

REPORTE TAREA #4

INTELIGENCIA

COMPUTACIONAL

línea horizontal

Jesús Rivera Flores - A00820643

Diego Arnoldo Azuela Rosas - A01208345



**CONTENIDO**

# 

[**INSTRUCCIONES**](#_1d3blirkvx7b) **3**

[HISTORIA](#_49k4nmgv0c6q) 3

[INSTRUCCIONES](#_16hk2purkd4v) 3

[INTERPRETACIÓN DEL PROBLEMA](#_ciwlzli16y57) 3

[**DESCRIPCIÓN DEL PROBLEMA**](#_tnnn92xp2u3i) **3**

[**DESCRIPCIÓN DE LA REPRESENTACIÓN**](#_w6g374e1aq4b) **3**

[DESCRIBIR EL FORMATO DE LA SOLUCIÓN](#_wjw3hbkbzoul) 3

[**DESCRIPCIÓN DE LA FUNCIÓN DE EVALUACIÓN**](#_9rlpbme1ta6k) **3**

[**DESCRIBIR DIFERENCIAS ENTRE LOS 3 ALGORITMOS**](#_9gjz3off516) **3**

[**RESULTADOS OBTENIDOS**](#_c0l1pninng4b) **3**

[**GRÁFICA CURVA MEJOR ENCONTRADO**](#_ip84yoj93dc3) **3**

[**CONCLUSIÓN Y RETOS ENCONTRADOS**](#_6lgapy8pm413) **3**

[**4. BIBLIOGRAFÍAS**](#_ytltuoh9d04) **5**

# 

# 

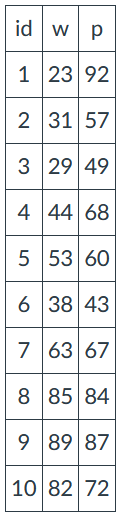
# INSTRUCCIONES

## OBJETIVO

* Utilizar tres variantes de algoritmos genéticos y comparar los resultados de las 3.

## INSTRUCCIONES

1. Resolver el problema de la mochila con los siguientes datos. El peso máximo es 165 kg. Los parámetros para el algoritmo quedan a discreción del alumno. Realizar la penalización fuera de la función de evaluación (usar decorator para penalización).



1. Correr al menos 10 veces cada uno de los 3 algoritmos durante al menos 50 iteraciones y almacenar los resultados.
   1. eaSimple
   2. eaMuPlusLambda
   3. eaMuCommaLambda
2. Desarrollar el reporte de la actividad.

# DESCRIPCIÓN DEL PROBLEMA

* El problema de la mochila se puede interpretar de la siguiente manera:
  + Se tiene una mochila en donde se busca meter el máximo número de elementos (‘p’) dentro sin que su peso (‘w’) sobrepase un límite previamente especificado. Se puede plantear el problema como el querer maximizar el volumen dentro de la mochila, minimizando el peso obtenido. Asegurándose que el peso se encuentre por debajo de 165 kg (‘W\_max’) en todo momento.

# DESCRIPCIÓN DE LA REPRESENTACIÓN

## DESCRIBIR EL FORMATO DE LA SOLUCIÓN

* El IDE utilizado es Colab de Google. Formato similar al presentado por Jupyter Notebooks, pero con la versatilidad de Google que permite acceder a la información de diferentes puntos y con la seguridad de que todo se ha guardado en la nube. Se presentará el código junto con una foto de la gráfica que produce el código.

