INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

GENETIC ALGORITHMS

REPORTE PRACTICA II

"REPRESENTACIÓN DE INDIVIDUOS"

ALUMNO: SOLIS SANCHEZ JHOVANY

GRUPO: 3CM5

BOLETA: 2015630489

INTRODUCCIÓN

Los objetivos de esta practica serán mostrar alguna formas de representar individuos en algoritmos genéticos. Las representaciones que se implementaran serán la binaria, código de Gray, codificación de números reales y codificación de números enteros. El llenado de datos se hará aleatoriamente.

Representación Binaria.
La representación tradicional usada para codificar un conjunto de soluciones es el esquema binario en el cual un cromosoma 1 es una cadena de la forma [1 0 1 0 ...] donde 1 y 0 se denominan alelos.

Código de Gray.
Podemos convertir cualquier número binario a un código de Gray haciendo XOR a sus bits consecutivos de derecha a izquierda. Por ejemplo, dado el número 0101 en binario, ejemplo: 1 ⊕ 0 = 1, 0 ⊕ 1 = 1, 1 ⊕ 0 = 1, produciéndose (el último bit de la izquierda permanece igual) 0111, el cual es el código de Gray equivalente.

Codificación de números reales.
Un número real se representa usando 32 bits, de los cuales 8 se usan para el exponente usando una notación en exceso-127, y la mantisa se representa con 23 bits, se podrá manejar un rango relativamente grande de números reales usando una cantidad fija de bits.

Codificación de números enteros.
Se supone una posición fija para el punto decimal en cada variable, aunque esta posición no tiene que ser necesariamente la misma para el resto de las variables codificadas en la misma cadena. La precisión está limitada por la longitud de la cadena, y puede incrementarse o decrementarse según se desee.

DESARROLLO

La forma de seleccionar alguna representación de individuos es a través de un menú de opciones.

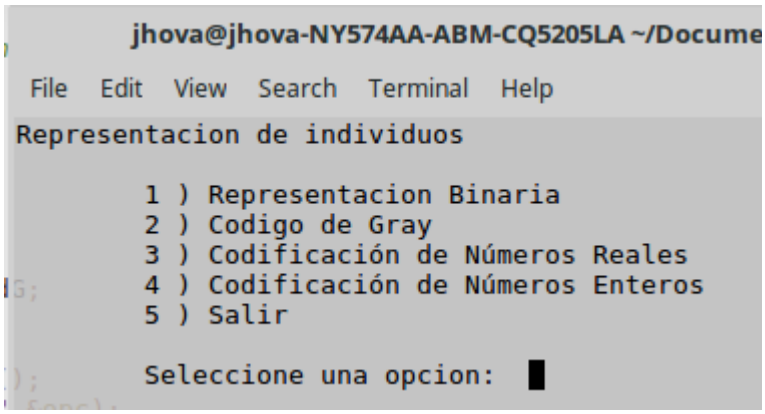

Illustration 1: Menú

La opción de representación binaria pide como parámetros el numero de bits del individuo y cuantos ejemplares se quieren mostrar.

```
Representacion de individuos

        1 ) Representacion Binaria
        2 ) Codigo de Gray
        3 ) Codificación de Números Reales
        4 ) Codificación de Números Enteros
        5 ) Salir

        Seleccione una opcion:  1

Representacion Binaria
Numero de bits, Numero de individuos "8 5" :    10 10
```

Illustration 3: Binary

Como resultado se obtiene:

```
Individuo       Binario

0 ->    |1|1|1|0|1|1|1|1|1|1|
1 ->    |1|0|1|0|1|0|1|0|1|1|
2 ->    |1|1|1|0|0|0|0|0|0|0|
3 ->    |1|1|0|1|1|0|1|0|1|1|
4 ->    |0|1|1|0|1|1|1|0|1|1|
5 ->    |1|1|1|1|0|0|1|0|1|0|
6 ->    |0|0|1|0|1|0|0|0|1|0|
7 ->    |1|1|1|0|1|0|1|1|1|1|
8 ->    |1|1|0|0|0|1|1|1|1|1|
9 ->    |0|0|1|0|1|1|0|0|1|1|
```

Illustration 2: Individuos Binario

```c
/* Función que obtiene un número aleatorio dependiendo los
    bits de rango que se quiera y lo almacena en un array */
int indBinary(char *ind,int bits){
    int num=0, p=0;
    double rango = pow(2,bits);
    srand(time(NULL)+rand());
    num = (rand() % (int)rango);
    getBinary(num,p,ind);
    return (int)num;
}

/* Funcion recursiva para convertir un número decimal
    a binario */
void getBinary(int num, int p, char *c){
    if(num==0) return;
    c[p]=num%2;
    num=num/2;
    getBinary(num,p+1,c);
}

void getGrayCode(char *indB,char *indG, int bits){
    indG[0] = xor(0,indB[0]);
    for(int i=bits-1;i>0;i--)
        indG[i] = xor(indB[i-1],indB[i]);
}

// Función lógica
char xor(char a, char b){
    if(a==b) return 0;
    return 1;
}
```

Illustration 4: Codigo Fuente Binario

El código utilizado para realizar esta representación se basa en generar primeramente un numero aleatorio dependiendo del el número de bits que el usuario indique.

