PROGRAMACIÓN AVANZADA

AEC1: DISEÑO E IMPLEMENTACIÓN DE UN SI UTILIZANDO ALGORITMOS GENÉTICOS

Antonio Luis Ojeda Soto

INDICE DE LA MEMORIA

CLASE CROMOSOMA.

Comienzo creando la clase abstracta Cromosoma para poder crear cromosomas concretos y sacar ciertos resultados de ellos. Veremos ahora los métodos más importantes de la clase Cromosoma:

- Primer constructor con parámetro de array booleano de genes: aquí reservaremos memoria para los genes del cromosoma.
- Segundo constructor con parámetros de límites y tolerancia: aquí reservamos la memoria para los límites superior e inferior y la tolerancia de representación.
- **crearCromosoma()**: primero calculará la longitud del cromosoma concreto a construir basándose en los parámetros que le introduzcamos, para ello utilizará el método obternerLongitud() de la clase Utilidades. Una vez obtenida la longitud del array de booleanos iremos eligiendo cada gen del cromosoma de forma aleatoria utilizando el método generarAleatorio() de la clase Utilidades.
- **fenotipo():** calculará el fenotipo de cada cromosoma concreto apoyándose en el método toIntValue() de la clase Utilidades.
- **clone():** devolverá un clon de un individuo exactamente igual al que le indiquemos.
- **mutar():** se encargará de cambiar un elemento del array de booleanos que conforma el individuo de forma aleatoria. Para ello se apoyará en el método generarAleatoriosEnteros() el cual escogerá un número entre 0 y la longitud del array de booleanos especificado.
- **toString():** sobrescritura del método para que muestre los individuos en binario en lugar de booleanos.

CLASES UTILIDADES.

A medida que voy creando la clase Cromosoma voy creando a la par esta clase. Voy creando los métodos necesarios para poder dar utilidad a la clase Cromosoma. También se irán añadiendo métodos cuando comience a crear la clase Algoritmo más adelante.

Descripción de los métodos:

- **obternerLongitud():** método que va a calcular la longitud de nuestro cromosoma según los parámetros que le digamos.
- **generarAleatorio():** generará un booleano aleatoriamente. Será utilizada por clase Cromosoma.

- **toIntValue():** método que devuelve el valor decimal (entero) de array de booleanos. Lo utilizará el método fenotipo() de la clase Cromosoma.
- **generarAleatorioEnteros():** método que devuelve un entero aleatoriamente de un rango entre 0 y el entero que se pase por parámetro. Lo utilizará el método mutar() de la clase Cromosoma.
- **generarAleatorioRangoEnteros():** método para generar aleatorios de enteros entre un rango pasado por parámetro.
- **generarAleatorioRangoDouble():** método para generar aleatorios de decimales entre un rango pasado por parámetro.
- **generarAleatorioDouble():** método para generar un decimal de tamaño máximo pasado por parámetro.

CLASES DE LAS FUNCIONES

A continuación, creo las tres funciones del enunciado a través de tres clases: **Funcion1**, **Funcion2** y **Funcion3**, las cuales extenderán de la clase Cromosoma el método abstracto **aptitud(),** cada cual a su forma. Al constructor de estas tres clases se le pasará como parámetros los límites superior e inferior y la tolerancia de representación (los mismos que para crear un cromosoma), con los que se calculará el fenotipo de cada individuo el cual será pasado por la función quedando como resultado el verdadero valor de cada cromosoma que más tarde nos servirá para hacer la criba de individuos que queramos que pasen a la siguiente generación para su reproducción.

CLASES DE ORDENACIÓN

Estas clases implementarán de la clase **Comparator** el método **compare()** para comparar la aptitud de dos cromosomas pasados por parámetro en clase **OrdenacionMaximizacion** y **clase OrdenacionMinimizacion** según convenga. Serán unas clases muy utilizadas por la clase Algoritmo el cual las llamará a través del método **sort()** de ordenación, ordenando de esta forma una población dada por medio del valor de la aptitud de cada uno.

CLASE ALGORITMO

Comienzo poniendo todos los atributos y sus respectivos **getters** y **setters**, además de un constructor vacío de la clase Algoritmo.

Más tarde comienzo el método **ejecutar()** pedido en el enunciado que contiene el algoritmo propiamente dicho de la práctica. En el mismo método paso por parámetros una población de cromosomas y un tipo de operación (maximizar o minimizar).

A continuación, comienzo con el propio algoritmo requerido poniendo el primer bucle **while** que indica que mientras no termine se supere el número de generaciones especificado se siga probando con generaciones distintas. Para medir esto pondremos un contador de generaciones al final del bucle. Acto seguido creamos un array de población temporal con el tamaño de la población actual pasada por parámetro, una población de padres a cruzar y una población de descendientes que será el resultado de haber cruzado ciertos padres. Acto seguido ordenamos la población actual de mayor a menor si es para maximizar y de menor a mayor si es para la operación minimizar.

Luego indicamos que si hay un tamaño de élite cogeremos los mejores y los introduciremos en la población temporal. Con esto pasamos al segundo **while** el cual sigue dentro del primero, y nos indica que si la población temporal no está llena haremos una selección, bien por torneo o por ruleta, de la población actual.