## 

| **DESARROLLO DEL CÓDIGO** | |
| --- | --- |
| **INICIALIZACIÓN DE DATOS E IMPORTACIÓN DE LIBRERÍAS** | pip install deap  from deap import base, creator  from deap import algorithms  from deap import tools  import numpy as np  import random  import matplotlib.pyplot as plt  import pandas as pd  w = [23, 31, 29, 44, 53, 38, 63, 85, 89, 82]  p = [92, 57, 49, 68, 60, 43, 67, 84, 87, 72]  w\_max = 165 |
| **FUNCIONES** | def func\_eval(u):  """  En esta función se evalúa  """  fu = np.sum(np.asarray(u) \* np.asarray(p))  return fu, |
| def func\_evalv(u):  """  En esta funcion se calcula la evaluacion    :param u:  :return:  """  fu = np.sum(np.asarray(u) \* np.asarray(p))  return fu |
| def peso(u):  """  En esta función se calcula que el peso del individuo  se encuentre dentro de los valores permitidos  :param u:  :return:  """  wi = np.sum(np.asarray(u) \* np.asarray(w))  if (wi < w\_max):  return True  return False |
| def convertir\_datos(log):  # Convertir a DataFrame los datos obtenidos  df\_log = pd.DataFrame(log[1])  df\_log = df\_log.apply(pd.Series)  # Obtener los valores 'max' para graficarlos  max\_element = 0  for i in range(len(df\_log)):  if (df\_log['max'][i] > max\_element):  max\_element = df\_log['max'][i]  elif (df\_log['max'][i] < max\_element):  df\_log['max'][i] = max\_element  return df\_log |
| def graficar\_datos(log):  # Graficar los valores máximos  df\_log = convertir\_datos(log)  plt.plot(df\_log['gen'], df\_log['max'])  plt.xlabel('Generaciones')  plt.ylabel('Valor máximo obtenido')  plt.title('eaSimple Mayor Valor Encontrado')  plt.show() |
| **PARÁMETROS** | # Definir si es un problema de maximizar o minimizar.  creator.create("FitnessMin", base.Fitness, weights=(1.0,))  # Definir que los individuos son listas y que se va a maximizar.  creator.create("Individual", list, fitness=creator.FitnessMin) |
| toolbox = base.Toolbox()  # Seleccionar la función de selección.  toolbox.register("select", tools.selRoulette)  # Seleccionar la función de mutación.  toolbox.register("mutate", tools.mutFlipBit, indpb=0.1)  # Seleccionar el de reproducción.  toolbox.register("mate", tools.cxOnePoint) |
| # Definir la función de evaluación.  toolbox.register("evaluate", func\_eval)  # Definir la penalización por el peso  toolbox.decorate("evaluate", tools.DeltaPenalty(peso,1)) #Se puede agregar la función de distancia |
| # Definir un elemento del individuo.  toolbox.register("attribute", random.randint, a=0, b=1)  # Definiendo la creación de individuos como una lista de n elementos.  toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attribute, n=10)  # Definiendo la creación de la población.  toolbox.register("population", tools.initRepeat, list, toolbox.individual) |
| **ESTADÍSTICAS** | pop = toolbox.population(n=10)  stats = tools.Statistics(key=lambda ind: ind.fitness.values)  stats.register("max", np.max)  stats.register("min", np.min)  stats.register("avg", np.mean)  stats.register("std", np.std) |
| **ESPECIFICACIÓN DE GENERACIONES** | hof = tools.HallOfFame(3)  #Número de individuos a seleccionar para la próxima generación  mu = 2  #Número de hijos que se producen el cada generación  lamb = 8 |
| **IMPLEMENTACIÓN** | simple = []  plus = []  comma = []    for experiment in range(10):  #Algoritmos con 50 generaciones para que sean 50 iteraciones  hof1 = tools.HallOfFame(1)  hof2 = tools.HallOfFame(1)  hof3 = tools.HallOfFame(1)  for iteracion in range(50):  #Algoritmo eaSimple  log1 = algorithms.eaSimple(population=pop, toolbox=toolbox, halloffame=hof1, cxpb=1.0, mutpb=1.0,  ngen=10, stats=stats, verbose=False)  val = func\_evalv(hof1[0])  simple.append([experiment, iteracion, val])    #Algoritmo eaMuPlusLambda  log2 = algorithms.eaMuPlusLambda(population=pop, toolbox=toolbox, halloffame=hof2, mu=mu, lambda\_=lamb, cxpb=0.5, mutpb=0.5,  ngen=10, stats=stats, verbose=False)  plus.append([experiment, iteracion, func\_evalv(hof2[0])])    #Algoritmo eaMuCommaLambda  log3 = algorithms.eaMuCommaLambda(population=pop, toolbox=toolbox, halloffame=hof3, mu=mu, lambda\_=lamb, cxpb=0.5, mutpb=0.5,  ngen=10, stats=stats, verbose=False)  comma.append([experiment, iteracion, func\_evalv(hof3[0])])  hof1.clear()  hof2.clear()  hof3.clear() |
| **GRAFICAR** | simpleDf = pd.DataFrame(simple, columns = ['Experiment', 'Iteration', 'Value'])  commaDf = pd.DataFrame(comma, columns = ['Experiment', 'Iteration', 'Value'])  plusDf = pd.DataFrame(plus, columns = ['Experiment', 'Iteration', 'Value'])    simpledf = commadf = plusdf = []  for x in range(10):  simpledf.append(pd.DataFrame(simpleDf[(simpleDf['Experiment'] == x)]))  plusdf.append(pd.DataFrame(plusDf[(plusDf['Experiment'] == x)]))  commadf.append(pd.DataFrame(commaDf[(commaDf['Experiment'] == x)]))    plt.subplot(1,3,1)  plt.xlabel('Iteraciones')  plt.ylabel('Valor máximo obtenido')  plt.title('eaSimple')  for x in range(10):  plt.plot(simpledf[x]['Iteration'], simpledf[x]['Value'], label = "Experiment"+str(x))    plt.subplot(1,3,2)  plt.xlabel('Iteraciones')  plt.ylabel('Valor máximo obtenido')  plt.title('eaMuPlusLambda')  for x in range(10):  plt.plot(plusdf[x]['Iteration'], plusdf[x]['Value'], label = "Experiment"+str(x))    plt.subplot(1,3,3)  plt.xlabel('Iteraciones')  plt.ylabel('Valor máximo obtenido')  plt.title('eaMuCommaLambda')  for x in range(10):  plt.plot(commadf[x]['Iteration'], commadf[x]['Value'], label = "Experiment"+str(x))      plt.legend()  plt.show() |
| **RESULTADOS** |  |

