Centro de Investigación en Cómputo
Instituto Politécnico Nacional
Metaheurísticas
Actividad No. 14
Operadores de cruzamiento en Algoritmos Genéticos
Curso impartido por: Dra Yenny Villuendas Rey

Adrian González Pardo

25 de noviembre de 2020

1. Ventajas y Desventajas de GA

Ventajas	Desventajas
Permite realizar multiples busqueda de soluciones a los problemas	Puede que el metodo de mutación o cruza seleccionado puede que no ayude a encontrar una buena solución
Esta bioinspirado en la genética y en la selección natural (Darwinismo)	Puede que el que en la aplicación en alguna etapa del GA ya no avance
Es una heurística poblacional	Puede que genere bastante uso de recursos en memoria y procesamiento
Es posible el trabajar soluciones de formas paralelizables o distribuidas	Puede ser dificil de implementar

2. Genotipo vs Fenotipo

Genotipo: es una representación en cadenas de bits en la cual generalmente es trabajada para generar un nuevo individuo en el algoritmo.
Fenotipo: es la representación que tiene la cadena de bits en el ambito del problema, es decir, la cadena de bits puede representar números reales $\mathbb{R}$, números enteros $\mathbb{Z}$, valores binarios $\{0, 1\}$, indices de la solución a algún problema.

3. Modelo Generacional vs Estacionario

3.1. Semejanzas

- Ambos generan en cada iteración nuevas respuesta a analizar
- Ambos remplazan a la generación anterior (Con precaución de como son seleccionados y de como trabajan en la siguiente iteración).
- Ambos realizan conceptualmente las mismas operaciones (Solo que de diferente manera a la hora de selección y cruza).

3.2. Diferencias

- El modelo generacional crea una nueva población completa, mientra que el modelo estacionario escoje dos partes de la población de acuerdo al muestreo que realice y sobre ellos aplica los operadores genéticos.

- El modelo generacional remplaza completamente a la anterior generación, mientras que el modelo estacionario remplaza a los M cromosas con los N descendientes de la población inicial $M \leq N$.
- El modelo generacional tiene un remplazo aleatorio, mientras que el modelo estacionario remplaza a los N peores.
- El modelo generacional teorícamente realiza una excesiva exploración lo cual no garantiza que tenga una convergencia en un optimo local (Explora espacialmente las regiones de solución), el modelo estacionario realiza una excesiva explotación lo cual converge en un optimo local (Busca mejorar al mejor individuo).

4. Operadores de Cruzamiento

4.1. Un punto corte

Es una forma de intercambiar información entre dos padres. Esta forma de cruce se selecciona un punto aleatorio que va desde 1 hasta $N-1$ como un número aleatorio donde $N-1$ es la longitud del cromosoma, apartir del número aleatorio obtenido es donde se realizara el cruzamiento de datos por lo que se obtendra lo siguiente:

0	0	0	1	1	0	1	1
1	0	1	0	0	1	1	0

Pasan a:

0	0	0	1	0	1	1	0
1	0	1	0	1	0	1	1

4.2. Dos puntos de corte

Parecido al algoritmo de un punto de corte pero en este se seleccionan dos puntos aleatorios de corte de forma que se los cortes van de $[1, a], [a, b], [b, N-1]$, quedando un ejemplo de la siguiente forma:

0	0	0	1	1	0	1	1
1	0	1	0	0	1	1	0

Pasan a:

0	0	0	1	0	1	1	1
1	0	1	0	1	0	1	0

4.3. Uniforme

Este tipo de cruzamiento se realiza bajo un vector con valores de probabilidad, destacando que cada elemento del arreglo tiene una probabilidad de cruza del 0,5 por lo que generaran valores aleatorios en el vector de tal manera que si $v_i \geq 0{,}5$ este indice realiza la cruza de datos de ese indice.
Un ejemplo de ello seria el siguiente: **Se genera un vector con los siguientes datos**

0.55	0.03	0.67	0.77	0.30	0.25	0.99	0.78

Teniendo los siguientes padres:

0	0	0	1	1	0	1	1
1	0	1	0	0	1	1	0

Entonces:

1	0	1	0	1	0	1	0
0	0	0	1	0	1	1	1

4.4. Aritmético

Para este tipo de algoritmo es necesario pensar en que nuestros dos vectores padres son valores reales los cuales van a ser cruzados a traves de un valor aleatorio $\alpha \in [0, 1]$ de modo que actuaran como valores escalares para nuestros vectores, de tal forma que existe un segundo valor $\beta = 1 - \alpha$ formando la siguiente ecuación para realizar la cruza:

Sean los vectores

$$X^1 = x_0^1, x_1^1, \cdots, x_N^1$$
$$X^2 = x_0^2, x_1^2, \cdots, x_N^2$$
$$y = \alpha x^1 + \beta x^2$$

4.5. BL$X - \alpha$

Si bien el metodo de cruzamiento aritmético es un modelo lineal no siempre se conservan todos los genes, de modo que existe el algoritmo cruzamiento por mezcla el cual amplia el rango de cruce aritmético Para el gen x_i^1 y x_i^2, suponga que $x_i^1 < x_i^2$ i sin pérdida de generalidad; su gen descendiente es:

$$y_i = rand\left[\left(x_i^1 - \alpha\left(x_i^2 - x_i^1\right)\right), \left(x_i^2 - \alpha\left(x_i^1 - x_i^2\right)\right)\right]$$

Donde:
$rand(a, b)$ es una función para generar un número aleatorio uniformemente distribuido en el rango(a,b), α es un parámetro definido por el usuario que controla la extensión de la expansión.
Nota: el autor recomiendo el uso de $\alpha = 0{,}5$ *para cualquier caso.*

4.6. Simulated Binary Crossover (SBX)

Gracias a Deb y Agrawal que encontraron que la descendencia del cruce de un solo punto tiene el mismo centroide que el de los padres. Luego sugirieron un cruzamiento binario simulado (SBX) para simular esta propiedad en un cruce de código real.
Definieron un término llamado factor de propagación β_i para el gen i del código real de la siguiente manera:

$$\beta_i = \left|\frac{c_i^1 - c_i^2}{p_i^1 - p_i^2}\right|$$

Donde:
c_i^1 y c_i^2 son el gen i de los hijos y p_i^1 y p_i^2 son el gen de los padres.

Si $\beta_i < 1$, el operador se denomina cruce de contratación.

Si $\beta_i > 1$, el operador se denomina cruce de expansión.

Si $\beta_i = 1$, el operador se denomina cruce estacionario.

Para simular la propiedad de compartir centroide del cruce de un solo punto para el código numérico binario, es deseable que la descendencia sea simétrica al centroide de los dos padres.
Para el cruce de código real tenga una alta probabilidad de ser un cruce estacionario y una baja probabilidad de ser un cruce de contracción y un cruce de expansión para cada variable. Apartir de esto se define a β_i como un valor aleatorio de la siguiente manera:

$$p\left(\beta_i\right) = \begin{cases} 0{,}5(n+1)\beta_i^n, & \beta_i \leq 1 \\ \dfrac{0{,}5(n+1)}{\beta_i^{n+2}}, & \beta_i > 1 \end{cases}$$

Donde:
n es el parametro de control.
Entonces para generar el valor aleatorio de β_i podemos guiarnos de un parametro aleatorio $u_i \in [0, 1]$ de modo en que nos apoyamos de la siguiente formula

$$\beta_i = \begin{cases} \left(2u_i\right)^{\frac{1}{n+1}}, & u_i \leq 0{,}5 \\ \left(2\left(1 - u_i\right)\right)^{-\frac{1}{n+1}}, & u_i > 0{,}5 \end{cases}$$

Finalmente para encontrar los valores c_i^1 y c_i^2 obtendremos la siguiente formula.

$$c_i^1 = 0{,}5\left(p_i^1 + p_i^2\right) + 0{,}5\beta_i\left(p_i^1 - p_i^2\right)$$

$$c_i^2 = 0{,}5\left(p_i^1 + p_i^2\right) + 0{,}5\beta_i\left(p_i^2 - p_i^1\right)$$

5. Estructuras de datos necesarias para la implementación

Para la implementación de cualquiera de estos algoritmos simplemente nos podemos apoyar del arreglo de valores que nos devuelve la función que estamos trabajando y optimizando, de modo que podamos trabajar libremente con cada uno de los datos que deseamos cruzar, de modo en que se realicen vía indice evitando tener complicación con el tipo de dato que estemos manejando.

