Edgar Joel Arévalo Chavarín 215861967

Seminario de Solución de Problemas de InteligenciaArtificial I

Practica 2

Introduccion:

El algoritmo genético es una técnica de búsqueda basada en la teoría de la evolución de Darwin, que ha cobrado tremenda popularidad en todo el mundo durante los últimos años. Se presentarán aquí los conceptos básicos que se requieren para abordarla, así como unos sencillos ejemplos que permitan a los lectores comprender cómo aplicarla al problema de su elección.

En los últimos años, la comunidad científica internacional ha mostrado un creciente interés en una nueva técnica de búsqueda basada en la teoría de la evolución y que se conoce como el **algoritmo genético**. Esta técnica se basa en los mecanismos de selección que utiliza la naturaleza, de acuerdo a los cuales los individuos más aptos de una población son los que sobreviven, al adaptarse más fácilmente a los cambios que se producen en su entorno. Hoy en día se sabe que estos cambios se efectúan en los genes de un individuo (unidad básica de codificación de cada uno de los atributos de un ser vivo), y que sus atributos más deseables (i.e., los que le permiten adaptarse mejor a su entorno) se transmiten a sus descendientes cuando éste se reproduce sexualmente.

Un investigador de la Universidad de Michigan llamado John Holland era consciente de la importancia de la selección natural, y a fines de los 60s desarrolló una técnica que permitió incorporarla a un programa. Su objetivo era lograr que las computadoras aprendieran por sí mismas. A la técnica que inventó Holland se le llamó originalmente "planes reproductivos", pero se hizo popular bajo el nombre "algoritmo genético" tras la publicación de su libro en 1975.

Una definición bastante completa de un algoritmo genético es la propuesta por John Koza:

"Es un algoritmo matemático altamente paralelo que transforma un conjunto de objetos matemáticos individuales con respecto al tiempo usando operaciones modeladas de acuerdo al principio Darwiniano de reproducción y supervivencia del más apto, y tras haberse presentado de forma natural una serie de operaciones genéticas de entre las que destaca la recombinación sexual. Cada uno de estos objetos matemáticos suele ser una cadena de caracteres (letras o números) de longitud fija que se ajusta al

modelo de las cadenas de cromosomas, y se les asocia con una cierta función matemática que refleja su aptitud.

Desarrollo:

La creación de los algoritmos geneticos se representa en el siguiente algoritmo.

```
def initialize(funcion, limit_inf, limit_sup, mutator_max = 0.4, poblation = 50, generations = 10, dimensions = 2)
  # function needed data

  # function that will be evaluated
  @function = funcion
  # lower limit of the function
  @limit_inf = limit_inf
  # upper limit of the function
  @limit_sup = limit_sup
  # number of dimensions of the function
  @dimensions = dimensions

  # data needed for the GA

  # poblation number
  @poblation = poblation
  # number of generations that will be made
  @generations = generations
  # probability to mutate for a son
  @mutator_probability = Random.rand(0.0..mutator_max)


  # value to compare initialize as infinite
```

```
f_best = +1.0/0.0
f_val = +1.0/0.0
x_best = +1.0/0.0
y_best = +1.0/0.0
# vector for every individual
@father_x = []
@father_y = []

# son initialition as emply array
@son_x = []
@son_y = []



# pobation initialition
poblationInit

puts "mutator_probability"
puts @mutator_probability
# repeat until all the generations were complete
1.step(@generations) do |generation|
  # sum of the aptitude of all the poblation
  @poblation_aptitud = 0
  # aptitud vectore, who aptitude is every father
  @aptitude = []
  # probability to be chose as a father for every individual in the poblation
  @elector_probability = []

  # calculate aptitud
```

aptitud

```
while @son_x.size < @poblation do
  # made rulet selection
  a = rulet
  # generate sons
  generateSons(a)
end
# puts @son_x.select{|i| i.nil?}.size

# mutate sons
mutate
@father_x = @son_x.dup
@father_y = @son_y.dup

# clear sons
@son_x.clear
@son_y.clear

@father_x.each_with_index.map do |value, i|
  f_val = f(value, @father_y[i])
  # puts f_val
  if(f_val < f_best)
    # puts i
    x_best = value
    y_best = @father_y[i]
    f_best = f_val
```

```
    end


   end
  end
  {best: {x: x_best, y: y_best}, f_val: f_best}
end
```

Donde esta funcion devolvera los mejores candidatos despues de realizar todas las generaciones, para definir cada uno de los metodos tales como inicializar la población calcular su aptitud, someterlos a sorteo por ruleta, generar los hijos y mutar estos, se explican a continuación.

