



# Universidad Autónoma del Estado de México

## Ingeniería en Computación

Materia: Algoritmos Genéticos  
Proyecto 2

Axel Valenzuela Juárez  
Profesor: Dr. Asdrúbal López Chau  
31 de Marzo del 2020

# 1. Introducción

Los Algoritmos Genéticos son métodos adaptativos que pueden usarse para resolver problemas de búsqueda y optimización. Están basados en el proceso genético de los organismos vivos.

Los Algoritmos Genéticos son capaces de ir creando soluciones para problemas del mundo real.

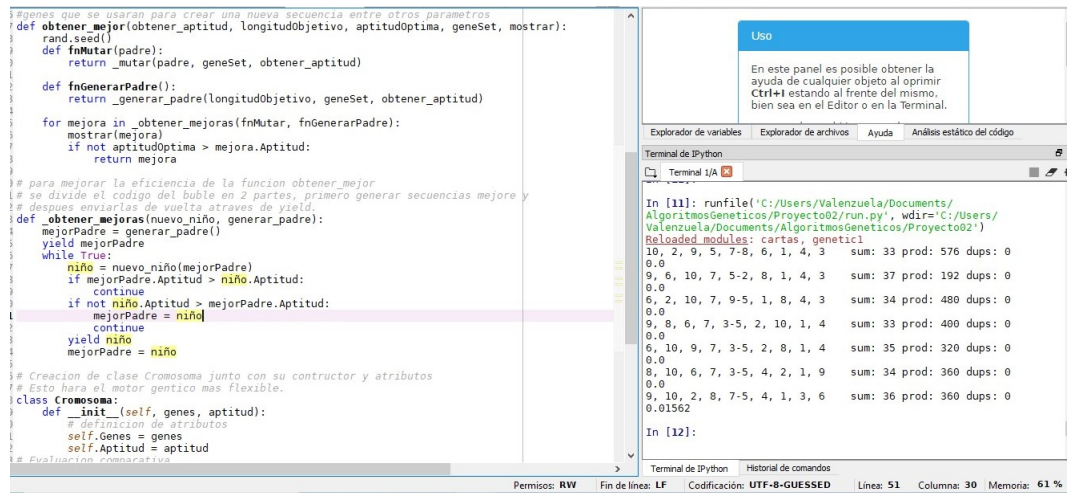
El poder de los Algoritmos Genéticos proviene del hecho de que se trata de una técnica robusta, y pueden tratar con éxito una gran variedad de problemas provenientes de diferentes áreas, incluyendo aquellos en los que otros métodos encuentran dificultades. Si bien no se garantiza que el Algoritmo Genético encuentre la solución óptima del problema, existe evidencia empírica de que se encuentran soluciones de un nivel aceptable, en un tiempo competitivo con el resto de los algoritmos de optimización combinatoria. En el caso de que existan técnicas especializadas para resolver un determinado problema, lo más probable es que superen al Algoritmo Genético, tanto en rapidez como en eficacia.

El gran campo de aplicación de los Algoritmos Genéticos se relaciona con aquellos problemas para los cuales no existen técnicas especializadas. Incluso en el caso en que dichas técnicas existan, y funcionen bien, pueden efectuarse mejoras de estas hibridándolas con los Algoritmos Genéticos.

## 2. Desarrollo

El problema de las cartas consiste en un conjunto de cartas del 2 al 10 y el as que se separan en dos grupos de 5 cartas, las cartas de cada grupo tendr n diferentes restricciones.

En el primer grupo las cartas deben tener un producto de 360 y en el otro grupo los valores deben sumar 36. Solo se pueden usar una vez las cartas.



```
#genes que se usaran para crear una nueva secuencia entre otros parametros
def obtener_mejor(obtener_apitud, longitudObjetivo, aptitudOptima, geneSet, mostrar):
    rand.seed()
    def fnMutar(padre):
        return _mutar(padre, geneSet, obtener_apitud)

    def fnGenerarPadre():
        return _generar_padre(longitudObjetivo, geneSet, obtener_apitud)

    for mejora in _obtener_mejoras(fnMutar, fnGenerarPadre):
        mostrar(mejora)
        if not aptitudOptima > mejora.Aptitud:
            return mejora

# para mejorar la eficiencia de la funcion obtener_mejor
# se divide el codigo del bubble en 2 partes, primero generar secuencias mejore y
# despues enviarlas de vuelta atraves de yield.
def _obtener_mejoras(nuevo_niño, generar_padre):
    mejorPadre = generar_padre()
    yield mejorPadre
    while True:
        niño = nuevo_niño(mejorPadre)
        if mejorPadre.Aptitud > niño.Aptitud:
            continue
        if not niño.Aptitud > mejorPadre.Aptitud:
            mejorPadre = niño
        continue
    yield niño
    mejorPadre = niño

# Creacion de clase Cromosoma junto con su constructor y atributos
# Esto hara el motor genético mas flexible.
class Cromosoma:
    def __init__(self, genes, aptitud):
        # definicion de atributos
        self.Genes = genes
        self.Aptitud = aptitud

# Evaluacion comparativa
```

Uso

En este panel es posible obtener la ayuda de cualquier objeto al oprimir **Ctrl+I** estando al frente del mismo, bien sea en el Editor o en la Terminal.