## 

## DESCRIBIR CÓDIGO

| **CÓDIGO** | |
| --- | --- |
| **INICIALIZACIÓN DE DATOS E IMPORTACIÓN DE LIBRERÍAS** | Se tiene que tener una importación de las librerías básicas a implementar. Aquellas con las que no se cuente, se instalarán para asegurar el correcto funcionamiento. |
| **FUNCIONES** | Básicamente se tienen 4 funciones:   * Para evaluar la función * Para calcular el peso * Para reemplazar el valor ‘max’ en log * Para graficar |
| **PARÁMETROS** | Se establecen los parámetros necesarios utilizando la función toolbox. |
| **ESTADÍSTICAS** | Se establecen las estadísticas necesarias utilizando la función toolbox. |
| **ESPECIFICACIÓN DE GENERACIONES** | Se especifican el número de generaciones y de hijos que se tendrán por generación. |
| **IMPLEMENTACIÓN** | Se juntan las funciones y parámetros anteriores para lograr un desarrollo coherente del algoritmo. |
| **GRAFICAR** | Se grafica la solución obtenida. |
| **RESULTADOS** |  |

# 

# DESCRIPCIÓN DE LA FUNCIÓN DE EVALUACIÓN

| **FUNCIÓN** | |
| --- | --- |
| **FUNCIÓN DE EVALUACIÓN** | def func\_evalv(u):  """  En esta funcion se calcula la evaluacion    :param u:  :return:  """  fu = np.sum(np.asarray(u) \* np.asarray(p))  return fu |
| **DESCRIPCIÓN** | Se puede observar que en esta porción del código simplemente se toman los valores de ‘u’ y ‘p’ como ‘arrays’ y se van a sumar. La parte de evaluación y de determinar si son mayores al valor máximo permitido vienen después. |

# 

# DESCRIBIR DIFERENCIAS ENTRE LOS 3 ALGORITMOS

| **ALGORITMOS** | |
| --- | --- |
| **eaSimple** |  |
| **eaMuPlusLambda** |  |
| **eaMuCommaLambda** |  |

Las diferencias entre los diferentes algoritmos se vuelven aparentes a partir de que se tiene una comparación de los primeros 2 algoritmos ‘eaSimple’ y ‘eaMuPlusLambda’. El primer algoritmo cuenta con 8 parámetros, los cuales son incluidos en el segundo algoritmo. Sin embargo el segundo algoritmo incluye de igual manera dos aspectos clave extra: ‘mu’ y ‘lambda\_’. El primero de estos parámetros especifica el número de individuos que serán incluidos en la siguiente generación. Por otro lado, el segundo parámetro le indica al algoritmo el número de ‘hijos’ que se debe de producir por generación.

De la misma manera, el algoritmo restante puede ser comparado con el segundo algoritmo de la siguiente manera: el tercer algoritmo crea las mutaciones o los traslapes a partir de un cierto número de individuos para determinar un número determinado de hijos. Mientras que el algoritmo número dos toma la siguiente generación a partir de los hijos y los padres.

# RESULTADOS OBTENIDOS

Se obtuvieron 3 gráficas, en cada una de las gráficas se puede observar un algoritmo diferente. Estas gráficas tienen diferente número de colores, cada uno de ellos demuestra una iteración diferente. El resultado obtenido fue el esperado. Gracias a la función de ‘convertir\_datos’ se toman los valores máximos de ‘log’, y se produce una gráfica a partir de esos datos. De esta manera se espera que sea una gráfica creciente con un máximo determinado por el valor del peso máximo. Las gráficas presentadas en la siguiente sección se comportan de la manera esperada.

# GRÁFICA CURVA MEJOR ENCONTRADO

## 

| **GRAFICAR** | |
| --- | --- |
| **RESULTADOS** |  |

## 

# CONCLUSIÓN Y RETOS ENCONTRADOS

La creación de esta librería simplifica en gran medida el proceso necesario para implementar algoritmos genéticos en problemas de optimización. La gran manera en que se simplifica el proceso implica que se sabe de qué manera se tiene que crear las diferentes generaciones y los traslapes. Por lo mismo, el usuario solamente necesita ingresar un valor fijo en el lugar adecuado para lograr un algoritmo funcional y de uso inmediato. Por otro lado, existen ciertas complicaciones como sea, se tuvo problemas implementando la función de ‘decorate’. Esta función tiene como propósito principal el ‘castigar’ a la función de cierta manera en caso de que se sobrepase el valor máximo permitido (W\_max).

La realización de este código fue interesante y probó ser un reto.Es un algoritmo robusto que maneja tanto mejores soluciones locales como mejores soluciones globales. El trabajo del equipo fue arduo e intenso. Al final se consolidó un código que sobrepasó las pruebas que se le hicieron para comprobar, sin embargo se seguirá trabajando para depurar cualquier imperfecto restante.

# 

# 

# BIBLIOGRAFÍAS

* https://deap.readthedocs.io/en/master/tutorials/advanced/constraints.html
* https://github.com/DEAP/deap/blob/master/deap/algorithms.py