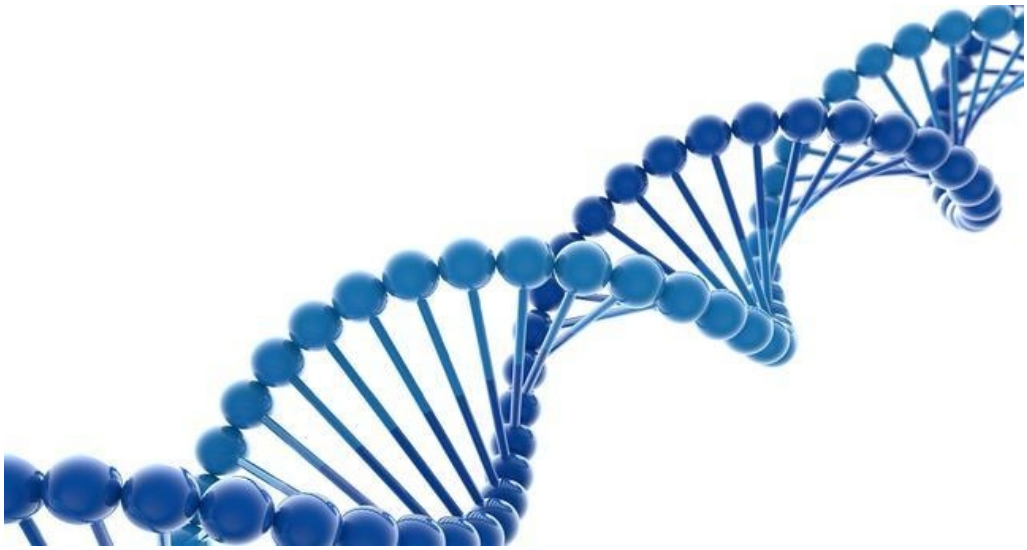
ALGORITMOS GENÉTICOS



Pascual Andrés Carrasco Gómez

Índice

1. Estudio del problema....3

2. Representación....3

3. Aptitud: Función fitness....4

4. Población inicial....4

5. Ciclo evolutivo....4

    5.1. Selección....4

    5.2. Cruce....5

    5.3. Mutación....5

    5.4. Remplazo....5

6. Condición de parada....6

7. Ejecución de ciclo evolutivo....6

8. Ejecuciones en Python....8

1. Estudio del problema

El problema se define en que dada una secuencia de números y un número objetivo se tiene que obtener un conjunto de operadores tal que la aplicación en orden de dichos operadores sobre la secuencia de números tiene que dar el número objetivo o una buena aproximación a dicho número.

Ejemplo:

Secuencia de números = [75, 50, 6, 3, 9, 7]

Objetivo = 248

Posible solución = ((((75 + 50) * 6) / 3) - 9) + 7) = 248

El problema tiene una complejidad exponencial definida por el número de operadores y el número de dígitos en la secuencia de números de entrada, es decir:

longitud(operadores) ^ (longitud(secuencia_números)-1)

En nuestro problema tenemos 4 operadores que son [+,-,*,/] y tenemos 6 dígitos en la secuencia de números, por lo tanto: $4^5 = 1024$, observamos que el problema explota en medida que añadimos operadores y más dígitos a la secuencia de números, por ejemplo, 8 operadores y 10 dígitos: $8^9 = 134217728$.

2. Representación

Nuestro problema tiene dos parámetros de entrada que son la secuencia de 6 números y el número objetivo. Estos dos parámetros son fijos (estáticos) , es decir, no forman parte de la representación del problema ya que no varían, pero los hemos de tener en cuenta para calcular la aptitud (fitness) de cada individuo y para establecer el criterio de parada como veremos posteriormente en los apartados correspondientes de la memoria.

Para nuestro problema un individuo de la población es una secuencia de operadores de longitud:

(longitud(secuencia_números) – 1)

En nuestro caso la longitud de secuencia de números es 6 por lo tanto la longitud de la secuencia de operadores es 6 – 1 = 5. Esto significa que cada individuo de nuestra población tendrá 5 operadores cuyo orden es importante ya que la ejecución es de izquierda a derechas (así evitamos el uso de paréntesis).

Vemos un ejemplo:

Secuencia de números = [2,5,4,3,4,6]

Individuo = [+,*,*,/,-]

Esto representa lo siguiente:

2 + 5 * 4 * 3 / 4 – 6 = 15

Observamos que no hay prioridad de operadores en el calculo, si no que el calculo se realiza de izquierda a derecha.

