca permite a los usuarios diseñar y ejecutar algoritmos genéticos adaptados a problemas de optimización específicos, lo que la convierte en una valiosa herramienta para investigadores y profesionales del campo de la computación evolutiva.

Como una alternativa para desarrollar programas ude computo evoultivo en Python usando GAFT se debe de instalar de la siguiente manera:

```
pip install gaft
```

D. *MEALPY (Metaheuristic algorithms in Python)*

MEALPY [15] (Metaheuristic algorithms in Python) es una biblioteca realizada en Python la cuál es multiplataforma y de código abierto orientada para algoritmos de optimización inspirados en la naturaleza. MEALPY contiene una gran cantidad de algoritmos metaheurísticos clásicos así como de última generación, contando con más de 160 algoritmos. MEALPY se puede utilizar para resolver problemas prácticos tanto en el ámbito de la investigación como en la optimización de problemas del mundo real.

Al igual que los otros frameworks evaluados la instalación de MEALPY se realiza mediante el comando `pip`:

```
pip install mealpy
```

III. Evaluación de desempeño

En esta sección se presentarán las diferentes evaluaciones realizadas a los framework considerados para la comparativa. Además, dado que cada framework tiene diferentes algoritmos implementados esta evaluación se enfoca en la implementación de un algoritmo genético ya que es una implementación común a los frameworks a comparar. Por esa razón, para este trabajo se eligió la implementación del algoritmo One-Max. Comenzremos entonces definiendo el problema usado para la evaluación, en este caso el algoritmo de One-Max.

El problema One-Max se considera sencillo y se emplea con frecuencia como ejemplo introductorio para ilustrar los principios de los algoritmos genéticos. El enfoque es sencillo pero muy eficaz para familiarizarse con los componentes esenciales de un algoritmo genético. Este problema puede resumirse en la siguiente pregunta: ¿Cuál es el valor máximo de la sumatoria de una secuencia compuesta únicamente de unos y ceros con una longitud de N?

Es obvio que la suma más grande que puede obtener una cadena de bits de longitud N es igual a N. Sin embargo, sería necesario examinar exhaustivamente 2^N respuestas distintas para demostrar esta afirmación mediante una búsqueda de fuerza bruta. Encontrar una solución para este problema en particular no es difícil cuando se trata de cadenas de bits de tamaños pequeños. Sin embargo, si aplicamos esta técnica a cadenas de bits con una longitud de 40, se tendría que buscar entre una cantidad de cadenas de bits que se estiman en más de un billón. Para suavizar este problema, se implementa un algoritmo genético para generar una solución ideal, evitando así la necesidad de iterar a través de todas las soluciones imaginables dentro del espacio de búsqueda.

Bien, una vez definido el problema que será implementado en cada uno de los frameworks evaluados se procede a las implementación en cada uno de estos y a obtener los parámetros correspondientes a la evaluación. Lo anterior con el objetivo de determinar cual de entre ellos puede ser la mejor opción para implementar un algoritmo evolutivo mediante algoritmos genéticos.

A. *DEAP*

Empezaremos con la implementación del algoritmo de One-Max en lel framework DEAP, por lo que en este apartado, se presenta una explicación detallada del código proporcionado, que utiliza el framework de DEAP para resolver un problema de optimización de One-Max. Como ya se comento el objetivo es encontrar un individuo, formado por una lista de 0's y 1's, que maximice la suma de sus elementos.

```
CODIGO 1.
Implementación del algoritmo One-Max usando DEAP

import random
from deap import base, creator, tools

creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMax)

toolbox = base.Toolbox()
toolbox.register("attr_bool", random.randint, 0, 1)
toolbox.register("individual", tools.initRepeat, creator.Individual,toolbox.attr_bool, 100)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

def evalOneMax(individual):
    return sum(individual),

toolbox.register("evaluate", evalOneMax)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
toolbox.register("select", tools.selTournament, tournsize=3)

def main():
    pop = toolbox.population(n=300)
    :  % Omitiendo algunas líneas para concisión
    CXPB, MUTPB = 0.5, 0.2

    g = 0
    while max(fits) < 100 and g < 1000:
        g = g + 1
        :  % Omitiendo algunas líneas para concisión
        pop[:] = offspring
# Resto del código...
```

En el CODIGO 1 se puede observar la implementación del algoritmo One-MAx realizado en el framework de DEAP. El código comienza importando los módulos necesarios, incluyendo `random` para la generación de números aleatorios y `deap` para el uso de su marco de trabajo.

