|  |  |  |
| --- | --- | --- |
|  | **Instituto Politécnico Nacional**  *Escuela Superior de Cómputo* |  |
|  | **Genetic Algorithms** |  |
|  | Profesor: M. en C. Maria Elena Cruz Meza |  |

**Practica 3**

**Creación de un AG simple**

Grupo: 3CM8

Objetivo: Mostrar los conocimientos en la creación algoritmos genéticos, la selección de población, cruza y función fitness de la población.

Pérez Garduño José Emiliano

*Fecha: 22 de octubre 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.
   1. divide k by 2, obtaining a quotient q (often denoted k div 2) and a remainder r (denoted (kmod2)).
   2. attach r to the left-hand side of the list of bits.
   3. 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



If Sigma\_n is 1, replace g\_n by 1-g\_n; otherwise, leave it the unchanged. Next compute



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.

Algoritmo Genético: Los algoritmos genéticos (AG) funcionan entre el conjunto de soluciones de un problema llamado fenotipo, y el conjunto de individuos de una población natural, codificando la información de cada solución en una cadena, generalmente binaria, llamada cromosoma. Los símbolos que forman la cadena son llamados genes. Cuando la representación de los cromosomas se hace con cadenas de dígitos binarios se le conoce como genotipo. Los cromosomas evolucionan a través de iteraciones, llamadas generaciones. En cada generación, los cromosomas son evaluados usando alguna medida de aptitud. Las siguientes generaciones (nuevos cromosomas), son generadas aplicando los operadores genéticos repetidamente, siendo estos los operadores de selección, cruzamiento, mutación y reemplazo.

Un algoritmo genético puede presentar diversas variaciones, dependiendo de cómo se aplican los operadores genéticos (cruzamiento, mutación), de cómo se realiza la selección y de cómo se decide el reemplazo de los individuos para formar la nueva población. En general, el pseudocódigo consiste en los siguientes pasos:

* Inicialización: Se genera aleatoriamente la población inicial, que está constituida por un conjunto de cromosomas los cuales representan las posibles soluciones del problema. En caso de no hacerlo aleatoriamente, es importante garantizar que dentro de la población inicial, se tenga la diversidad estructural de estas soluciones para tener una representación de la mayor parte de la población posible o al menos evitar la convergencia prematura.
* Evaluación: A cada uno de los cromosomas de esta población se aplicará la función de aptitud para saber cómo de "buena" es la solución que se está codificando.
* Condición de término: El AG se deberá detener cuando se alcance la solución óptima, pero esta generalmente se desconoce, por lo que se deben utilizar otros criterios de detención. Normalmente se usan dos criterios: correr el AG un número máximo de iteraciones (generaciones) o detenerlo cuando no haya cambios en la población. Mientras no se cumpla la condición de término se hace lo siguiente:
* Selección: Después de saber la aptitud de cada cromosoma se procede a elegir los cromosomas que serán cruzados en la siguiente generación. Los cromosomas con mejor aptitud tienen mayor probabilidad de ser seleccionados.
* Recombinación o cruzamiento: La recombinación es el principal operador genético, representa la reproducción sexual, opera sobre dos cromosomas a la vez para generar dos descendientes donde se combinan las características de ambos cromosomas padres.
* Mutación: Modifica al azar parte del cromosoma de los individuos, y permite alcanzar zonas del espacio de búsqueda que no estaban cubiertas por los individuos de la población actual.
* Reemplazo: Una vez aplicados los operadores genéticos, se seleccionan los mejores individuos para conformar la población de la generación siguiente.

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:

## 

Referencias:

[*https://faculty.uml.edu/klevasseur/ads/s-binary\_Representation\_of\_Positive\_Integers.html*](https://faculty.uml.edu/klevasseur/ads/s-binary_Representation_of_Positive_Integers.html)

*http://mathworld.wolfram.com/GrayCode.html*