3. Aptitud: Función fitness

La función fitness nos permite medir la adecuación de cada individuo (de la población actual) al problema. Nuestra función fitness se basa en obtener la distancia de la solución obtenida en cada individuo con el número objetivo establecido, cuya formula es la siguiente:

F. fitness (individuo) = | objetivo - (solución individuo) |

Vamos a ver un ejemplo de calculo de la función fitness para una población de dos individuos

Secuencia de números = [2,5,4,3,4,6]

Objetivo: 25

Individuo 1 = [+,*,*,/,-]

Individuo 2 = [*,+,+,*,/]

La función fitness de cada individuo es la siguiente:

F. fitness (individuo 1) = | 25 - (2 + 5 * 4 * 3 / 4 – 6) | = | 25 – 15 | = 10

F. fitness (individuo 2) = | 25 - (2 * 5 + 4 + 3 * 4 / 6) | = | 25 – 11,33 | = 13,67

Observamos que el individuo 1 tiene una solución más buena que el individuo 2 ya que 15 esta más cerca de 25 que 11,33, por lo tanto el mejor individuo de la población actual es el mínimo de la función fitness de todos los individuo de dicha población.

4. Población inicial

La población inicial es un conjunto de individuos obtenidos de forma aleatoria, en nuestro problema hemos decidido que la población siempre sea de tres individuos, donde cada individuo se genera de la siguiente forma, como cada individuo esta compuesto de cinco operadores, pues generamos cinco operadores aleatorios, los cuales pueden estar repetidos.

5. Ciclo evolutivo

El ciclo evolutivo de un algoritmo genético esta formado por operadores evolutivos que nos permiten combinar soluciones existentes con el objetivo de obtener nuevas soluciones. Hay cuatro operadores evolutivos que son: Selección, Cruce, Mutación y Remplazo que vamos a describir a continuación para nuestro problema en concreto.

5.1. Selección

Consiste en seleccionar individuos con mejores funciones fitness que reciben el nombre de padres. En nuestro problema serán los individuos con mínimo valor de la función fitness, hemos decidido que la selección se realiza sobre dos individuos, es decir, en la selección obtenemos dos padres.

5.2. Cruce

El cruce consiste en combinar a los padres para obtener a los hijos. En nuestro problema se realiza un cruce uniforme que consiste en seleccionar de forma aleatoria uno de los dos padres para establecer un operador en el hijo, como resultado obtenemos dos hijos que son el opuesto uno de otro respecto a la procedencia del operador del padre, lo vemos con un ejemplo para que quede más claro:

Padre 1 = [*,+,+,*,+]

Padre 2 = [*,+,/,*,*]

Imaginemos que de forma aleatoria obtenemos [1,1,2,1,2] donde cada índice de la lista corresponde de que padre vamos a coger el operador, por lo tanto tendremos los siguiente hijos:

Hijo 1 = [*,+,/,*,*] con [1,1,2,1,2]

Hijo 2 = [*,+,+,*,+] con [2,2,1,2,1]

Nota: El cruce en nuestro problema es importante ya que al utilizar un cruce uniforme aseguramos que los hijos obtenidos tengan los operadores comunes (en naranja) que tienen los padres y esto hace que los hijos hereden la estructura común de sus padres.

5.3. Mutación

Se realiza una mutación aleatoria de cualquier operador del hijo (sea común entre los padres o no).

Nota: Esta mutación es importante porque imaginemos después de un conjunto de iteraciones que obtenemos los siguientes padres en la población actual:

Padre 1 = [+,+,+,+,+]

Padre 2 = [+,+,+,+,+]

Si no realizamos dicha mutación siempre obtendríamos los mismos individuos y nuestro algoritmo genético no tendría variedad, y la variedad en los individuos es lo que hace poderoso el algoritmo genético.

5.4. Remplazo

El remplazo en nuestro algoritmo consiste en remplazar los dos hijos que hemos obtenido por los dos individuos de la población actual que tengan mayor valor en la función fitness (es decir que sean los peores individuos en la población actual).

Este punto también es muy importante porque garantiza que siempre va a estar el mejor individuo hasta la fecha dentro de la población actual.

6. Condición de parada

La condición de parada en nuestro algoritmo depende de dos parámetros que son la solución del mejor individuo de la población actual y el número de iteraciones, es decir, nuestro algoritmo finaliza cuando o el valor de la solución de nuestro mejor individuo es igual al objetivo buscado o cuando el número de iteraciones es cero.

7. Ejecución de ciclo evolutivo

En este apartado de la memoria vamos a realizar paso a paso dos iteraciones del algoritmo genético que hemos implementado con la finalidad de agrupar todos los apartados de la memoria y tener una idea de como funciona internamente.