A continuación, se crean dos clases utilizando `creator.create`: `FitnessMax` para definir la función de aptitud y `Individual` para definir la estructura de un individuo, que es una lista de elementos.

Se registra un atributo y operadores genéticos en la `toolbox` para la creación de individuos y poblaciones. El atributo `attr_bool` representa un gen que puede ser 0 o 1.

La función `evalOneMax` calcula la aptitud de un individuo como la suma de sus elementos.

Posteriormente, se registran los operadores de cruce, mutación y selección en la `toolbox` utilizando DEAP. La función `main` contiene el bucle principal de evolución, que incluye la inicialización de la población, la evaluación de la aptitud, la creación de nuevas generaciones y la aplicación de operadores genéticos.

Se inicializa la población con 300 individuos utilizando la función `toolbox.population()`. Además, se definen las probabilidades de crossover (CXPB) y mutación (MUTPB) a 0.5 (50 %) y 0.2 (20 %) respectivamente.

Después de cada generación, se reemplaza la población anterior con la nueva generación generada a través de operadores de selección, cruzamiento y mutación. Esto se logra con la línea `pop[:] = offspring`, donde `offspring` representa la nueva generación de individuos.

Tabla I
MÉTRICAS OBTENIDAS PARA EL FRAMEWORK DEAP

Métrica	**Valor**
Media del CPU Usage (%)	3,79
Máximo del CPU Usage (%)	6,7
Mínimo del CPU Usage (%)	2,1
Media del Memory Usage (%)	51,29
Máximo del Memory Usage (%)	63,2
Mínimo del Memory Usage (%)	47,2
Media del Execution Time (s)	0,49
Máximo del Execution Time (s)	0,52
Mínimo del Execution Time (s)	0,47
Desviación estándar del CPU Usage (%)	1,30
Desviación estándar del Memory Usage (%)	4,99
Desviación estándar del Execution Time (s)	0,02

En la Tabla I se presentan las métricas obtenidas, en las cuales podemos observar tanto el tiempo de ejecución como el uso de CPU y memoria. En estos podemos ver que presenta un tiempo de ejecución promedio es menos de un segundo, de hecho el tiempo máximo tampoco llega al segundo. Respecto al porcentaje de uso de CPU y de memoria son aceptables aunque se comentaran más adelante.

B. PyGAD

Ahora toca el turno de analizar la implementación del algoritmo genético para resolver el problema One Max. En este caso el código ha sido desarrollado utilizando la biblioteca `PyGAD` en Python y `numpy` para operaciones numéricas.

El código (Ver CODIGO II) comienza definiendo la función de aptitud (`fitness_func`) que calcula la aptitud de una solución en el problema One Max. La aptitud es simplemente la suma de los elementos de la solución, lo que representa el objetivo del problema.

A continuación, se establecen los parámetros clave para configurar el algoritmo genético. Estos parámetros incluyen el número de generaciones, el tamaño de la población, el número de genes por solución, la probabilidad de mutación y cruce, y otros parámetros relacionados con la operación del algoritmo genético.

```
CODIGO 2.
Implementación del algoritmo One-Max usando PyGAD

import pygad
import numpy as np

num_generations = 1000
num_parents_mating = 300
sol_per_pop = 300
num_genes = 100
gene_space = {"low":0, "high": 1}
mutation_percent_genes = 2
mutation_type ="swap"
crossover_probability=0.5
mutation_probability=0.2
crossover_type="single_point"
parent_selection_type = "tournament"
last_fitness = 0

def on_generation(ga_instance):
    global last_fitness
    print(f"Generation = {ga_instance.↩
    generations_completed}")
    print(f"Fitness    = {ga_instance.↩
    best_solution(pop_fitness=ga_instance.↩
    last_generation_fitness)[1]}")
    print(f"Change     = {ga_instance.↩
    best_solution(pop_fitness=ga_instance.↩
    last_generation_fitness)[1] - last_fitness}")
    last_fitness = ga_instance.best_solution(↩
    pop_fitness=ga_instance.↩
    last_generation_fitness)[1]

ga_instance = pygad.GA(num_generations=↩
    num_generations,
                       num_parents_mating=↩
    num_parents_mating,
                       sol_per_pop=sol_per_pop,
                       num_genes=num_genes,
                       fitness_func=fitness_func,
                       gene_space=gene_space,
                       mutation_type=mutation_type↩
    ,
                   crossover_probability=↩
    crossover_probability,
                       mutation_probability=↩
    mutation_probability,
                       crossover_type=↩
    crossover_type,
                       parent_selection_type=↩
    parent_selection_type,
                       on_generation=on_generation↩
    )

# ... (resto del código)
```