- Inicializar la población

```
def poblationInit
  0.step(@poblation - 1) do |i|
    @father_x << @limit_inf + ( (@limit_sup - @limit_inf) * Random.rand(0.0..1.0) )
    @father_y << @limit_inf + ( (@limit_sup - @limit_inf) * Random.rand(0.0..1.0) )
  end
end
```

- Calcular aptitud

```
def aptitud
  0.step(@poblation - 1) do |i|
    value = f(@father_x[i], @father_y[i])
    @aptitude << aptitudEvalution(value)
  end
  @poblation_aptitud = @aptitude.reduce{ |a,b| a + b }
end


def aptitudEvalution(function_value)
  if (function_value < 0)
    1 + function_value.abs
```

```
  else
    1 / (1 + function_value)
  end
end
```

- Someter a sorteo por ruleta para seleccionar los 2 padres

```
def rulet
  @elector_probability = @aptitude.map{|value| ruletValue(value)}
  i = selection
  j = selection
  while (i == j)
    j = selection
  end
  {father_1: i, father_2: j}
end

def ruletValue(evaluted)
  evaluted / @poblation_aptitud
end

def selection
  p_sum = 0
  index = -1
  # probability need to be chosen as father
  selector = Random.rand(0.0..1.0)

  @elector_probability.each_with_index.map do |value, i|
    p_sum += value
    if (p_sum >= selector)
```

```
      index = i
      break
    end
  end
  index = @elector_probability.size if index > @elector_probability.size
  index
end
```

- generar por cada pareja de padres 2 hijos

```
def generateSons(a)
  pc = Random.rand(1..@dimensions)
  father_1 = a[:father_1]
  father_2 = a[:father_2]
  if(pc == 1)
    @son_x << @father_x[father_1]
    @son_y << @father_y[father_2]

    @son_x << @father_x[father_2]
    @son_y << @father_y[father_1]
  else
    @son_x << @father_x[father_2]
    @son_y << @father_y[father_2]

    @son_x << @father_x[father_1]
    @son_y << @father_y[father_1]
  end
end
```

- Mutación

```
def mutate
```

```
  0.step(@poblation - 1) do |j|
    0.step(@dimensions - 1) do |i|
    ra = Random.rand(0.0..1.1)
    rb = Random.rand(0.0..1.1)
    if(ra < @mutator_probability)
      mutate = @limit_inf + ( (@limit_sup - @limit_inf) * rb )
      if (@dimensions % 2 == 0)
        @son_x[j] = mutate
      else
        @son_y[j] = mutate
      end
    end


  end
end


end
```

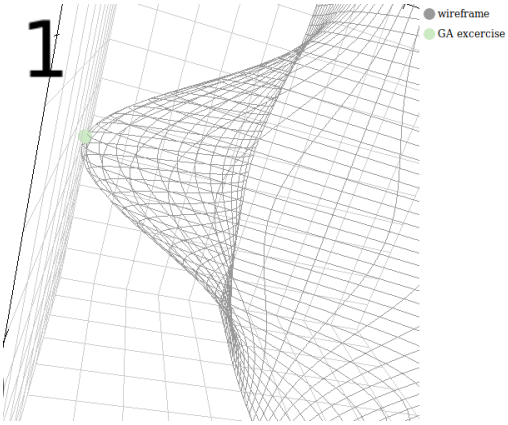
Para realizarlos bastaba con seguir el algoritmo a detalle.

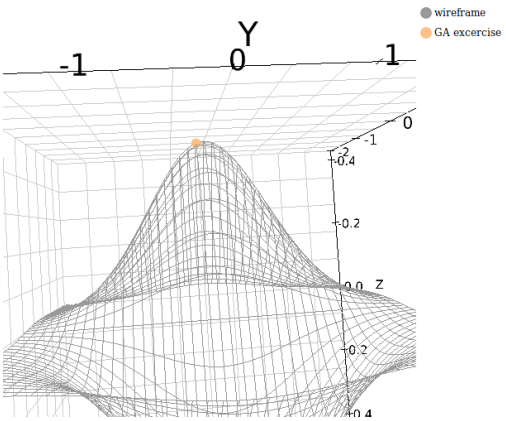
Resultados:

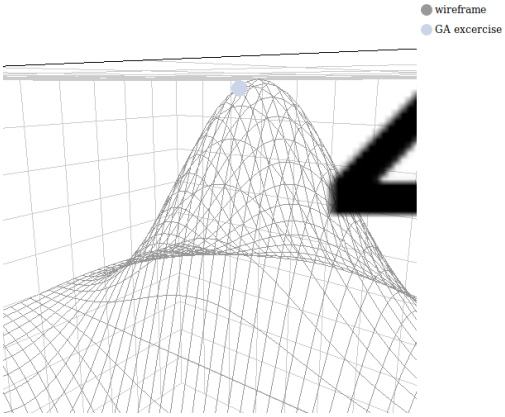
$f(x,y)=xe^{-x^2-y^2}; x,y\in[-2,2]$

x	y	f(x,y)	Factor de mutación
-0.7277846222927351	0.0683258668979354	-0.4265229925783700	60.6033803110601691
-0.721153671693775	-0.090210251391539	-0.4252391615913226	30.10729542885510523
-0.5969666192123981	-0.093240747487093	-0.4143861282838221	0.38664991364582