Datos de entrada: (Indicados por el usuario)

Secuencia de números = [2,4,1,6,3,5]

Objetivo = 15

Iteraciones = 2

Población inicial: (3 Individuos obtenidos aleatoriamente)

Individuo 1 = [+,*,+,-,+]

Individuo 2 = [*,*,-,+,+]

Individuo 3 = [-,*,+,+,-]

Iteración 1:

1. Verificación de parada:

   Individuo 1 = | 15 - (2 + 4 * 1 + 6 - 3 + 5) | = | 15 – 14 | = 1

   Individuo 2 = | 15 – (2 * 4 * 1 - 6 + 3 + 5) | = | 15 – 10 | = 5

   Individuo 3 = | 15 – (2 - 4 * 1 + 6 + 3 - 5) | = | 15 – 2 | = 13

   El mejor individuo es: arg min(1,5,13) = individuo 1 con valor de solución 14.

   Como (valor_sol != objetivo) y (iteracion != 0), es decir, como (14 != 15) y (2 != 0) continuamos.

2. Selección:

   arg min(1,5,13) = Individuo 1

   arg min(5,13) = Individuo 2

   Por lo tanto los padres son:

   Padre 1 = [+,*,+,-,+]

   Padre 2 = [*,*,-,+,+]

3. Cruce: (Cruce uniforme)

Obtenemos de forma aleatoria la lista de padres de cada operador para el hijo 1, el hijo dos es la lista opuesta de padres del hijo 1.

lista hijo 1 = [2,1,1,1,2]

lista hijo 2 = [1,2,2,2,1]

Por lo tanto ya podemos definir a los dos hijos:

hijo 1 = [*,*,+,-,+]

hijo 2 = [+,*,-,+,+]

4. Mutación:

Mutación aleatoria sobre el hijo 1:

Se elige una posición aleatoria que va desde la posición 1 hasta la posición 5, imaginemos que es 2, y se elige un operador aleatorio entre los operadores que tenemos definidos, imaginemos que es -, por lo tanto el hijo 1 queda de la siguiente manera:

hijo 1 = [*,-,+,-,+]

Mutación aleatoria sobre el hijo 2:

Se elige una posición aleatoria que va desde la posición 1 hasta la posición 5, imaginemos que es 3, y se elige un operador aleatorio entre los operadores que tenemos definidos, imaginemos que es +, por lo tanto el hijo 2 queda de la siguiente manera:

hijo 2 = [+,*,+,+,+]

5. Remplazo:

Remplazamos los dos hijos obtenidos por los dos individuos de la población actual cuyos valores de función fitness son máximos.

arg max (1,5,13) = Individuo 3

arg max (1,5) = Individuo 2

Por lo tanto la población actual es:

Individuo 1 = [+,*,+,-,+]

Individuo 2 = [+,*,+,+,+]

Individuo 3 = [*,-,+,-,+]

Iteración 2:

1. Verificación de parada:

Individuo 1 = | 15 - (2 + 4 * 1 + 6 - 3 + 5) | = | 15 – 14 | = 1

Individuo 2 = | 15 – (2 + 4 * 1 + 6 + 3 + 5) | = | 15 – 20 | = 5

Individuo 3 = | 15 – (2 * 4 - 1 + 6 - 3 + 5) | = | 15 – 15 | = 0

El mejor individuo es: arg min(1,5,0) = individuo 3 con valor de solución 15.

Como (valor_sol == objetivo), es decir, como (15 == 15) finaliza la ejecución y devuelve el mejor individuo que es: [*,-,+,-,+]

8. Ejecuciones en Python

La implementación se ha realizado en Python2.7 cuyo código adjunto en el trabajo, a continuación vamos a realizar una serie de ejecuciones. Hemos de tener en cuenta que las ejecuciones no son deterministas, es decir, cada ejecución puede proporcionar resultados diferentes ya que cada generación de población inicial es diferente y las mutaciones realizadas en los hijos obtenidos también son aleatorias por lo tanto difieren en cada ejecución.

```
pascu@Acer ~/Escritorio $ python2.7 genetico.py
Introduce 6 numeros: (ej: 2 5 1 3 5 4)
12 26 4 30 15 8
Introduce un numero objetivo:
638
Introduce el numero de iteraciones:
200
------------------------------------------------------------
 SOLUCION:
------------------------------------------------------------
Individuo:  ['*', '-', '+', '*', '/']
Valor:  633.75
Iteracion:  200
------------------------------------------------------------
```

El programa nos pide como entrada la secuencia de seis números, un número objetivo y el número de iteraciones a realizar. En esta ejecución observamos que con 200 iteraciones nos proporciona el mejor individuo: [*,-,+,*,/] con un valor de 633.75.