Explorador de variables Explorador de archivos Ayuda Análisis estático del código

Terminal de IPython

```
In [11]: runfile('C:/Users/Valenzuela/Documents/
AlgoritmosGeneticos/Proyecto02/run.py', wdir='C:/Users/
Valenzuela/Documents/AlgoritmosGeneticos/Proyecto02')
Reloaded modules: cartas, genetic1
10, 2, 9, 5, 7-8, 6, 1, 4, 3 sum: 33 prod: 576 dups: 0
0.0
9, 6, 10, 7, 5-2, 8, 1, 4, 3 sum: 37 prod: 192 dups: 0
0.0
6, 2, 10, 7, 9-5, 1, 8, 4, 3 sum: 34 prod: 480 dups: 0
0.0
9, 8, 6, 7, 3-5, 2, 10, 1, 4 sum: 33 prod: 400 dups: 0
0.0
6, 10, 9, 7, 3-5, 2, 8, 1, 4 sum: 35 prod: 320 dups: 0
0.0
8, 10, 6, 7, 3-5, 4, 2, 1, 9 sum: 34 prod: 360 dups: 0
0.0
9, 10, 2, 8, 7-5, 4, 1, 3, 6 sum: 36 prod: 360 dups: 0
0.01562

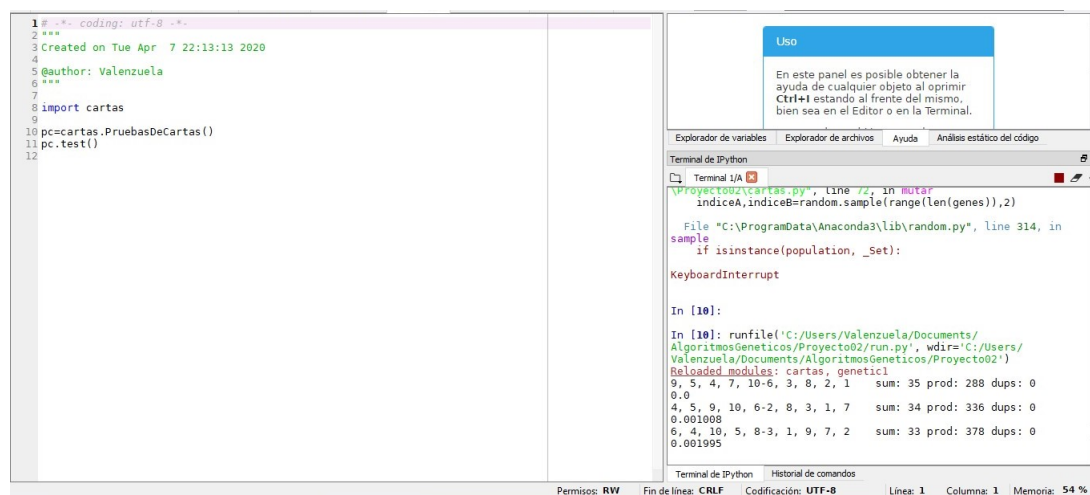
In [12]:
```

Terminal de IPython Historial de comandos

Permisos: RW Fin de línea: LF Codificación: UTF-8-GUESSED Línea: 51 Columna: 30 Memoria: 61 %

Figura 1: Cambios en el archivo genetic.

La primer parte del algoritmo funciona correctamente pero el algoritmo se traba ya que el algoritmo tiene que cambiar dos números, eso es lo que a continuación se busca hacer.



```
1 # -*- coding: utf-8 -*-
2 """
3 Created on Tue Apr 7 22:13:13 2020
4
5 @author: Valenzuela
6 """
7
8 import cartas
9
10 pcc=cartas.PruebasDeCartas()
11 pc.test()
12
```

Uso

En este panel es posible obtener la ayuda de cualquier objeto al oprimir **Ctrl+I** estando al frente del mismo, bien sea en el Editor o en la Terminal.