6. Implementación

```ruby
#!/usr/bin/env ruby

class Crossover
  attr_accessor :parent1,:parent2,:child1,:child2
  def initialize(parent1=[],parent2=[])
    """
      @parent1 y @parent2 ->  es un arreglo con los datos que vamos a utilizar
                              para la cruza de sus datos
    """
    @parent1=parent1
    @parent2=parent2
    @child1=[]
    @child2=[]
  end

  def uniforme()
    """
      array_p ->  Es el vector que contiene los valores probabilisticos de
                  realizacion de cruza o no de los datos, contiene valores
                  aleatorios de [0,1]
    """
    array_p=[]
    @child1=[]
    @child2=[]
    @parent1.each{|i|
      array_p.push(rand())
    }
    print "Probabilidades: ["
    array_p.each{|i|
      print " #{sprintf("%.5f",i)} "
    }
    puts "]"
    for i in 0..(@parent1.length-1)
      #puts "[#{i}] Parent 1: #{@parent1[i]}, Parent 2: "+
      #  "#{@parent2[i]}, probability: #{sprintf("%.5f",array_p[i])}"
      if array_p[i]>=0.5
        @child1.push(@parent2[i])
        @child2.push(@parent1[i])
      else
        @child1.push(@parent1[i])
        @child2.push(@parent2[i])
      end
    end
  end

  def aritmetic()
    """
      alpha ->  Es un valor aleatorio entre [0,1] que va a multiplicar a los
                valores reales de sus padre y asi realiza la cruza
    """
    @child1=[]
    @child2=[]
    alpha=0
    for i in 0..(@parent1.length-1)
      alpha=rand()
      child1.push(alpha*@parent1[i]+(1-alpha)*@parent2[i])
      child2.push(alpha*@parent2[i]+(1-alpha)*@parent1[i])
    end
  end
```

```
60
61    def single_crossover()
62      """
63        k ->  Es un valor entero el cual contendra el valor donde se realizara
64              el corte para la cruza de datos
65      """
66      @child1=[]
67      @child2=[]
68      k=rand(@parent1.length)
69      puts "Corte en indice #{k}"
70      @parent1.length.times{|i|
71        if i>=k
72          @child1.push(@parent2[i])
73          @child2.push(@parent1[i])
74        else
75          @child1.push(@parent1[i])
76          @child2.push(@parent2[i])
77        end
78      }
79    end
80
81  end
82
83  array1=[0,1,1,0,1]
84  array2=[1,0,2,0,0]
85  c=Crossover.new(array1,array2)
86  puts "Cruza uniforme"
87  c.uniforme()
88  puts "Padre 1:\t#{array1.to_s}\nPadre 2:\t#{array2.to_s}\n\n"+
89      "Hijo c1:\t#{c.child1.to_s}\nHijo c2:\t#{c.child2.to_s}\n\n"+
90      "Metodo aritmetico"
91  c.aritmetic
92  puts "Padre 1:\t#{array1.to_s}\nPadre 2:\t#{array2.to_s}\n\n"+
93      "Hijo c1:\t#{c.child1.to_s}\nHijo c2:\t#{c.child2.to_s}\n\n"+
94      "Metodo single crossover"
95  c.single_crossover()
96  puts "Padre 1:\t#{array1.to_s}\nPadre 2:\t#{array2.to_s}\n\n"+
97      "Hijo c1:\t#{c.child1.to_s}\nHijo c2:\t#{c.child2.to_s}"
```