Si volvemos a ejecutar el programa con los mismos parámetros observamos como hemos comentado anteriormente que el resultado puede ser distinto.

```
pascu@Acer ~/Escritorio $ python2.7 genetico.py
Introduce 6 numeros: (ej: 2 5 1 3 5 4)
12 26 4 30 15 8
Introduce un numero objetivo:
638
Introduce el numero de iteraciones:
200
------------------------------------------------------------
 SOLUCION:
------------------------------------------------------------
Individuo:  ['*', '*', '-', '/', '*']
Valor:  649.6
Iteracion:  200
------------------------------------------------------------
```

Con los mismos parámetros hemos obtenido otra solución, ya que en este caso el mejor individuo es [*,*,-,/,*] con un valor de 649.6 y también ha necesitado 200 iteraciones para obtenerlo.

Si aumentamos el número de iteraciones tenemos más posibilidad de encontrar una mejor solución, como observamos en la siguiente ejecución.

```
pascu@Acer ~/Escritorio $ python2.7 genetico.py
Introduce 6 numeros: (ej: 2 5 1 3 5 4)
12 26 4 30 15 8
Introduce un numero objetivo:
638
Introduce el numero de iteraciones:
100000
------------------------------------------------------------
 SOLUCION:
------------------------------------------------------------
Individuo:  ['*', '+', '*', '/', '+']
Valor:  640.0
Iteracion:  100000
------------------------------------------------------------
```

Por otra parte es interesante comentar que si queremos observar que sucede en cada iteración de la ejecución, he implementado una opción verbose que se ejecuta con el parámetro -v al ejecutar el programa, vemos la salida en la siguiente ejecución.

```
Introduce 6 numeros: (ej: 2 5 1 3 5 4)
2 8 7 1 4 6
Introduce un numero objetivo:
16
Introduce el numero de iteraciones:
2
Iteracion:  0
Secuencia de numeros:  [2.0, 8.0, 7.0, 1.0, 4.0, 6.0]
Objetivo:  16.0
Poblacion actual:  [['+', '-', '-', '*', '/'], ['/', '*', '+', '/', '/'], ['*', '-', '/', '+', '*']]
Soluciones actuales:  [1.3333333333333333, 0.11458333333333333, 78.0]
Funcion fitness:  [14.666666666666666, 15.885416666666666, 62.0]
Padres:  [['+', '-', '-', '*', '/'], ['/', '*', '+', '/', '/']]
Cruce (Uniforme):  [['/', '*', '+', '*', '/'], ['+', '-', '-', '/', '/']]
Mutaciones:  [['/', '*', '+', '*', '+'], ['+', '-', '-', '/', '-']]
Remplazo (Poblacion actual):  [['+', '-', '-', '*', '/'], ['+', '-', '-', '/', '-'], ['/', '*', '+', '*', '+']]
############################################################
Iteracion:  1
Secuencia de numeros:  [2.0, 8.0, 7.0, 1.0, 4.0, 6.0]
Objetivo:  16.0
Poblacion actual:  [['+', '-', '-', '*', '/'], ['+', '-', '-', '/', '-'], ['/', '*', '+', '*', '+']]
Soluciones actuales:  [1.3333333333333333, -5.5, 17.0]
Funcion fitness:  [14.666666666666666, 21.5, 1.0]
Padres:  [['/', '*', '+', '*', '+'], ['+', '-', '-', '*', '/']]
Cruce (Uniforme):  [['/', '-', '+', '*', '+'], ['+', '*', '-', '*', '/']]
Mutaciones:  [['-', '-', '+', '*', '+'], ['+', '*', '-', '*', '/']]
Remplazo (Poblacion actual):  [['+', '*', '-', '*', '/'], ['-', '-', '+', '*', '+'], ['/', '*', '+', '*', '+']]
############################################################
------------------------------------------------------------
 SOLUCION:
------------------------------------------------------------
Secuencia de numeros:  [2.0, 8.0, 7.0, 1.0, 4.0, 6.0]
Objetivo:  16.0
Poblacion actual:  [['+', '*', '-', '*', '/'], ['-', '-', '+', '*', '+'], ['/', '*', '+', '*', '+']]
Soluciones actuales:  [46.0, -42.0, 17.0]
Funcion fitness:  [30.0, 58.0, 1.0]
Individuo:  ['/', '*', '+', '*', '+']
Valor:  17.0
Iteracion:  2
------------------------------------------------------------
```