Se define también una función de retorno de llamada (`on_generation`) que se invoca después de cada generación. Esta función imprime detalles sobre la generación actual, incluyendo el número de generación, la aptitud y el cambio en la aptitud con respecto a la generación anterior.

A continuación, se crea una instancia del algoritmo genético (`ga_instance`) utilizando la biblioteca `PyGAD` con los parámetros definidos anteriormente. Se especifica la función de aptitud y la función de retorno de llamada. Luego, se ejecuta el algoritmo genético llamando a `run()`, lo que inicia la optimización y evolución.

Tabla II
MÉTRICAS OBTENIDAS PARA EL FRAMEWORK PYGAD

Métrica	Valor
Media del CPU Usage (%)	3,5533
Máximo del CPU Usage (%)	4,8
Mínimo del CPU Usage (%)	2,6
Media del Memory Usage (%)	59,5666
Máximo del Memory Usage (%)	68,0
Mínimo del Memory Usage (%)	53,6
Media del Execution Time (s)	17,42329
Máximo del Execution Time (s)	18,25156
Mínimo del Execution Time (s)	16,922
Desviación estándar del CPU Usage (%)	0,5976
Desviación estándar del Memory Usage (%)	4,9148
Desviación estándar del Execution Time (s)	0,36

De igual manera que para la implementación del algoritmo One-Max pero utilizando la librería de PyGAD se obtienen los valores que se muestran en la Tabla II. En esta Tabla II se puede observar los resultados del tiempo de ejecución así como el uso de CPU y memoria. En estos podemos ver que presenta un tiempo de ejecución promedio más alto que en el apartado anterior llegando a 17.42 segundos. Lo relacionado con el porcentaje de uso de CPU y de memoria se comentaran más adelante.

C. *GAFT*

Prosiguiendo con la evaluación de las librerías se analizara la implementación del algoritmo genético pero ahora utilizando el framework *GAFT*. En el CODIGO III, se muestra la configuración de la estructura del individuo binario, la creación de operadores genéticos y la definición de funciones clave, todo en el contexto de un algoritmo genético. Los párrafos siguiente presentan la explicación cada parte del código y su función en la optimización de una función binaria.

En la sección de importación de bibliotecas, se incluyen las librerías necesarias para la ejecución del algoritmo genético, con especial énfasis en la biblioteca *numpy* para operaciones numéricas eficientes.

A continuación, se define la plantilla del individuo binario y se crea la población inicial. La plantilla define la estructura de un individuo, con 100 genes binarios en el rango [0, 1]. Posteriormente, se inicializa una población de 300 individuos utilizando la plantilla definida.

Se procede a crear los operadores genéticos, que son esenciales para el funcionamiento del algoritmo genético. Se utilizan operadores como selección por torneo, crossover uniforme y mutación de bit invertido, con probabilidades específicas para cada operación.

El motor del algoritmo genético se crea con la población de individuos y los operadores previamente definidos. Además, se utiliza *FitnessStore* para almacenar información sobre la aptitud de los individuos durante la ejecución del algoritmo.