Edgar Joel Arévalo Chavarín 215861967

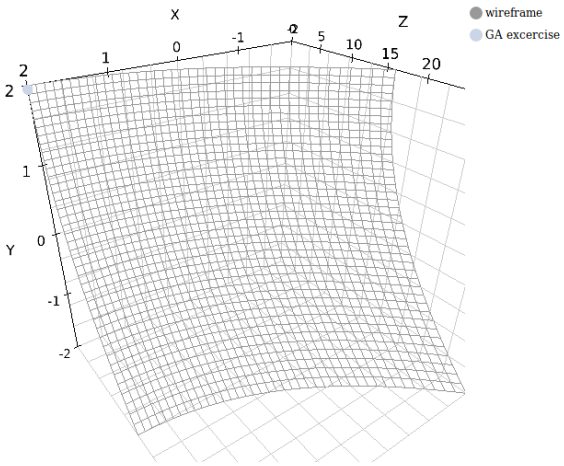


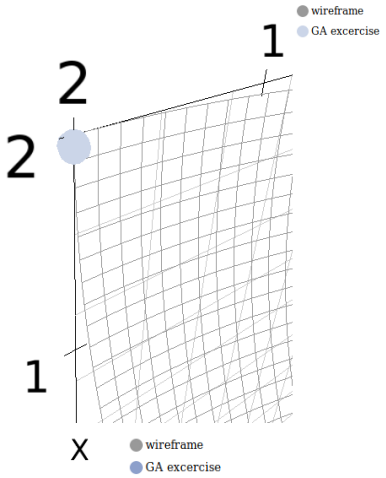


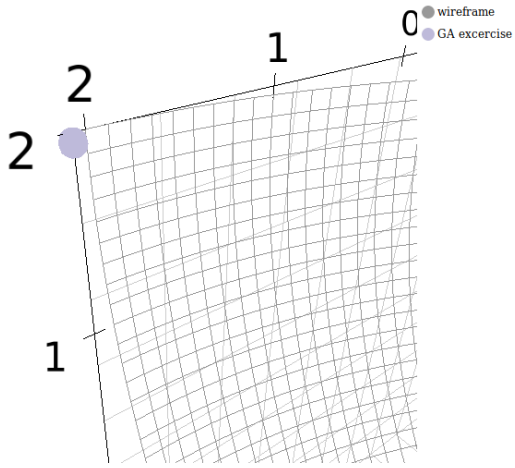


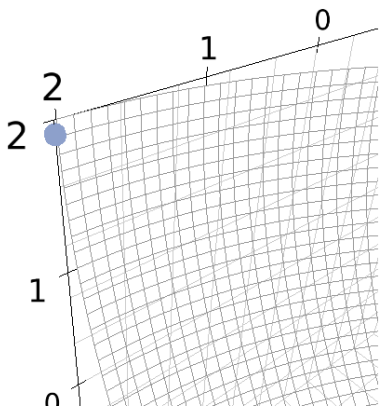
$$f(x)=\sum_{i=1}^{d}(x_i-2)^2, d=2$$

x	y	f(x,y)	Factor de mutación
2.0011734979748894	1.9560622202890308	0.0019319055834267295	0.8064519702122719
2.056051784395609	1.957265612539281	0.004968030405574683	1.3416032384577834
2.012866102049319	1.9178374394929434	0.006916222931019231	1.4483392740521825









Conclución

¿El método GA es mejor que los métodos tradicionales? ¿Por qué?

Dependiendo del problema, ya que no necesitan conocimientos específicos sobre el problema que intentan resolver, lo cual le da gran adaptivilidad a casi cualquier problema.

- Operan de forma simultánea con varias soluciones, en vez de trabajar de forma secuencial como las técnicas tradicionales.
- Cuando se usan para problemas de optimización maximizar una función objetivo- resultan menos afectados por los máximos locales (falsas soluciones) que las técnicas tradicionales.
- Resulta sumamente fácil ejecutarlos en las modernas arquitecturas masivamente paralelas.
- Usan operadores probabilísticos, en vez de los típicos operadores determinísticos de las otras técnicas.
- Pueden tardar mucho en converger, o no converger en absoluto, dependiendo en cierta medida de los parámetros que se utilicen tamaño de la población, número de generaciones, etc.-.
- Pueden converger prematuramente debido a una serie de problemas de diversa índole.

El poder de los Algoritmos Genéticos proviene del hecho de que se trata de una técnica robusta, y pueden tratar con éxito una gran variedad de problemas provenientes de diferentes áreas, incluyendo aquellos en los que otros métodos encuentran dificultades. Si bien no se garantiza que el Algoritmo Genético encuentre la solución óptima, del problema, encuentran soluciones de un nivel aceptable, en

un tiempo competitivo con el resto de algoritmos de optimización combinatoria. Sin embargo, en el caso de que existan técnicas especializadas para resolver un determinado problema, lo más probable es que superen al Algoritmo Genético, tanto en rapidez como en eficacia, por lo que en esos casos o otros en los que el problema sea absolutamente simple seria un gran desperdicio de recursos utilizando algoritmos geneticos.