Tarea 4
Algoritmos genéticos

Mario Emilio Jiménez Vizcaíno
A01173359@itesm.mx

Jesus Abraham Haros Madrid
A01252642@itesm.mx

1 INTRODUCCIÓN

Durante las últimas décadas, una de las grandes inspiraciones para los avances en la inteligencia artificial ha sido la naturaleza, y los algoritmos genéticos no son la excepción. Este tipo de algoritmos se basan en 3 operaciones simples sobre vectores de bits: la cruza, la mutación y la selección, para así poder imitar los mecanismos de la selección natural y la reproducción.[2]

En esta práctica utilizamos 3 algoritmos genéticos para resolver el problema de la mochila, mejor conocido como *Knapsack problem*, en el que se tiene una lista de ítems con pesos y valores, y el objetivo es seleccionar de la lista una combinación de ítems cuya suma de valores sea máxima, mientras la suma de sus pesos sea inferior a un límite predefinido.

2 METODOLOGÍA E IMPLEMENTACIÓN

Para la realizar la implementación de este problema utilizamos la librería *deap* de Python, la cual implementa los 3 algoritmos genéticos presentados.[1] La implementación del algoritmo puede ser encontrada el apéndice A.

2.1 Representación de los individuos

La generación y administración de los individuos es una tarea que la librería maneja por nosotros, aunque si especificamos cómo generar los atributos de los individuos (cada bit del vector, utilizando números aleatorios para generar un bit), cómo generar cada individuo (repitiendo la generación del atributo n veces, siendo n el número de ítems de los cuales podemos escoger), y cómo generar a la población (repitiendo la generación del individuo, en nuestro caso 10 veces).

Concretamente, cada uno de los individuos de la población está formado de 24 bits, inicialmente aleatorios.

2.2 Función de evaluación

La función de evaluación es muy simple, y está formada por dos pasos:

(1) El cálculo de la suma de los valores de los ítems seleccionados, primero multiplicando los bits del individuo (0/1) por la lista de valores de los ítems, y finalmente sumando los valores. Esta suma determina qué tan eficiente es el individuo, y el objetivo del algoritmo es maximizarla.
(2) El cálculo de la suma de los pesos de los ítems seleccionados, que es usado para que, en caso de que exceda el límite predefinido, la evaluación del individuo sea penalizada. En nuestro caso preferimos solamente asignarle una evaluación de 0.

2.3 Algoritmos genéticos

Los tres algoritmos genéticos principales se explican a continuación.

2.3.1 EA simple. Este algoritmo utiliza una población de individuos, que son cruzados entre sí y sufren mutaciones aleatorias. Como consecuencia de que los individuos pueden sufrir modificaciones sin importar si tuvieron una buena evaluación o no, el algoritmo puede tener resultados muy variados.

2.3.2 EA $(\mu + \lambda)$. Este algoritmo y el siguiente tienen en común que sólo sufren una variación, ya sea cruce o mutación. En el caso de este algoritmo, la nueva generación se escoje de un conjunto de los hijos y su descendencia, por lo que si algún padre tiene una mejor evaluación que sus hijos, puede continuar produciendo hijos en las siguientes generaciones.

2.3.3 EA (μ, λ). Este algoritmo es muy parecido al anterior, pero difiere en que la siguiente generación de la población sólo se selecciona de la descendencia de los padres. Esto produce una mayor rapidez de "exploramiento" de posibles resultados, aunque perdiendo a los padres en cada generación.

3 RESULTADOS

3.1 Mejores instancias

Las mejores instancias que obtuvimos durante la ejecución de los algoritmos fueron las siguientes. También incluímos un porcentaje que describe qué tan cerca estuvieron de la solución óptima de 13,549,094.

3.1.1 EA simple.

(1) 13,452,724 (99.289%)
(2) 13,375,768 (98.721%)
(3) 13,354,550 (98.564%)

3.1.2 EA $(\mu + \lambda)$.

(1) 13,461,886 (99.357%)
(2) 13,433,220 (99.145%)
(3) 13,419,377 (99.043%)

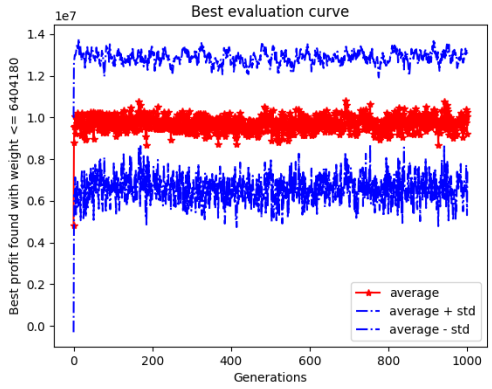
3.1.3 EA (μ, λ).

(1) 13,518,963 (99.778%)
(2) 13,396,655 (98.875%)
(3) 13,377,418 (98.733%)

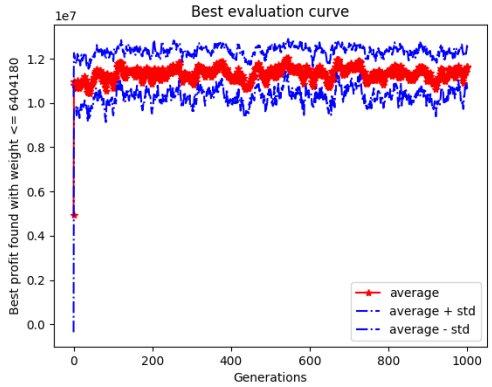
3.2 Curva de mejor encontrado

A continuación se muestran tres gráficas, una por cada algoritmo, de la mejor evaluación encontrada por cada generación, promediada entre las 10 ejecuciones de los algoritmos.