La función de aptitud es fundamental en un algoritmo genético. En este caso, se define como la suma de los genes binarios del individuo. Esta función será maximizada por el algoritmo genético a lo largo de las generaciones.

```
CODIGO 3.
Implementación del algoritmo One-Max usando GAFT

import numpy as np
from gaft import GAEngine
from gaft.components import BinaryIndividual, Population
from gaft.operators import TournamentSelection, UniformCrossover, FlipBitMutation
from gaft.plugin_interfaces.analysis import OnTheFlyAnalysis
from gaft.analysis.fitness_store import FitnessStore

indv_template = BinaryIndividual(ranges=[(0, 1)] * 100, eps=0.001)
population = Population(indv_template=indv_template, size=300).init()

engine = GAEngine(
    population=population, selection=selection,
    crossover=crossover, mutation=mutation,
    analysis=[FitnessStore]
)

@engine.fitness_register
def fitness(indv):
    return sum(indv.solution)

    def register_step(self, g, population, engine):
        best_indv = population.best_indv(engine.fitness)
        msg = 'Generation: {}, best fitness: {:.3f}'.format(g, engine.ori_fmax)
        self.logger.info(msg)

    def finalize(self, population, engine):
        best_indv = population.best_indv(engine.fitness)
        x = np.round(best_indv.solution)
        y = engine.ori_fmax
        msg = 'Optimal solution: ({}, {})'.format(x, y)
        self.logger.info(msg)

if __name__ == '__main__':
    engine.run(ng=200)
```

Tabla III
MÉTRICAS OBTENIDAS PARA EL FRAMEWORK GAFT

Métrica	Valor
Media del CPU Usage (%)	5,0666
Máximo del CPU Usage (%)	6,9
Mínimo del CPU Usage (%)	1,9
Media del Memory Usage (%)	59,7933
Máximo del Memory Usage (%)	64,7
Mínimo del Memory Usage (%)	53,9
Media del Execution Time (s)	110,46727
Máximo del Execution Time (s)	151,979
Mínimo del Execution Time (s)	71,16
Desviación estándar del CPU Usage (%)	1,45220
Desviación estándar del Memory Usage (%)	3,23305
Desviación estándar del Execution Time (s)	35,8469

Finalmente, en el bloque principal, se inicia la ejecución del motor del algoritmo genético durante 200 generaciones. Este bloque es el punto de partida para la optimización de la función binaria mediante el algoritmo genético.

Los resultados obtenidos con la ejecución de esta implementación se muestran en la Tabla III. En esta Tabla III se puede apreciar que el tiempo promedio obtenido es de 110.46 segundos. Lo que equivale a 1 minuto con 50 segundos; siendo de momento el más lento en la ejecución.

D. *MEALPY*

Finalmente, se presenta el código realizado para la maximización del problema de One-Max pero utilizando el framework de MEALPY para implementar el algoritmo genético (GA). En el CODIGO IV, se presenta las líneas del código que junto con sus respectivos parámetros permiten la implementación del algoritmo de prueba.

```
CODIGO 4.
Implementación del algoritmo One-Max usando MEALPY

import numpy as np
from mealpy.evolutionary_based.GA import BaseGA

def fitness_function(solution):
    return np.sum(solution)

problem_dict1 = {
    "fit_func": fitness_function,
    "lb": np.zeros(100),
    "ub": np.ones(100),
    "minmax": "max",
}

epoch = 1000
pop_size = 300

pc = 0.5 # probabilidad de cruce
pm = 0.2 # probabilidad de mutacion
```

Así que el código comienza importando la biblioteca NumPy, comúnmente usada para operaciones numéricas y manipulación de matrices en Python. A continuación, se importa la clase `BaseGA` del módulo `Genetic Algorithm (GA)` de la biblioteca MEALPY, que constituye la base del algoritmo genético.

Se define una función de aptitud, `fitness_function`, encargada de calcular la suma de los elementos de una solución. Luego, se configura un diccionario (`problem_dict1`) que contiene información esencial sobre el problema de optimización, incluyendo la función de aptitud a maximizar, límites inferiores y superiores, y la naturaleza del problema (maximización).

Se especifican parámetros cruciales, como el número de épocas, el tamaño de la población, y las probabilidades de cruce y mutación. Posteriormente, se crea una instancia del objeto `BaseGA` con estos parámetros, junto con el tipo de operadores de cruce y mutación a utilizar.

Se ejecuta el algoritmo genético utilizando la función `solve` con el diccionario de problemas definido previamente. Finalmente, se redondean los elementos de la mejor posición a 0 o 1 asumiendo que es un problema binario, y se imprime la solución y su respectiva aptitud.

Cabe mencionar que esta es la implementación que ocupa menor cantidad de líneas de código. Comparado con los otros tres frameworks.

