Centro de Investigación en Cómputo
Instituto Politécnico Nacional
Metaheurísticas
Actividad No. 13
Solución de problemas mediante Algoritmos Genéticos GA
Curso impartido por: Dra Yenny Villuendas Rey

Adrian González Pardo

18 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 selección

4.1. Muestreo Aleatorio Universal (SUS)

El modelo de selección se trata de seleccionar varios elementos de acuerdo al tamaño de la población N dado que se obtiene una circunferencia donde existen varias flechas de selección dadas por $\frac{2\pi}{N}$ donde se busca girar todas las flechas uniformemente de tal manera que el giro y la selección se determina por $\left[rand \times \left(\frac{2\pi}{N}\right)\right]$ donde $rand \sim U(0,1)$, el cual al ser una parte universal elimina el sesgo probabilistico a la hora de seleccionar los elementos.

4.2. Torneo

Este modelo se trata de ir realizando un k número de arreglos permutados de forma aleatoria de tamaño N que es el tamaño dimensional de la respuesta de tal modo en que se busca seleccionar el valor indice de aquellos datos que esten mejor evaluados en el torneo, gráficamente el torneo se ve de la siguiente manera:
$k = 2$, entonces [0,5, 1, 4, 0,1], la generación aleatoria, generaria lo siguiente por ejempo:

2	1	4	3
4	2	3	1

Se accedera a los valores de cada subindice del arreglo en su valor y se evaluara el menor valor vía indices es decir de arriba hacia abajo para seleccionar al mejor.

4	1	4	1

4.3. Proporcional

Para este tipo de selección se usa la función

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$$

Donde:
p_i es la probabilidad de ser seleccionado p_i

f_i es la función de fitness para ese valor de i

Por tanto al obtener esto tendremos un pequeño sesgo probabilistico a la hora de seleccionar un área por tanto definiremos al error selectivo como:

$$n_i = p_i \times N = \left\lfloor N \times \frac{f_i}{\sum_{j=1}^{N} f_j} \right\rfloor$$

En esta ecuación nos podremos percatar de que redondeamos el error a un valor entero, lo que significa que cada individuo podría perderse cualquiera que sea su valor relativo de aptitud.

4.4. Por Ruleta

Para determinar este modelo se considera que la probabilidad para generar la circunferencia con ayuda de la ecuación p_i de la proporcional pero con la ayuda de la selección de 1 elemento a diferencia de como lo hace el Muestreo Universal que selecciona K elementos.

4.5. Por Emparejamiento Variado Inverso (NAM)

Un padre se escoge aleatoriamente, para el otro selecciona Nnam padres y escoge el más lejano al primer ($Nnam = 3, 5, \cdots$). Está orientado a generar diversidad.

5. Estructuras de datos necesarias para la implementación

Para obtener una mejor selección de nuestras soluciones es necesario hacer notar que muchas de estas soluciones son acompañadas del valor indice podemos pensar en un comportamiento de una cola circular de modo en que podamos realizar permutaciones y giros de acuerdo a como nos plasca, por otro lado tambien podemos pensar en hacer uso de arreglos de dimensión k donde $k \leq N$ de modo en que podemos crear una cantidad de arreglos y permutarlos de forma aleatoria para implementar grantidad de nuevos elementos, al menos para lograr implementar una selección por torneo, mientras que por otro lado podemos el obtener la cola donde cada indice sea un valor a la solución total podemos pensar en que este nos ayuda a implementar las funciones de probabilidad de modo que podemos ahorrarnos bastante tiempo a la hora de realizar las evaluaciones.

6. Implementación

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

class Seleccion
  attr_accessor :array,:table
  def initialize(array=[])
    """
      @array  ->  es el arreglo el cual contiene las evaluaciones de la funcion
                  a optimizar o es el arreglo de elementos para evaluar
    """
    @array=array
    @table={}
  end

  def sum_fitness()
    sum=0
    @array.length.times{|i|
      sum+=array[i]
    }
    sum
  end

  def gen_permutacion()
    permutacion=[]
    (@array.length).times{|i|
      permutacion.push(i)
    }

    (@array.length).times{
      cambia=rand(@array.length-1)
      cambia1=rand(@array.length-1)
      val0=permutacion[cambia]
      val1=permutacion[cambia1]
      permutacion[cambia1]=val0
      permutacion[cambia]=val1
    }
    puts "Permutacion de indices\t#{permutacion.to_s}"
    permutacion
  end

  def torneo(max_min=0)
    """
      @array  ->  son los valores de fitness de distintos valores a
                  ser evaluados por lo que en esta seccion compiten
    """
    k=rand(2..@array.length)
    puts "Valor de cuantas iteraciones #{k}"
    valores=[]
```

