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.
   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.

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

def FuncionFitness(n):
    x = n * n
    return x

def CalcularLongBinaria(n):
    long_bit = n.bit_length()
    max_bit = 2 ** long_bit
    print("La longitud maxima del rango es: 2^{}={}".format(long_bit,max_bit))
    return long_bit

def CalcularProbabilidad(prob):
    x = random.random()
    return 1 if x < prob else 0

def GuardarNumAleatorio(arreglo,x):
    arreglo.append(x)
    return arreglo

#Crea un bit aleatorio a partir de una probabilidad de 1/2

def RegresarBitAleatorio(x):
    return 1 if x > 0.5 else 0

#Crea un numero aleatorio del rango de bits indicado por el usuario
def CrearNumAleatorio(rango,arreglo):
    aux = ''
    num = 0
    for x in range(0,rango):
        rand = random.random()
        arreglo.append(rand)
        print("Guardo el numero {} de la iteracion {}".format(rand,x))
        aux = aux + str(RegresarBitAleatorio(rand))
    num = int(aux)
    arreglo.append(num)
    return arreglo

def VerificacionINT(str):
    try:
```

```
        value = int(str)
        print(value)
        flag = True
    except ValueError:
        print("Escoger un numero entero positivo")
        flag = False
    return flag

def BinarioADecimal(n):
    for x in range(0,len(n)):
        n[x] = int(n[x])
    return int(str(n[0]),2)

def OpcionFitness():
        print("Se realizara la funcion fitness")
        flag = False
        while(flag == False):
            user_input = input("Indicar el limite inferior del rango\n")
            flag = VerificacionINT(user_input)
            if flag == True:
                lim_inf = int(user_input)
        flag = False
        while(flag == False):
            user_input = input("Indicar el limite superior del rango\n")
            flag = VerificacionINT(user_input)
            if flag == True:
                lim_sup = int(user_input)
                if(lim_sup <= lim_inf):
                    print("Escoger un limite superior mayor que al menor")
                    flag = False
        flag = False
        while(flag == False):
            user_input = input("Indicar el numero de individuos\n")
            flag = VerificacionINT(user_input)
            if(flag == True):
                num_individuos = int(user_input)
                if(num_individuos % 2 == 0):
                    flag = True
                else:
                    print("El numero de individuos debe ser par")
                    flag = False

        max_bit = CalcularLongBinaria(lim_sup)
        print("Se crearan {0:.0f} pares de individuos".format(num_individuos/2))

        conjunto_individuos = [None] * num_individuos
```

```
        arreglo_num_aleatorios = []

        for y in range(0,num_individuos):
            aux = [None] * (lim_sup-lim_inf)
            a = CrearNumAleatorio(max_bit,arreglo_num_aleatorios)
            arreglo_num_aleatorios = [item for item in a if isinstance(item,float)]
            a = [item for item in a if isinstance(item,int)]
            aux = a
            conjunto_individuos[y] = aux

        print("El conjunto de individuos queda como:")
        for x in range(0,num_individuos):
            print(conjunto_individuos[x])

        print("El conjunto de numeros aleatorios queda como:")
        for x in range(0,len(arreglo_num_aleatorios)):
            print("{0:.2f}".format(arreglo_num_aleatorios[x]))

        flag = False
        while(flag == False):
            user_input = input("Indicar la probabilidad de cruza[0-100]\n")
            flag = VerificacionINT(user_input)
            if(flag == True):
                prob_cruce = int(user_input)
                if(prob_cruce <= 0):
                    print("Introducir un numero mayor a 0")
                    flag = False
                else:
                    prob_cruce = prob_cruce / 100
                    flag = True
                    break

        print("La probabilidad de cruce es: {}".format(prob_cruce))

        flag = False
        while(flag == False):
            user_input = input("Indicar la probabilidad de mutacion[0-100]\n")
            flag = VerificacionINT(user_input)
            if(flag == True):
                prob_mut = int(user_input)
                if(prob_mut <= 0):
                    print("Introducir un numero mayor a 0")
                    flag = False
                else:
                    prob_mut = prob_mut / 100
                    break
```