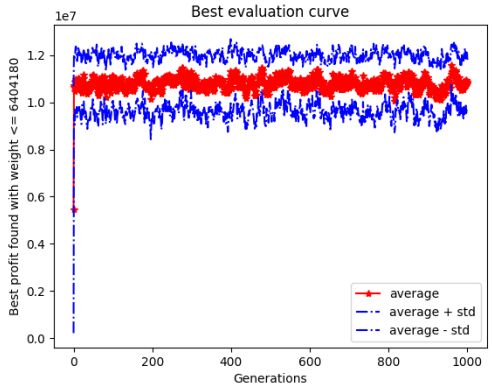
3.2.1 EA simple. Este algoritmo tuvo la mayor desviación estándar como consecuencia de que las variaciones se aplicaban a todos los individuos sin importar si tenían o no una buena evaluación.



3.2.2 EA $(\mu+\lambda)$. En la gráfica de este algoritmo se puede apreciar que los cambios y las mutaciones son mucho más suaves, ya que se le permitía a los padres formar parte de las siguientes iteraciones, y por lo tanto era más difícil hacer grandes cambios en la población general.



3.2.3 EA (μ,λ). Finalmente esta gráfica tiene un gran parecido con la gráfica anterior, pero se puede observar cambios más rápidos y "aleatorios" ya que la descendencia a través de las generaciones podía tener evaluaciones más variadas que en el algoritmo anterior.



4 CONCLUSIONES Y RETOS ENCONTRADOS

Los algoritmos genéticos nos presentan una forma diferente y útil de resolver problemas que pueden ser representados de manera simple, aunque si creemos que la selección de parámetros iniciales y la configuración de los algoritmos es una tarea que merece una investigación exclusiva.

Los principales retos que tuvimos fueron al comprender cómo funciona la librería *deap*, ya que fue de gran utilidad para describir y ejecutar los algoritmos genéticos a través de las funciones que ofrece la librería, pero también utilizando un par ejemplos proporcionados por el profesor como base.

REFERENCES

[1] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary Algorithms Made Easy. *Journal of Machine Learning Research* 13 (jul 2012), 2171–2175.

[2] John H Holland. 1992. Genetic algorithms. *Scientific american* 267, 1 (1992), 66–73.

A IMPLEMENTACIÓN DE LOS ALGORITMOS GENÉTICOS

```python
import sys

from deap import algorithms, base, creator, tools
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd


# Ensuring an algorithm was selected
possible_algs = ["simple", "plus", "comma"]
if len(sys.argv) < 2 or sys.argv[1] not in possible_algs:
    print("Add an argument from the list:", possible_algs)
    exit(1)

# Load inputs
p = pd.read_csv('./p08_p.txt', header=None, names=["p"])["p"]
w = pd.read_csv('./p08_w.txt', header=None, names=["w"])["w"]
wmax = 6404180

# Params for the genetic algorithm
cxpb = 0.7
mutpb = 0.3
ngen = 1000
mu = 10
lambda_ = 20

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

toolbox = base.Toolbox()
# Evolution params
toolbox.register("select", tools.selRoulette)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.1)
toolbox.register("mate", tools.cxOnePoint)
# Evaluation params
toolbox.register("evaluate", lambda i: (np.sum(i * p),))
toolbox.decorate("evaluate",
                 tools.DeltaPenalty(lambda i: np.sum(i * w) <= wmax, 0))
# Population and individual params
toolbox.register("attribute", np.random.randint, low=0, high=2)
toolbox.register("individual", tools.initRepeat, creator.Individual,
                 toolbox.attribute, n=24)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
```

```
# Stats for every execution
avgs_list = []
stds_list = []
root_hof = tools.HallOfFame(5)

# Execute the algorithm 10 times
for i in range(10):
    print(f"Execution {i + 1}")
    popul = toolbox.population(n=10)

    stats = tools.Statistics(key=lambda ind: ind.fitness.values)
    stats.register("avg", np.mean)
    stats.register("std", np.std)

    hof = tools.HallOfFame(5)

    logbook = None
    if sys.argv[1] == possible_algs[0]:
        _, logbook = algorithms.eaSimple(
            popul, toolbox, cxpb, mutpb, ngen, stats, hof, False)
    elif sys.argv[1] == possible_algs[1]:
        _, logbook = algorithms.eaMuPlusLambda(
            popul, toolbox, mu, lambda_, cxpb, mutpb, ngen, stats, hof, False)
    else:
        _, logbook = algorithms.eaMuCommaLambda(
            popul, toolbox, mu, lambda_, cxpb, mutpb, ngen, stats, hof, False)

    avgs_list.append(logbook.select("avg"))
    stds_list.append(logbook.select("std"))
    root_hof.update(hof.items)


# Print the individuals with the best evaluation
print("*** Hall of fame *** ")
for idx, indiv in enumerate(root_hof.items):
    print(f"{idx + 1}. {indiv} = {np.sum(indiv * p)}")


# Plot the generations through the evolutions
x = range(ngen + 1)
avg = np.average(np.array(avgs_list), axis=0)
std = np.sqrt(np.average(np.array(stds_list) ** 2, axis=0))
plt.title('Best evaluation curve')
plt.plot(x, avg, color='red', marker='*')
plt.plot(x, avg + std, color='b', linestyle='-.')
plt.plot(x, avg - std, color='b', linestyle='-.')
plt.xlabel('Generations')
plt.ylabel(f'Best profit found with weight ⩽ {wmax}')
plt.legend(['average', 'average + std', 'average - std'])
plt.savefig(f'{sys.argv[1]}.png')
```