Tabla IV
MÉTRICAS OBTENIDAS PARA EL FRAMEWORK MEALPY

Métrica	**Valor**
Media del CPU Usage (%)	4,9
Máximo del CPU Usage (%)	8,8
Mínimo del CPU Usage (%)	3,5
Media del Memory Usage (%)	60,4066
Máximo del Memory Usage (%)	62,5
Mínimo del Memory Usage (%)	55,1
Media del Execution Time (s)	66,65942
Máximo del Execution Time (s)	83,5631
Mínimo del Execution Time (s)	63,5273
Desviación estándar del CPU Usage (%)	1,374045
Desviación estándar del Memory Usage (%)	2,703199
Desviación estándar del Execution Time (s)	4,6663773

En esta implementación el tiempo promedio obtenido para las diferentes ejecuciones fue de 66.659 segundos, lo que equivale a 1 minuto y 6 segundos. Estos resultados se pueden observar en la Tabla IV.

Los resultados mostrados en las Tablas I,II, III, y IV se obtuvieron mediante el uso de la librería Psutil [16](*process and system utilities*). Psutil es una biblioteca de Python que ofrece amplias funcionalidades para acceder y gestionar información relacionada con el sistema. Permite a los desarrolladores acceder a datos detallados sobre el estado del sistema, incluido el uso de la CPU, la memoria, la utilización del disco, la actividad de la red, los sensores de temperatura y la alimentación del sistema. Entre sus funcionalidades tenemos la de gestionar los procesos en ejecución, así como la monitorización del uso del CPU y la memoria para analizar la carga de trabajo del sistema.

IV. DISCUSIÓN

En esta sección se realizará una discusión acerca de los resultados obtenido al evaluar los cuatro frameworks para cómputo evolutivo. En este caso se presentaran las ventajas y desventajas, de acuerdo al análisis realizado tanto de manera cuantitativa como cualitativa.

A. *Análisis cuantitativo*

Comenzaremos esta discusión a partir de la información que se obtiene gracias a la libreria de Psutil, que como ya se había comentado en párrafos anteriores se usó para generar las Tablas I-IV. Si bien es cierto que ya se consideró el desempeño respecto al tiempo de ejecución del algoritmo de One-Max implementado en cada librería, falto comentar lo referente al porcentaje de uso tanto del CPU como de la memoria.

En la Tabla V se presentan los valores promedio de los parámetros considerados para la evaluación de los algoritmos, los cuales son: 1) Uso del CPU, 2) Uso de la memoria, y 3)

tiempo de ejecución. Como se puede apreciar se marcaron con color amarillo los mejores valores obtenidos de cada parámetro para con respecto a las librerías que se están evaluando. Con ello es fácil identificar que el framework que necesito menos tiempo de ejecución fue, sin lugar a dudas, DEAP. De igual manera, DEAP fue el ganadoe en el porcentaje de uso de memoria, aunque en este caso la diferencia con los requerimientos de memoria de los otros frameworks no fue tan evidente. Lo anterior debido a que la implementaciones con PyGAD y con GAFT solo hubo un 8 % de diferencia y con MEALPY un 9 %. Finalmente, en el porcentaje de uso de CPU PyGAD fue la librería que obtuvo el mejor desempeño aunque DEAP la siguio muy de cercar con solo 0.24 % de diferencia.

De lo comentando en el párrafo anterior, podemos deducir que el framework con un desempeño destacado fue DEAP dado que obtuvo mejores valores en dos de los tres parámetros evaluados y a que en el restante parámetro el desempeño fue ligeramente menor al obtenido por PyGAD.

Tabla V
COMPARACIÓN DE DESEMPEÑO DE FRAMEWORKS EVALUADOS

Framework	CPU Usage (%)	Memory Usage (%)	Execution Time (s)
DEAP	3.79	51.29	0.49
PyGAD	3.55	59.57	17.42
GAFT	5.07	59.79	110.47
MEALPY	4.9	60.41	66.66

Otras métricas que pueden ser de utilidad para decidir cual librería usar son el número de líneas utilizado para la codificación del algoritmo. Así como el soporte o las actualizaciones que han tenido cada uno de los frameworks analizados.

