Instituto Politécnico Nacional

Escuela Superior de Cómputo

Genetic Algorithms

Profesor: M. en C. Maria Elena Cruz Meza

Practica 2
Importancia de la representación en los AG.

Grupo: 3CM8

Objetivo: Mostrar los conocimientos en la creación de conjuntos binarios para comprender cromosomas aleatorios.

Pérez Garduño José Emiliano

Fecha: 18 de septiembre de 2019

Marco Teórico:

Representación Binaria: The binary number system differs from the decimal number system in that units are grouped by twos, fours, eights, etc. That is, the group sizes are powers of two instead of powers of ten. For example, twenty-three can be grouped into eleven groups of two with one left over. The eleven twos can be grouped into five groups of four with one group of two left over. Continuing along the same lines, we find that twenty-three can be described as one sixteen, zero eights, one four, one two, and one one, which is abbreviated “10111” two, or simply 10111 if the context is clear.

The process that we used to determine the binary representation of 23 can be described in general terms to determine the binary representation of any positive integer n. A general description of a process such as this one is called an algorithm.

1. Start with an empty list of bits.
2. Assign the variable k the value n
3. While k's value is positive, continue performing the following three steps until k becomes zero and then stop.
   a) divide k by 2, obtaining a quotient q (often denoted k div 2) and a remainder r (denoted (kmod2)).
   b) attach r to the left-hand side of the list of bits.
   c) assign the variable k the value q.

Codigo Gray:
A Gray code is an encoding of numbers so that adjacent numbers have a single digit differing by 1. The term Gray code is often used to refer to a "reflected" code, or more specifically still, the binary reflected Gray code.

To convert a binary number d_1d_2...d_(n-1)d_n to its corresponding binary reflected Gray code, start at the right with the digit d_n (the nth, or last, digit). If the d_(n-1) is 1, replace d_n by 1-d_n; otherwise, leave it unchanged. Then proceed to d_(n-1). Continue up to the first digit d_1, which is kept the same since d_0 is assumed to be a 0. The resulting number g_1g_2...g_(n-1)g_n is the reflected binary Gray code.

To convert a binary reflected Gray code g_1g_2...g_(n-1)g_n to a binary number, start again with the nth digit, and compute

$$\Sigma_n \equiv \sum_{i=1}^{n-1} g_i \pmod 2.$$

If Sigma_n is 1, replace g_n by 1-g_n; otherwise, leave it the unchanged. Next compute

$$\Sigma_{n-1} \equiv \sum_{i=1}^{n-2} g_i \pmod 2,$$

and so on. The resulting number d_1d_2...d_(n-1)d_n is the binary number corresponding to the initial binary reflected Gray code.

Desarrollo de la Práctica:

Código Fuente:

```
#!/usr/bin/python
#Codigo realizado por Jose Emiliano para la materia de Algoritmos Geneticos 3CM8
import re
import random

#Crea un bit aleatorio a partir de una probabilidad de 1/2
def CrearBinarioAleatorio():
    x = random.random()
    return 1 if x > 0.5 else 0

#Crea un numero aleatorio del rango de bits indicado por el usuario
def CrearNumAleatorio(rango):
    num = 0
    aux = '';
    for x in range(0,rango):
        aux = aux + str(CrearBinarioAleatorio())
    num = int(aux)
    return num

#crea arreglo de binarios que seran poblados
def CrearBinarios():
    tam_array = VerificacionInputListas()
    print("Introducir el numero de conjuntos que habra"),
    tam_conjunto = VerificacionInputEntero()
    Binario = [None] * tam_array
    print("Introducir el rango de bits que tendra"),
    rango_bits = VerificacionInputEntero()
    for y in range(0,tam_array):
        aux = [None] * tam_conjunto
        for x in range(0,tam_conjunto):
            a = CrearNumAleatorio(rango_bits)
            aux[x] = a
        Binario[y] = aux
    return Binario

def PoblarArregloDecimal():
    min_range = input("Introducir el limite menor de generacion.\n")
    max_range = input("Introducir el limite mayor de generacion.\n")
    longitud = input("Introducir el numero de objetos que generara.\n")
    Decimal = [None] * longitud
    for x in range (0,longitud):
        Decimal[x] = random.uniform(min_range,max_range)
    return Decimal
```

```
#Crea el arreglo de decimales a partir del arreglo de binario ya creado
def TransformarBinarioDecimal(Binario):
    Decimal = [None] * len(Binario)
    aux = []
    for y in range (0, len(Binario)):
        for x in range(0,len(Binario[y])):
            aux.append(int(str(Binario[y][x]),2))
        Decimal[y] = aux
        aux = []
    return Decimal

#Crea el arreglo de codigo gray a partir del arreglo de binarios ya creado
def TransformarBinarioGray(Binario):
    Gray = [] * len(Binario)
    for y in range(0,len(Binario)):
        aux = [] * len(Binario[y])
        for x in range(0,len(Binario[y])):
            a = int(str(Binario[y][x]),2)
            a ^= (a >> 1)
            a = bin(a)[2:]
            aux.append(a)
        Gray.append(aux)
    return Gray

#Verifica que el numero introducido por el usuario sea un entero
def VerificacionInputEntero():
    flag = False
    while not flag:
        n = raw_input("con un numero entero\n")
        patron = re.compile(r"^[0-9][0-9]*\.?[0-9]*")
        flag = re.search(patron, n)
    return int(n)

#Verifica que el numero sea entero y par
def VerificacionInputListas():
    flag = False
    while not flag:
        n = raw_input("Introducir el numero de arreglos que habra (debe ser par)\n")
        patron = re.compile(r"^[0-9][0-9]*\.?[0-9]*")
        flag = re.search(patron, n)
        if flag:
            if int(n) % 2 == 0:
                return int(n)
            else:
                flag = False
```

```
#Imprime todas las listas
def ImprimirTodo(Binario,Decimal,Gray):
    print("Los elementos en binario son: ")
    for y in range (0,len(Binario)):
        print("L[{}] = [".format(y)),
        for x in range (0,len(Binario[y])):
            if x == len(Binario[y]) - 1:
                print("{}".format(Binario[y][x])),
            else:
                print("{}, ".format(Binario[y][x])),
        print("]\n")

    print("Los elementos en decimal son: ")
    for y in range (0, len(Decimal)):
        print("L[{}] = [".format(y)),
        for x in range (0,len(Decimal[y])):
            if x == len(Decimal[y]) - 1:
                print("{}".format(Decimal[y][x])),
            else:
                print("{},".format(Decimal[y][x])),
        print("]\n")


    print("Los elementos en Gray son: ")
    for y in range (0, len(Gray)):
        print("L[{}] = [".format(y)),
        for x in range (0,len(Gray[y])):
            if x == len(Gray[y]) - 1:
                print("{}".format(Gray[y][x])),
            else:
                print("{},".format(Gray[y][x])),
        print("]\n")

def main():
    print("Jose Emiliano Perez Garduno")
    print("********Practica 2********")

    Binarios = CrearBinarios()
```

```
    Decimal = TransformarBinarioDecimal(Binarios)
    Gray = TransformarBinarioGray(Binarios)
    ImprimirTodo(Binarios,Decimal,Gray)


if __name__ == '__main__':
    main()
```

Resultado:

```
[emiliano@localhost Practica2]$ ./final.py
Jose Emiliano Perez Garduno
********Practica 2********
Introducir el numero de arreglos que habra (debe ser par)
4
Introducir el numero de conjuntos que habra con un numero entero
4
Introducir el rango de bits que tendra con un numero entero
4
Introducir el limite menor de generacion.
0.2
Introducir el limite mayor de generacion.
1.8
Introducir el numero de objetos que generara.
20
Los elementos en binario son:
L[0] = [ 111,  11,  1110,  1100 ]

L[1] = [ 100,  111,  1110,  1101 ]

L[2] = [ 101,  0,  101,  11 ]

L[3] = [ 110,  1001,  110,  1000 ]

Los elementos en Gray son:
L[0] = [ 100, 10, 1001, 1010 ]

L[1] = [ 110, 100, 1001, 1011 ]

L[2] = [ 111, 0, 111, 10 ]

L[3] = [ 101, 1101, 101, 1100 ]

Los elementos creados en decimal son:
Decimal = [ 0.56, 0.52, 1.50, 0.31, 1.47, 0.74, 0.91, 1.74, 1.76, 1.44, 1.73, 1.25, 0.49, 1.46, 0.59, 0.67, 0.30, 0.84, 0.89, 0.23 ]
```

Referencias:

https://faculty.uml.edu/klevasseur/ads/s-binary_Representation_of_Positive_Integers.html
http://mathworld.wolfram.com/GrayCode.html