Explorador de variables Explorador de archivos Ayuda Análisis estático del código

Terminal de IPython

```
Terminal 1/A
V:\Proyecto02\cartas.py", line 72, in mutar
    indiceA, indiceB=random.sample(range(len(genes)),2)
File "C:\ProgramData\Anaconda3\lib\random.py", line 314, in
sample
    if isinstance(population, _Set):
KeyboardInterrupt

In [10]:

In [10]: runfile('C:/Users/Valenzuela/Documents/
AlgoritmosGeneticos/Proyecto02/run.py', wdir='C:/Users/
Valenzuela/Documents/AlgoritmosGeneticos/Proyecto02')
Reloaded modules: cartas, genetic1
9, 5, 4, 7, 10-6, 3, 8, 2, 1 sum: 35 prod: 288 dups: 0
0.0
4, 5, 9, 10, 6-2, 8, 3, 1, 7 sum: 34 prod: 336 dups: 0
0.001008
6, 4, 10, 5, 8-3, 1, 9, 7, 2 sum: 33 prod: 378 dups: 0
0.001995
```

Terminal de IPython Historial de comandos

Permisos: RW Fin de línea: CRLF Codificación: UTF-8 Línea: 1 Columna: 1 Memoria: 54 %

Figura 2: Resultados de la segunda prueba.

En la segunda prueba del algoritmo se hicieron los cambios posibles para cambiar los dos números, para esto fue necesario mover a el archivo genetic.py, aun así después de ejecutar este segundo intento se volvieron a tener problemas con la ejecución. Para resolver esto fue necesario mejorar la función mutar del archivo cartas.py. Ref:2

```

12 El problema de las cartas
13 Fecha: 30 de Marzo de 2020
14 @author: Valenzuela
15
16
17 import unittest
18 import datetime
19 import genetic1
20 import operator
21 import functools
22 import random
23
24 class PruebasDeCartas(unittest.TestCase):
25     def obtener_apitud(self, genes):
26         sumaDelGrupo1 = sum(genes[0:5]) #se suma los datos
27         productoDelGrupo2 = functools.reduce(operator.mul, genes[5:10]) #multiplicacion del set
28         duplicados=(len(genes)-len(set(genes))) #diferencia del tamaño de los genes
29         return Aptitud(sumaDelGrupo1, productoDelGrupo2, duplicados) #Se regresa a Aptitud
30
31     def test(self): #Funcion test
32         geneSet = [i+1 for i in range (10)] # for el cual se hara 10 veces para meter datos e
33         horaInicio = datetime.datetime.now() #se obtiene hora actual
34
35         def fnMostrar(candidato):
36             mostrar(candidato, horaInicio) #funcion que sirve para mostrar los datos, es llama
37
38         def fnObtenerAptitud(genes):
39             return self.obtener_apitud(genes) #se regresa genes a la funcion obtener_apitud,
40
41         def fnMutar(genes):
42             mutar(genes, geneSet) #se manda a llamar a la funcion mutar
43
44         aptitudOptima= Aptitud(36,360,0)
45         mejor= genetic1.obtener_mejor(fnObtenerAptitud,10,aptitudOptima,geneSet,fnMostrar,mutar)
46         self.assertTrue(not aptitudOptima > mejor.Aptitud)
47
48
49

```

```

In [11]: runfile('C:/Users/Valenzuela/Documents/AlgoritmosGeneticos/Proyecto02/run.py', wdir='C:/Users/Valenzuela/Documents/AlgoritmosGeneticos/Proyecto02')
Reloaded modules: cartas, genetic1
10, 2, 9, 5, 7-8, 6, 1, 4, 3 sum: 33 prod: 576 dups: 0
0.0
9, 6, 10, 7, 5-2, 8, 1, 4, 3 sum: 37 prod: 192 dups: 0
0.0
6, 2, 10, 7, 9-5, 1, 8, 4, 3 sum: 34 prod: 480 dups: 0
0.0
9, 8, 6, 7, 3-5, 2, 10, 1, 4 sum: 33 prod: 400 dups: 0
0.0
6, 10, 9, 7, 3-5, 2, 8, 1, 4 sum: 35 prod: 320 dups: 0
0.0
8, 10, 6, 7, 3-5, 4, 2, 1, 9 sum: 34 prod: 360 dups: 0
0.0
9, 10, 2, 8, 7-5, 4, 1, 3, 6 sum: 36 prod: 360 dups: 0
0.01562
In [12]:

```

Figura 3: Resultados de la tercer prueba.

Una vez que se mejoró la función y se volvió a ejecutar se pudo observar que el error fue arreglado y esta vez el algoritmo fue correctamente terminado. Ref: 3

### 3. Conclusión

El problema de las cartas es un problema que me costa bastante entender, así que tuve que investigar más al respecto y repasar el recurso que nos proporciona para entender.

Esta práctica nos hace aprender aún más los casos prácticos que tienen los algoritmos genéticos, aumentamos un poco más la complejidad de el archivo genetic para resolver este problema en específico y gracias a el fragmento del libro se logró solucionar el problema de las cartas.

Para finalizar creo que seria bueno una retroalimentación de este problema más específicamente, entiendo la mayor parte del código pero algunas partes pueden ser difíciles de entender.

### 4. Referencias

(2019). Algoritmos Genéticos Con Python