En la Tabla VI se puede observar que la implementación del algoritmo de One-Max que necesitó de menos líneas para su codificación fue la implementación con MEALPY, usando solamente 18 líneas de código. En la misma Tabla VI observamos que la librería más actual es la de PyGAD. Sin embargo, las librerías de MEALPY y DEAP tiene solo uno y dos mes de diferencia, respectivamente, respecto a la última actualización de PyGAD. Por lo que podemos decir que estas tres librerías si ofrecen un soporte al usuario tanto en actualización como en documentación. Por lo comentado, en este caso el mejor framework sería MEALPY.

Tabla VI
ESPECIFICACIONES DE CÓDIGO Y VERSIONES

Framework	Líneas de Código	Primera Versión	Última Versión
DEAP	62	0.6 (7/10/2010)	1.4.1 (21/07/2023)
PyGAD	42	1.0.15 (16/04/2020)	3.2.0 (7/09/2023)
GAFT	38	0.1.1 (21/07/2017)	0.5.7 (30/11/2018)
MEALPY	18	0.1.0 (16/03/2020)	2.5.4 (7/08/2023)

B. Análisis cualitativo

Para el análisis cualitativo se mencionaran las principales ventajas y desventajas que se encontraron al realizar cada una de las implementación con los diferentes frameworks.

DEAP

Ventajas:

- Versatilidad: DEAP es versátil y ofrece una amplia gama de algoritmos evolutivos, programación genética y algoritmos genéticos.
- Comunidad activa: DEAP tiene una comunidad activa y solidaria, lo que conduce a actualizaciones continuas, corrección de errores y una variedad de algoritmos y extensiones contribuidas.
- Eficiencia: Está optimizado para el rendimiento y puede manejar problemas de optimización a gran escala.
- Desarrollo activo: DEAP se desarrolla activamente, con actualizaciones y mejoras que se publican periódicamente.

Desventajas:

- Curva de aprendizaje: El marco puede tener una curva de aprendizaje más pronunciada, especialmente para personas nuevas en algoritmos evolutivos y programación genética.
- Desafíos en la documentación: Aunque DEAP tiene documentación, algunos usuarios pueden encontrar que le falta orientación práctica.

PyGAD

Ventajas:

- Facilidad de uso: PyGAD está diseñado para ser fácil de usar, lo que lo hace accesible tanto para principiantes como para profesionales experimentados.
- Flexibilidad y personalización: El marco proporciona flexibilidad y permite la personalización de los componentes del algoritmo genético para adaptarse a requisitos de problemas específicos.
- Ligereza: PyGAD es ligero y eficiente, lo que lo hace adecuado para varias aplicaciones.
- Desarrollo activo: PyGAD se desarrolla activamente, con actualizaciones y mejoras que se publican periódicamente.

Desventajas:

- Variedad de algoritmos: La gama de algoritmos de optimización disponibles puede ser más limitada en comparación con algunos otros marcos.
- Tamaño de la comunidad: La comunidad puede ser más pequeña, lo que podría resultar en una documentación menos extensa y menos recursos en línea.

GAFT

Ventajas:

- Simplicidad: GAFT está diseñado con simplicidad en mente, lo que lo hace relativamente fácil de usar y entender, especialmente para aquellos nuevos en algoritmos genéticos.

- Personalización: El marco permite una fácil personalización y modificación de los componentes del algoritmo genético para adaptarse a dominios de problemas específicos.

Desventajas:

- Tipos limitados de algoritmos: En comparación con marcos más extensos como DEAP, GAFT puede ofrecer un conjunto más limitado de algoritmos y operadores evolutivos.
- Poca documentación: La comunidad puede ser más pequeña, lo que podría resultar en una documentación potencialmente menos completa y menos recursos en línea.

MEALPY

Ventajas:

- Facilidad de uso: MEALPY es fácil de usar y comprender, tanto para principiantes como para investigadores experimentados.
- Modularidad: Ofrece modularidad, lo que permite a los usuarios integrar y experimentar con diversos algoritmos de optimización metaheurística.
- Documentación: Dispone de una buena documentación y ejemplos, lo que ayuda a los usuarios a entender y utilizar el marco de manera efectiva.

Desventajas:

- Tamaño de la comunidad: La comunidad puede ser más pequeña en comparación con la existente para otras librerías, lo que podría resultar en menos recursos en línea y apoyo de la comunidad.

En este caso no es tan evidente el determinar cual de los frameworks es mejor ya que depende de las necesidades del usuario.