A continuación el numero se convierte a código binario a través de una función recursiva.

El resultado se guarda en un array según su posición en el código binario.

La opción de cogido de Gray muestra una conversión de un código binario a código de Gray. Al igual que la opción de representación binaria el usuario tiene que ingresar los bits del individuo y el numero de ejemplares. Para tener como resultado:

```
Individuo       Binario         Codigo Gray
0 ->    |1|1|1|1|0|1|1|0|1|1|-> |1|0|0|0|1|1|0|1|1|0|
1 ->    |1|1|1|0|0|1|1|0|1|0|-> |1|0|0|1|0|1|0|1|1|1|
2 ->    |0|0|0|0|1|1|0|0|0|1|-> |0|0|0|0|1|0|1|0|0|1|
3 ->    |0|0|1|1|1|1|1|0|1|1|-> |0|0|1|0|0|0|0|1|1|0|
4 ->    |0|0|1|0|1|1|0|1|1|1|-> |0|0|1|1|1|0|1|1|0|0|
5 ->    |0|0|1|0|1|0|1|0|0|0|-> |0|0|1|1|1|1|1|1|0|0|
6 ->    |0|1|1|0|0|0|0|1|1|1|-> |0|1|0|1|0|0|0|1|0|0|
7 ->    |0|0|0|1|1|1|1|0|1|1|-> |0|0|0|1|0|0|0|1|1|0|
8 ->    |1|0|0|1|1|0|0|1|1|1|-> |1|1|0|1|0|1|0|1|0|0|
9 ->    |1|1|0|0|1|0|1|0|0|0|-> |1|0|1|0|1|1|1|1|0|0|
```

Illustration 5: Codigo Gray Individuos

```
void getGrayCode(char *indB,char *indG, int bits){
    indG[0] = xor(0,indB[0]);
    for(int i=bits-1;i>0;i--)
        indG[i] = xor(indB[i-1],indB[i]);
}

// Función lógica
char xor(char a, char b){
    if(a==b) return 0;
    return 1;
}
```

Illustration 6: Codigo Fuente Gray

Para esta funciona los parámetros que necesita son una array con el individuo en binario para poder aplicar la función xor y un array donde guardar los resultados.

La función xor es la implementación de la tabla de verdad de esta misma. Retorna un valor 0 si ambas entradas son iguales en caso contrario retorna 1.

Para la codificación de números reales el usuario solo tiene que ingresa el numero que desea codificar. El programa dividirá la parte entera de la parte flotante y combatir a cada una en su correspondiente binario para después unirlas.

```
       Seleccione una opcion:  3
Codificacion de Numeros Reales
Numero Real :   12.54
|0|0|1|1|1|1|1|1|0|0|-> |0|0|1|1|1|0|1|0|0|0|
```

Illustration 7: Codificacion numero real

Finalmente en la codificación de números enteros el usuario ingresara un numero entero y el programa se encargara de dividirlo y almacenar cada dígito en la posición indicada en el arreglo.

```
       Seleccione una opcion:  4

Codificacion en Numeros Enteros
Numero a codificar :    1234568

Entero  Codificado

1234568 ->      |1|2|3|4|5|6|8|
```

Illustration 8: Codificacion de numeros enteros

```
void codEntero(char *indB, int num, int bits){
    int x = num;
    for(int i=bits-1;i>=0;i--){
        indB[i] = x % 10;
        x /= 10;
    }
}

int countBits(int num){
    int x = num;
    int bits=0;
    while(x){
        x /= 10;
        bits+=1;
    }
    return bits;
}
```

Illustration 9: Codigo Codificador entero

Con este código se obtiene el numero de bits que ocupara el numero entero en el arreglo para con ello poder crearlo, después se van almacenando cada uno de ellos en la posición indicada.

CONCLUSIÓN

Como se puede ver en algunos códigos fuente la representación de individuos puede ser muy sencilla de implementar como en el caso de la representación binaria y el código gray pero en algunas ocasiones la implementación podría generar el uso de mas recursos computacionales como es el caso de las implementaciones de la codificación de números reales y enteros y con ello se podría complicar la manipularon de esta ultimas en algunas otra operaciones. A mi parecer todo dependerá de que problema se desea resolver se podrá utilizar alguna u otra.