La **selección por torneo** la hacemos eligiendo dos individuos al azar de la población actual y enfrentándolos entre ellos comparando la aptitud de cada uno. Gana el que tiene mayor aptitud si el tipo de operación es Maximizar y el de menor aptitud si es la operación minimizar. Y vamos introduciendo en el array padres los ganadores.

La **selección por ruleta** creando una variable que contendrá el total de aptitudes, un array que contendrá los fitness normalizados y otro con los fitness acumulados de cada individuo. Calculamos el total de aptitudes de la población, luego calculamos el fitness normalizado de cada individuo, que será la aptitud de cada uno dividida entre el total de aptitudes. A continuación, calculamos el fitness acumulado de cada individuo que será el su fitness normalizado más el fitness acumulado del elemento anterior dándonos así la porción de ruleta de cada uno. Luego creamos un número aleatorio decimal entre 0 y 1 cogeremos el individuo que esté entre este número aleatorio y el fitness acumulado del anterior individuo a este y lo iremos metiendo en la población de padres. Para selección por ruleta con minimizar sería lo mismo que lo anterior lo único que aquí meteremos el factor de corrección para que los individuos con menor valor tengan más porción de la ruleta.

Seguimos con el algoritmo y llegamos a los cruces donde los habrá si se supera la probabilidad de cruce. Su hay probabilidad de cruce cogeremos dos padres. Si el cruce elegido es por un punto (SPX, en el método **cruceSpx()**) se escogerá un número de posición de gen y a partir de este punto se cogerá la primera parte del primer padre con la segunda parte del segundo y obtendremos un descendiente. Para el segundo descendiente se cogerá la segunda parte del primer padre y la primera del segundo padre y obtendremos el segundo descendiente. Si el cruce elegido es cruce por uniforme (método **cruceUniforme()**) se creará un nuevo individuo de forma aleatoria

que servirá de máscara, la cual indicará si es valor true o false se cogerá un gen de un padre o de otro para formar los dos descendientes.

Si se supera la probabilidad de mutación se mutará un solo gen de ambos descendientes y estos dos descendientes serán introducidos en la población temporal siempre que ésta no esté llena.

Una vez que tenemos la población temporal llena la ordenamos según la opción maximizar o minimizar.

Finalmente se establece como nueva población actual la población temporal, se actualiza el contador de generaciones, se escoge como mejor individuo de cada generación el primer individuo de población actual, ya que está previamente ordenado, y se sacan por pantalla las características de cada generación.

CLASE PRUEBAS.

Creamos la clase pruebas para probar nuestro algoritmo con los parámetros del enunciado y nos salen los siguientes resultados:

Con tamaño de elite 1:

Generación 1 de 50: Mejor individuo=Cromosoma [genes=[11110101111], fenotipo=0.9609184171958964, aptitud=1.6693804190018664

Generación 2 de 50: Mejor individuo=Cromosoma [genes=[11110101111], fenotipo=0.9609184171958964, aptitud=1.6693804190018664

Generación 3 de 50: Mejor individuo=Cromosoma [genes=[11110101111], fenotipo=0.9609184171958964, aptitud=1.6693804190018664

Generación 4 de 50: Mejor individuo=Cromosoma [genes=[11100100111], fenotipo=0.8944797264289204, aptitud=1.8203290473722955

Generación 5 de 50: Mejor individuo=Cromosoma [genes=[11110100111], fenotipo=0.9570102589154861, aptitud=1.8816952929675315

Con tamaño de elite 2:

Generación 1 de 50: Mejor individuo=Cromosoma [genes=[11001010100], fenotipo=0.7914020517830972, aptitud=1.643821619269443

Generación 2 de 50: Mejor individuo=Cromosoma [genes=[11001010110], fenotipo=0.7923790913531998, aptitud=1.6919640559105547

Generación 3 de 50: Mejor individuo=Cromosoma [genes=[11001010110], fenotipo=0.7923790913531998, aptitud=1.6919640559105547

Generación 4 de 50: Mejor individuo=Cromosoma [genes=[11001011100], fenotipo=0.7953102100635075, aptitud=1.7829625117830683

Generación 5 de 50: Mejor individuo=Cromosoma [genes=[11011011100], fenotipo=0.8578407425500733, aptitud=1.8459692581844795

Generación 6 de 50: Mejor individuo=Cromosoma [genes=[11011011100], fenotipo=0.8578407425500733, aptitud=1.8459692581844795

CONCLUSIONES.

Me salen pocas generaciones, hay incluso algunas veces que no me sale ninguna generación y teniendo fallos de ordenación, pero esto es algo que deberé tener mal en el algoritmo. Cuando me salen generaciones enseguida converge hacia una solución de forma rápida. Esto me hace pensar que a lo mejor debería haber dado más oportunidades a los individuos peor adaptados para que no evolucionara tan rápido.

REFERENCIAS.

https://www.youtube.com/watch?v=_S8t-LV1BNo&list=PL6VsnkqbwvkKlmG617wluvBxl0F20369T&index=2

https://www.youtube.com/watch?v=xWFUfuJgwDU&list=PL6VsnkqbwvkKlmG617wluvBxl0F20369T&index=3

https://www.youtube.com/watch?v=mibDWs41gXY&list=PL6VsnkqbwvkKlmG617wluvBxl0F20369T&index=4

https://www.youtube.com/watch?v=L7ABeOYXPck&list=PL6VsnkqbwvkKtHPPpFTlDExnImvQ6T0MR&index=2