V. Conclusiones

En este trabajo se presentó una evaluación de cuatro frameworks para implementar algoritmos evolutivos, principalmente el de algoritmos genéticos. De la evaluación realizada podemos comentar lo siguiente:

- DEAP tiene un uso moderado de CPU y memoria, con tiempos de ejecución muy bajos en comparación con las otras bibliotecas.
- PyGAD muestra un rendimiento moderado en términos de CPU y memoria, pero su tiempo de ejecución es significativamente menor en comparación con MEALPY y GAFT.
- GAFT presenta altos tiempos de ejecución, pero también un uso moderado de CPU y memoria.
- MEALPY parece tener un mayor uso de CPU y memoria en comparación con las otras bibliotecas, pero también tiene tiempos de ejecución más bajos en comparación con GAFT.

En resumen, la elección del framework a utilizar depende de diversos factores como son: a) el nivel de experiencia del usuario, b) los algoritmos de optimización específicos necesarios, c) la facilidad de uso, d) la documentación existente, y e) el nivel de apoyo de la comunidad o ejemplos existentes que puede ser útiles para la implementación del proyecto a desarrollar.

Referencias

[1] J. H. Holland, *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*, ser. Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence. Oxford, England: U Michigan Press, 1975.

[2] H. Zhi and S. Liu, "Face recognition based on genetic algorithm," *Journal of Visual Communication and Image Representation*, vol. 58, p. 495–502, Jan. 2019.

[3] N. Maleki, Y. Zeinali, and S. T. A. Niaki, "A k-nn method for lung cancer prognosis with the use of a genetic algorithm for feature selection," *Expert Systems with Applications*, vol. 164, p. 113981, Feb. 2021.

[4] M. Kordos, J. Boryczko, M. Blachnik, and S. Golak, "Optimization of warehouse operations with genetic algorithms," *Applied Sciences*, vol. 10, no. 1414, p. 4817, Jan. 2020.

[5] I. Benchaji, S. Douzi, and B. El Ouahidi, "Using genetic algorithm to improve classification of imbalanced datasets for credit card fraud detection," in *Smart Data and Computational Intelligence*, ser. Lecture Notes in Networks and Systems, F. Khoukhi, M. Bahaj, and M. Ezziyyani, Eds. Cham: Springer International Publishing, 2019, p. 220–229.

[6] Q. Ji, Z. Qian, L. Ren, and L. Ren, "Torque curve optimization of ankle push-off in walking bipedal robots using genetic algorithm," *Sensors*, vol. 21, no. 1010, p. 3435, Jan. 2021.

[7] G. Capi, Y. Nasu, L. Barolli, K. Mitobe, and K. Takeda, "Application of genetic algorithms for biped robot gait synthesis optimization during walking and going up-stairs," *Advanced Robotics*, vol. 15, no. 6, p. 675–694, Jan. 2001.

[8] C. Lamini, S. Benhlima, and A. Elbekri, "Genetic algorithm based approach for autonomous mobile robot path planning," *Procedia Computer Science*, vol. 127, p. 180–189, Jan. 2018.

[9] X. Liu, D. Jiang, B. Tao, G. Jiang, Y. Sun, J. Kong, X. Tong, G. Zhao, and B. Chen, "Genetic algorithm-based trajectory optimization for digital twin robots," *Frontiers in Bioengineering and Biotechnology*, vol. 9, 2022. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fbioe.2021.793782

[10] J. Liu, Z. Chen, Y. Zhang, and W. Li, "Path planning of mobile robots based on improved genetic algorithm," in *Proceedings of the 2020 2nd International Conference on Robotics, Intelligent Control and Artificial Intelligence*, ser. RICAI '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 49–53. [Online]. Available: https://doi.org/10.1145/3438872.3439054

[11] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Advances in Engineering Software*, vol. 69, p. 46–61, Mar. 2014.

[12] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, jul 2012.

[13] A. F. Gad, "Pygad: An intuitive genetic algorithm python library," 2021.

[14] Z. Shao, "Gaft documentation."

[15] N. Van Thieu and S. Mirjalili, "Mealpy: An open-source library for latest meta-heuristic algorithms in python," *Journal of Systems Architecture*, 2023.

[16] G. Rodola, "Psutil: Process and system utilities library," Oct. 2023. [Online]. Available: https://github.com/giampaolo/psutil