```
        print("La probabilidad de mutacion es: {}".format(prob_mut))
        print("No. Cadena\tPoblacion Inicial\tValor x\t\tAptitud f(x)\tProb i\tProb Acumu
lada\tPadres Elegidos")
        array_aptitud = [None] * num_individuos
        array_prob = [None] * num_individuos
        for y in range (0,num_individuos):
            bin_decimal = BinarioADecimal(conjunto_individuos[y])
            fit_decimal = FuncionFitness(bin_decimal)
            array_aptitud[y] = fit_decimal
        suma_aptitud = sum(array_aptitud)
        promedio_aptitud = suma_aptitud/num_individuos
        max_aptitud = max(array_aptitud)
        prob_acum = 0
        for y in range (0,num_individuos):
            bin_decimal = BinarioADecimal(conjunto_individuos[y])
            fit_decimal = FuncionFitness(bin_decimal)
            array_prob[y] = array_aptitud[y] / suma_aptitud
            print("{}\t\t{}\t\t\t{}\t\t{}".format(y+1,conjunto_individuos[y],bin_decimal,
fit_decimal),end='\t\t')
            print("{0:.2f}".format(array_prob[y]),end='\t')
            prob_acum = prob_acum + array_prob[y]
            print("{0:.2f}".format(prob_acum))
        print("Suma\t\t\t\t\t\t{}".format(suma_aptitud))
        print("Promedio\t\t\t\t\t{}".format(promedio_aptitud))
        print("Max\t\t\t\t\t\t{}".format(max_aptitud))

def OpcionGeneralizacional():
    print("Se realizara la Generalizacional")

def OpcionElitista():
    print("Se realizara la funcion Elitista")

def menu():
    opc = -1
    while (opc > 3 or opc < 0):
        print("1.-Funcion Fitness")
        print("2.-Generalizacional")
        print("3.-Elitista")
        print("0.-Salir")
        opc = int(input("Escoger opcion\n"))

    if(opc == 1):
        #Función Fitness
        OpcionFitness()
```

```
    elif(opc == 2):
        #Generalizacional
        OpcionGeneralizacional()
    elif(opc == 3):
        #Elitista
        OpcionElitista()
    elif(opc == 0):
        print("Adios")
        return 0

def main():
    print("Jose Emiliano Perez Garduno")
    print("********Practica 3********")
    menu()

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

Resultado:

```
Jose Emiliano Perez Garduno
********Practica 3********
1.-Funcion Fitness
2.-Generalizacional
3.-Elitista
0.-Salir
Escoger opcion
1
Se realizara la funcion fitness
Indicar el limite inferior del rango
0
0
Indicar el limite superior del rango
15
15
Indicar el numero de individuos
4
4
La longitud maxima del rango es: 2^4=16
Se crearan 2 pares de individuos
El conjunto de individuos queda como:
[1001],[1111],[11],[1010],

El conjunto de numeros aleatorios queda como:
0.80,0.10,0.23,0.78,0.61,0.66,0.80,0.99,0.36,0.43,0.89,0.70,0.66,0.47,0.51,0.23,

Indicar la probabilidad de cruza[0-100]
70
70
La probabilidad de cruce es: 0.7
Indicar la probabilidad de mutacion[0-100]
30
30
La probabilidad de mutacion es: 0.3
No. Cadena      Poblacion Inicial       Valor x         Aptitud f(x)    Prob i  Prob Acumulada  Padres Elegidos
1               [1001]                  9               81              0.20    0.20
2               [1111]                  15              225             0.54    0.74
3               [11]                    3               9               0.02    0.76
4               [1010]                  10              100             0.24    1.00
Suma                                                    415
Promedio                                                103.75
Max                                                     225
```

Referencias:

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