```
    for i in 0..k-1
      valores.push(gen_permutacion)
    end

    mejor=valores.pop
    valores.each{|val|
      (@array.length-1).times{|i|
        if max_min==0
          if @array[mejor[i]]>@array[val[i]]
            mejor[i]=val[i]
          end
        else
          if @array[mejor[i]]<@array[val[i]]
            mejor[i]=val[i]
          end
        end
      }
    }
    @table={}
    mejor.each{|i|
      if table.key?(i)
        table[i]+=1
      else
        table[i]=1
      end
    }
    @table=@table.sort_by {|k, v| -v}
    @table[0][0]
  end

  def proporcional()
    """
      @array  ->  en esta intervienen los fitness evaluados para ver cuales
                  elementos son considerados para su seleccion
                  (por la forma que tienen) es mejor que sea para una
                  maximizacion
    """
    sum_t=sum_fitness()
    pi=[]
    @array.length.times{|i|
      pi.push(@array[i].to_f/sum_t)
    }
    #print "Parte proporcional de cada pi ["
    #pi.each{|i|
    #  print " #{sprintf("%.4f",i)} "
    #}
    #puts "]"
    puts "Intervalos de la seleccion proporcional:"
    sum=0
    pi.each{|i|
      puts "\t\t[#{sprintf("%.4f",sum)} a #{sprintf("%.4f",sum+i)}]"
      sum+=i
    }
    arrow_init=rand(0.0..1.0)
    puts "Elemento seleccionado #{arrow_init}"
    mejor_respuesta=[]
    sum_acumulada=0
    pi.length.times{|j|
      sum_acumulada+=pi[j]
      if sum_acumulada>=arrow_init
        return j
      end
    }
  end


  def ruleta()
    """
      @array  ->  en esta cada elemento tiene la misma probabilidad de ser seleccionados
    """
    pi=[]
```

```
    piece=1.0/@array.length
    puts "Tamanio de la separacion de las piezas: #{sprintf("%.4f",piece)}"
    puts "Intervalos de la ruleta"
    @array.length.times{|i|
      pi.push(piece)
      puts "\t\t[#{sprintf("%.4f",piece*i)},#{sprintf("%.4f",piece*(i+1))}]"
    }
    arrow_init=rand(0.0..1.0)
    puts "Elemento seleccionado #{arrow_init}"

    sum_acumulada=0
    pi.length.times{|j|
      sum_acumulada+=pi[j]
      if sum_acumulada>=arrow_init
        return j
      end
    }
  end

  def table_cost()
    # Modificacion de llenar matriz de distancia por costo computacional
    for i in 0..(@array.length-1)
      for j in (i+1)..(@array.length-1)
        arreglo1=@array[i]
        arreglo2=@array[j]
        sum=0
        l=0
        for l in 0..(arreglo1.length-1)
          sum+=(arreglo2[l]-arreglo1[l]).abs
        end
        @table["#{i},#{j}"]=sum
      end
    end
  end

  def nam()
    """
      @array  ->  es el arreglo de valores de cada solucion por lo
                  que es necesario sacar las distancias de cada uno para
                  entregar dos valores cuya distancia sea la mejor
    """
    puts "Arreglo:\n\t#{@array.to_s}"

    valores_d=[]
    puts "Tabla de distancias\n\t#{table.to_s}"
    for i in 0..(table.length-1)
      for j in (i+1)..(table.length-1)
        valores_d.push(table.values[i]-table.values[j])
      end
    end
    seleccionados=[]
    valores_d.each_with_index{|i,j|
      if i!=1 && i!=-1
        seleccionados.push(table.keys[j].split(",",-1).map(&:to_i))
      end
    }
    seleccionados
  end

end

array_fitness=[2,5,6,1,4,10]
puts "Fitness function\t#{array_fitness.to_s}"

s=Seleccion.new(array_fitness)

mejor_torneo=s.torneo
puts "\nElemento seleccionado del torneo\t#{mejor_torneo}\n\n"

ruleta=s.ruleta()
puts "\nMejor elemento por ruleta\t#{ruleta.to_s}\n\n"
```

```
190
191 prop=s.proporcional
192 puts "\nMejor elemento por proporcional\t#{prop}\n\n"
193
194 array_fitness=[ [2,5,6,1,4,10], [6,7,1,4,5,6], [10,10,10,10,10,10] ]
195 s.array=array_fitness
196 s.table={}
197 s.table_cost()
198 na=s.nam()
199 puts "Indices seleccionados\t#{na.to_s}"
```